Learning HTML 3.2 by Examples

Table of Contents

Learning HTML 3.2 by Examples 1
Preface 1
To whom? Previous knowledge needed? 1
About what? What's HTML 3.2? 2
Why should you learn HTML? 2
But why HTML 3.2? 2
The scope of this document 4
On the versions of this document 4
Best viewed on... 5
Copyright notice 5
How to study HTML 3.2 5
Getting started with HTML in general 5
Learning HTML 3.2 systematically 6
The official HTML 3.2 specification 7
Additional sources of information 7
Checking your HTML 8
General remarks on the syntax of HTML 10
Character set 10
HTML tags 11
HTML elements 11
Attributes 12
URLs 13
Absolute URLs 13
Notes and warnings 14
Relative URLs 14
Fragment identifiers 15
More information about URLs 15
URL encodings, or what to do e.g. with spaces 15
Case sensitivity 16
Division into lines and the use of blanks and tabs 16
Classification of elements 18
Allowed nesting of elements 19
Miscellaneous notes: about escape sequences (character entities), names, colors, widths, pixels, non-breaking spaces ( ), comments 21
Escape sequences (character entities) 21
Names 22
Colors 23
Widths 24
Pixels 24
Non-breaking spaces ( ) 25
Comments 26
Media types 26
Fundamental structures in HTML 3.2, with examples 27
The obligatory structure of a document 27
The recommended structure of a document 28

Information about the document - the HEAD section 29
Organizing the contents - headings, paragraphs, lists, etc 30
Text markup - emphasis, citations, code, etc 32
Logical vs physical markup 32
Phrase elements (logical text markup) 33
Font elements (physical text markup) 34
Rendering of markup 34
Presenting interaction with computer 35
Controlling the layout 37
Links 37
The link concept 37
Normal links 38
Expressing the nature of a link 38
Link types (REL values) 39
Practical guidelines for setting up normal links 40
Audio and video resources 41
Links to binary and plain text files 41
Links to other non-HTML resources 42
Images, formulas, etc. 42
Tables (Not in HTML 2.0!) 45
The table concept in HTML 3.2 45
Tags used to represent tables 46
The very basic table structure 46
Additional features; a typical table with text cells 47
Parallel texts 47
Using a table to present a definition list 48
Numerical tables 48
Using tables to represent menus 49
Pseudo-table: preformatted text 50
Using just characters as separators 50
A flexible pseudo-table 51
Table elements occupying several rows or columns 52
Nested tables 52
Alignment of cells 53
Fonts in table elements 54
Style sheets 55
Descriptions of HTML 3.2 tags 56
Index and legend 56
A - anchors, hyperlinks, etc 56
Purpose 56
Typical rendering 57
Basic syntax 57
Possible attributes 57
Allowed context 58
Contents 58
Examples 58
Notes 59
ADDRESS - document author information 60
Purpose 60
Typical rendering 60

Basic syntax 60
Possible attributes 60
Allowed context 60
Contents 60
Examples 60
Notes 61
APPLET - Java applets (Not in HTML 2.0!) 61
Purpose 61
Typical rendering 62
Basic syntax 62
Possible attributes 62
Allowed context 63
Contents 64
Examples 64
Notes 65
AREA - area in a clickable map (Not in HTML 2.0!) 65
Purpose 65
Typical rendering 65
Basic syntax 66
Possible attributes 66
Allowed context 67
Contents 67
Examples 67
Notes 67
B - bolding 67
Purpose 67
Typical rendering 67
Basic syntax 68
Possible attributes 68
Allowed context 68
Contents 68
Examples 68
Notes 68
BASE - base for URLs 68
Purpose 68
Typical rendering 69
Basic syntax 69
Possible attributes 69
Allowed context 69
Contents 69
Example 69
Notes 69
BASEFONT - base font size (Not in HTML 2.0!) 70
Purpose 70
Typical rendering 70
Basic syntax 70
Possible attributes 70
Allowed context 70
Contents 70
Examples 71

Notes 71
BIG - big font (Not in HTML 2.0!) 71
Purpose 71
Typical rendering 71
Basic syntax 71
Possible attributes 71
Allowed context 71
Contents 71
Examples 72
Notes 72
BLOCKQUOTE - long quotation 72
Purpose 72
Typical rendering 72
Basic syntax 72
Possible attributes 72
Allowed context 72
Contents 72
Examples 73
Notes 73
BODY - document body 74
Purpose 74
Typical rendering 74
Basic syntax 74
Possible attributes (Not in HTML 2.0!) 74
Allowed context 75
Contents 75
Examples 75
Notes 76
BR - line break 76
Purpose 76
Typical rendering 76
Basic syntax 77
Possible attributes (Not in HTML 2.0!) 77
Allowed context 77
Contents 77
Examples 77
Notes 77
CAPTION - caption for a table (Not in HTML 2.0!) 78
Purpose 78
Typical rendering 78
Basic syntax 78
Possible attributes 78
Allowed context 78
Contents 78
Examples 78
Notes 79
CENTER - centering (Not in HTML 2.0!) 79
Purpose 79
Typical rendering 79
Basic syntax 79

Possible attributes 79
Allowed context 79
Contents 79
Examples 79
Notes 80
CITE - citations 80
Purpose 80
Typical rendering 80
Basic syntax 80
Possible attributes 80
Allowed context 80
Contents 81
Examples 81
Notes 81
CODE - program code 81
Purpose 81
Typical rendering 81
Basic syntax 81
Possible attributes 81
Allowed context 82
Contents 82
Examples 82
Notes 82
DD - definition data 82
Purpose 82
Typical rendering 82
Basic syntax 82
Possible attributes 82
Allowed context 83
Contents 83
Examples 83
Notes 83
DFN - defining occurrence (Not in HTML 2.0!) 83
Purpose 83
Typical rendering 83
Basic syntax 83
Possible attributes 83
Allowed context 83
Contents 84
Examples 84
Notes 84
DIR - unnumbered list in directory-like form 84
Purpose 84
Typical rendering 84
Basic syntax 84
Possible attributes 84
Allowed context 85
Contents 85
Examples 85
Notes 85

DIV - document division (Not in HTML 2.0!) 85
Purpose 85
Typical rendering 86
Basic syntax 86
Possible attributes 86
Allowed context 86
Contents 86
Examples 86
Notes 87
DL - definition list 87
Purpose 87
Typical rendering 87
Basic syntax 87
Possible attributes 87
Allowed context 88
Contents 88
Examples 88
Notes 88
DT - definition term 89
Purpose 89
Typical rendering 89
Basic syntax 89
Possible attributes 89
Allowed context 89
Contents 89
Examples 89
EM - emphasis 89
Purpose 89
Typical rendering 89
Basic syntax 90
Possible attributes 90
Allowed context 90
Contents 90
Examples 90
Notes 90
FONT - font size and color (Not in HTML 2.0!) 90
Purpose 90
Typical rendering 90
Basic syntax 91
Possible attributes 91
Allowed context 91
Contents 91
Examples 91
Notes 91
FORM - fill-out form 92
Purpose 92
Typical rendering 92
Basic syntax 92
Possible attributes 93
Allowed context 93

Contents 93
Examples 93
Notes 96
H1, H2, H3, H4, H5, H6 - headings 97
Purpose 97
Typical rendering 97
Basic syntax 97
Possible attributes (Not in HTML 2.0!) 98
Allowed context 98
Contents 98
Examples 98
Notes 98
HEAD - document head 99
Purpose 99
Typical rendering 99
Basic syntax 99
Possible attributes 99
Allowed context 99
Contents 99
Examples 99
Notes 100
HR - change in topic (horizontal rule) 100
Purpose 100
Typical rendering 100
Basic syntax 100
Possible attributes **(Not in HTML 2.0!)** 100
Allowed context 100
Contents 100
Examples 101
Notes 101
HTML - the top-level element in HTML 101
Purpose 101
Typical rendering 101
Basic syntax 101
Possible attributes 102
Allowed context 102
Contents 102
Examples 102
Notes 102
I - text in italics 102
Purpose 102
Typical rendering 102
Basic syntax 102
Possible attributes 103
Allowed context 103
Contents 103
Examples 103
Notes 103
IMG - inline images 103
Purpose 104

Typical rendering 104
Basic syntax 104
Possible attributes 104
Allowed context 107
Contents 107
Examples 107
Notes 107
INPUT - input fields in forms 109
Purpose 109
Typical rendering 109
Basic syntax 109
Possible attributes 109
Allowed context 114
Contents 114
Examples 114
Notes 114
ISINDEX - simple keyword searches 114
Purpose 114
Basic syntax 115
Typical rendering 115
Possible attributes (Not in HTML 2.0!) 115
Allowed context 115
Contents 115
Examples 115
Notes 115
KBD - keyboard input 116
Purpose 116
Typical rendering 116
Basic syntax 116
Possible attributes 116
Allowed context 116
Contents 116
Examples 116
Notes 116
LI - list item 117
Purpose 117
Typical rendering 117
Basic syntax 117
Possible attributes (Not in HTML 2.0!) 117
Allowed context 118
Contents 118
Examples 118
Notes 118
LINK - relationships with other documents 118
Purpose 118
Typical rendering 118
Basic syntax 119
Possible attributes 119
Allowed context 119
Contents 119

Examples 119
Notes 119
MAP - clickable map (Not in HTML 2.0!) 120
Purpose 120
Typical rendering 120
Basic syntax 120
Possible attributes 120
Allowed context 120
Contents 120
Examples 120
Notes 121
MENU - unnumbered list in menu-like form 121
Purpose 121
Typical rendering 121
Basic syntax 121
Possible attributes 121
Allowed context 121
Contents 122
Examples 122
Notes 122
META - meta info 122
Purpose 122
Typical rendering 122
Basic syntax 122
Possible attributes 123
Allowed context 123
Contents 123
Examples 123
Notes 123
OL - ordered (numbered) list 124
Purpose 124
Typical rendering 124
Basic syntax 125
Possible attributes 125
Allowed context 125
Contents 125
Examples 125
Notes 126
OPTION - an option in a select menu 127
Purpose 127
Typical rendering 127
Basic syntax 127
Possible attributes 127
Allowed context 128
Contents 128
Examples 128
P - normal paragraph 128
Purpose 128
Typical rendering 128
Basic syntax 128

Possible attributes (Not in HTML 2.0!) 128
Allowed context 129
Contents 129
Examples 129
Notes 129
PARAM - applet parameters (Not in HTML 2.0!) 130
Purpose 130
Typical rendering 130
Basic syntax 130
Possible attributes 130
Allowed context 130
Contents 130
Examples 131
Notes 131
PRE - preformatted text 131
Purpose 131
Typical rendering 131
Basic syntax 131
Possible attributes 131
Allowed context 132
Contents 132
Examples 132
Notes 133
SAMP - sample output 134
Purpose 134
Typical rendering 134
Basic syntax 134
Possible attributes 134
Allowed context 134
Contents 134
Examples 134
Notes 134
SCRIPT - client-side scripting languages (Not in HTML 2.0!) 135
Purpose 135
Typical rendering 135
Basic syntax 135
Possible attributes 135
Allowed context 135
Contents 135
Examples 136
Notes 136
SELECT - menu in a form 136
Purpose 136
Typical rendering 136
Basic syntax 136
Possible attributes 136
Allowed context 137
Contents 137
Examples 137
Notes 137

SMALL - small font (Not in HTML 2.0!) 137
Purpose 137
Typical rendering 137
Basic syntax 137
Possible attributes 137
Allowed context 138
Contents 138
Examples 138
Notes 138
STRIKE - strike-through text (Not in HTML 2.0!) 139
Purpose 139
Typical rendering 139
Basic syntax 139
Possible attributes 139
Allowed context 139
Contents 139
Examples 139
Notes 140
STRONG - strong emphasis 140
Purpose 140
Typical rendering 140
Basic syntax 140
Possible attributes 140
Allowed context 140
Contents 141
Examples 141
Notes 141
STYLE - style sheets (Not in HTML 2.0!) 141
Purpose 141
Typical rendering 141
Basic syntax 141
Possible attributes 141
Allowed context 141
Contents 141
Examples 142
Notes 142
SUB - subscript (Not in HTML 2.0!) 142
Purpose 142
Typical rendering 142
Basic syntax 142
Possible attributes 143
Allowed context 143
Contents 143
Examples 143
Notes 143
SUP - superscript (Not in HTML 2.0!) 144
Purpose 144
Typical rendering 144
Basic syntax 144
Possible attributes 144

Allowed context 144
Contents 144
Examples 144
Notes 145
TABLE - tables (Not in HTML 2.0!) 145
Purpose 145
Typical rendering 145
Basic syntax 146
Possible attributes 146
Allowed context 146
Contents 146
Examples 147
Notes 147
TD - table data (cell) (Not in HTML 2.0!) 148
Purpose 148
Typical rendering 148
Basic syntax 148
Possible attributes 148
Allowed context 149
Contents 149
Examples 150
Notes 150
TEXTAREA - multi-line text input in a form 150
Purpose 150
Typical rendering 150
Basic syntax 150
Possible attributes 150
Allowed context 151
Contents 151
Examples 151
Notes 151
TH - table heading (cell) (Not in HTML 2.0!) 152
Purpose 152
Typical rendering 152
Basic syntax 152
Possible attributes 152
Allowed context 153
Contents 153
Examples 154
Notes 154
TITLE - "external" title 154
Purpose 154
Typical rendering 154
Basic syntax 154
Possible attributes 154
Allowed context 154
Contents 154
Example 155
Notes 155
TR - table row (Not in HTML 2.0!) 155

Purpose 155
Typical rendering 155
Basic syntax 155
Possible attributes 155
Allowed context 156
Contents 156
Examples 156
Notes 156
TT - teletype (monospaced) text 156
Purpose 156
Typical rendering 156
Basic syntax 156
Possible attributes 156
Allowed context 156
Contents 156
Examples 156
Notes 157
U - underline (Not in HTML 2.0!) 157
Purpose 157
Typical rendering 157
Basic syntax 157
Possible attributes 157
Allowed context 157
Contents 157
Examples 157
Notes 157
UL - unnumbered list 158
Purpose 158
Typical rendering 158
Basic syntax 158
Possible attributes 158
Allowed context 158
Contents 158
Examples 159
Notes 159
VAR - variables 159
Purpose 159
Typical rendering 159
Basic syntax 160
Possible attributes 160
Allowed context 160
Contents 160
Examples 160
Notes 160

Learning HTML 3.2 by Examples

A short description of this document is available.

Jukka Korpela

Preface

To whom? Previous knowledge needed?

This document is intended for people who *have an idea of what the World Wide Web is* and who produce, or intend to produce, information onto the Web. That is, this is for you if you have surfed on the Web and wish to produce Web pages. If HTML is something entirely new to you, you may need to study some introductory texts (e.g. those mentioned in this document) before you can really benefit from this document.

No specific previous knowledge is required for learning HTML. In particular, HTML authoring is **not programming**, and HTML is not a programming language. In HTML, you basically markup (annotate) the structure of a document, by indicating which texts constitute headings, or are to be emphasized, or form lists, or refer (link) to other documents, etc.

If you know *French, German, Italian, Portuguese or Spanish* better than English and if you find some sentences in this document difficult to understand, you might use the AltaVista Translation Service, if you are working online. Depending on your Web browser, you might even be able to follow the link above so that the translation service opens in a new window, then just use cut & paste to get translations. (You could then leave the new window on the screen for later use.) You may wish to test the translation service, also called "Babelfish", using the following form:

Provide either plain text or the address (URL) of a Web page to translate:

Translate from English to

- ◇ French
- ◇ German
- ◇ Italian
- ◇ Portuguese
- ◇ Spanish

You can also and type again.

Please notice that the translation service doesn't yet translate technical Web-related terms well. Therefore, you may need to consult some specialized dictionary of Internet terms.

On the other hand, this document tries to define the *technical terms* it uses or to provide links to definitions. If you find terms which are unknown to you and not defined here, please consult e.g. the *Terms* section of HTML 2.0 specification or some of the general Internet glossaries. (The most authoritative Internet glossary is probably RFC 1983.) Last but not least, there is a very good page about Internet glossaries at WebReference.

About what? What's HTML 3.2?

This document discusses HTML 3.2, a version of the document description language HTML used on the Web. Its authoritative definition is W3C Recommendation *HTML 3.2 Reference Specification*. It is also known under the code name **Wilbur**.

People who have heard about HTML 3.0 should notice that HTML 3.2 is **not** an extension or a variant of it. (The version numbers 3.0 and 3.2 are misleading!) HTML 3.0 which was a *draft* which expired 1995, mainly due to major browser vendors' unwillingness to implement it, though vendors claimed conformance to it long thereafter!

More exactly, HTML 3.2 contains

- HTML 2.0 (with a couple of minor omissions)
- some features from HTML 3.0, partially restricted or otherwise modified; this in particular applies to tables
- some vendor extensions upon which an agreement was found.

For a good summary of the new features in HTML 3.2 as compared with HTML 2.0, consult the article *What's New in HTML 3.2* in the *World Wide Web Journal*, but please notice that it contains a few mistakes.

Why should you learn HTML?

It is possible to provide information on the Web without knowing the HTML language, since HTML can be produced by various specialized editors and converters. This document, however, was written for people who write HTML directly or at least occasionally check and modify HTML code. There are several good reasons to do so.

Writing HTML directly isn't difficult - possibly it's easier than learning to use an HTML editor or converter. Moreover, the HTML editors and converters are often limited in their capabilities, or buggy, or produce bad HTML code which does not work on different platforms.

But why HTML 3.2?

The HTML language exists in several variants and continues to evolve, but the HTML 3.2 constructs will most probably be usable in the future, too. By learning HTML 3.2 and by sticking to it as far as possible, you can produce documents which can be browsed by a large variety of Web software now and in the future. Later you may learn to add some useful constructs defined in HTML 4.0 (or future HTML standards as they are defined). This does not exclude the possibility of using other features,

such as enhancements provided by Netscape Navigator or Internet Explorer or some other product, if it really serves your purposes and you are willing to accept the consequences (e.g. limitations on accessibility). But it is wise to adopt the habit of producing documents in a standardized language and using extensions only when really necessary.

HTML 3.2 has been defined by the World Wide Web Consortium, W3C. It is supported by several browsers to a large extent, and it will probably become the common basis understood by almost all relevant Web software.

The next version of HTML, an extension to HTML 3.2, is known as **HTML 4.0** (or the code name Cougar). It was approved as a W3C recommendation in December 18th, 1997, but it takes time before there will be new browser versions which support it and before users widely upgrade to such versions. In particular, Netscape 4.0 and Internet Explorer 4.0 do **not** support HTML 4.0 in general; see especially Stephanos Piperoglou's *HTML 4.0 in Netscape and Explorer* (to which I have some minor annotations). Thus, for quite a long time, netwise long, it will be **safest to use HTML 3.2**, adding useful HTML 4.0 features when needed. If possible, when using HTML 4.0 try to do things so that they "degrade gracefully" on browsers which only support HTML 3.2. On the other hand, the HTML 4.0 specification makes some HTML 3.2 elements and attributes "deprecated", but they are mostly for presentational features and not recommended in this document anyway; and in this document the note "Deprecated in HTML 4.0" is given for them. Moreover, there are changes to the syntax of some elements which impose stricter rules than HTML 3.2. Some of these stricter rules apply to "HTML 4.0 Strict" only; the HTML 4.0 specification defines the Strict (and recommended) syntax as well as the more permissive Transitional syntax, recommending that new documents use Strict. Therefore, and since such strictness requires just a little attention from the author, appropriate notes about the stricter syntax rules in HTML 4.0 are given in the presentation of elements in this document.

An older standard, **HTML 2.0**, is supported to an even larger extent, since HTML 3.2 is an extension of HTML 2.0.

However, to be exact, the following HTML 2.0 features have been *removed* in HTML 3.2:

- NEXTID element
- URN and METHODS attributes in A elements
- the escape notation for double quote, " (notice that you can practically always use just plain " as such, using single quotes around attribute values if they contain the " character)
- the occurrence of an IMG element within a PRE element (it probably wasn't the intention to allow that in HTML 2.0)
- the occurrence of a heading element within an A element (notice that nesting an A element within a heading element is allowed and was the recommended way in HTML 2.0)
- the use of the SAMP element to indicate "a sequence of literal characters" in general; that element is now reserved for presenting sample output only.

It might be a good idea to try to write your documents in HTML 2.0 if possible (avoiding the above-mentioned omitted features, of course). For this reason, constructs (e.g. tags, tag attributes, or attribute values) which are legal HTML 3.2 but not HTML 2.0 are flagged in this document as follows: **(Not in HTML 2.0!)** Notice that even by sticking strictly to HTML 2.0 you cannot absolutely guarantee a proper rendering of your documents, since there are deficiencies in browser implementations.

The scope of this document

This document can be used both for a **systematic study** of the HTML language, specifically HTML 3.2, and as a **reference**.

For recommendations on how to use this document for a systematic study, please refer to section *Learning HTML 3.2 systematically*.

For reference use, there is a systematic description of all HTML 3.2 elements and their all possible attributes, illustrated with examples. You can access them either using the table of contents page or the index and legend of element descriptions. There is also important reference material e.g. about syntax of various attributes, in the section *General remarks on the syntax of HTML*, but the descriptions of elements contain links to relevant information there.

Since this document is an overall description of HTML 3.2, it cannot go into details of good practical usage of elements very much. Instead, there are links to such information elsewhere.

This document does **not** discuss **general issues of Web authoring**, such as overall design of documents and document collections. As regards to them, see my list of suggested reading. However, in appropriate places in this document, there are practical stylistic recommendations, aimed at promoting *structural clarity and universal accessibility using any browser* (within reasonable limits).

In addition to such issues, you need to know where to put your HTML document to make it accessible to the world. This may involve things like transferring files from your own computer to a Web server and setting up directory and file protections suitably. Please consult your local Web support for information relevant at your site.

This document concentrates on **basic HTML usage**. In particular, it does not describe how applets or CGI scripts are written (programmed), although a few potentially useful links are given. This implies no opinion of the usefulness of such things; it's just a consequence of this document being about HTML, not about the World Wide Web in its entirety.

On the versions of this document

This document exists both as a collection of **interlinked** smaller HTML files and as a **single** HTML file. The interlinked version is usually much more convenient to use; you can use its table of contents and access the particular parts you are interested in. But you might wish to access the single-file version e.g. in order to search for a particular term or phrase (using the "search in this document" function of your browser). The master (most up-to-date) copies are accessible at

- http://www.hut.fi/u/jkorpela/HTML3.2/ (the index file of the collection of interlinked files)
- http://www.hut.fi/u/jkorpela/HTML3.2/all.html (the one-file variant).

For **printing on paper**, you may wish to use the PostScript version (about 150 pages; generated from the HTML version with html2ps), which also exists in a much smaller form, as compressed with the Unix `compress` utility. Naturally you can also access the single-file version and use the print function of your browser, but unfortunately such functions are rather primitive in most browsers.

There is also a zipped version of the material, at http://www.hut.fi/u/jkorpela/HTML3.2/html32.zip, which you can **download for offline browsing**. Please unzip the material into a separate directory, since it consists of a large number of files. You can then use any Web browser to access the resource

on your disk, normally starting from the table of contents `index.html`. Everything should work just fine in offline mode, too; naturally, links to resources outside the material will only work when you are online. It depends on the browser how you open a local file. In a typical graphical browser, you use an item in a `File` menu to select a local resource. Having done that, you can bookmark it, for simpler access in the future.

The initial version of this document was written in December 1996 - January 1997. It was developed, with the aid of valuable comments from a large number of people, to reach maturity in December 1997. After this, no major changes are to be expected. For the record, revision history since Dec. 22nd 1997 is available.

Best viewed on...

Of course, this document complies with the HTML 3.2 specification, to the best knowledge of the author. No attempt has been made to "optimize" the document for presentation on some particular browser.

In general, you should be able to read this document on **any** normal WWW browser. However, tables (TABLE elements) have been used in this document, mainly in the description of attributes, since they are essentially tabular information best presented so. Unfortunately this means that parts of this document are almost illegible when viewed with browsers which cannot present such tables well enough (this applies most notably to Lynx).

Copyright notice

Copyright © 1997, 1998, 1999 Jukka Korpela.

The author hereby gives general permission to copy and distribute this document or parts thereof in any medium, provided that all copies contain, in a manner appropriate for the medium, an acknowledgement of authorship and the URL of the original document, i.e. http://www.hut.fi/u/jkorpela/HTML3.2/

The permission granted above does *not* imply permission to distribute this document in a modified form or as a translation. Please contact the author to discuss the conditions for such actions.

Explanation: The author wishes to preserve the integrity of the document. This includes specifying the context when distributing or using excerpts and informing the reader about the availability of the entire document in its most up-to-date form.

How to study HTML 3.2

Getting started with HTML in general

If you do not previously know HTML in any version, you should first read some **introduction to the basic concepts and ideas** behind HTML. You might consider the following options:

- *10 Minute Guide to HTML* by Dave Raggett, available at `http://www.w3.org/MarkUp/Guide/`
- *HTML Tutorial -- Writing Web Pages* by Michael Hamm at **`http://www.crosswinds.net/%7emsh210/html.html`**

- *Introduction to HTML* by Dianne Gorman, at **`http://www.awpa.asn.au/html/index.html`**
- *HTML with Style Tutorials* by Stephanos Piperoglou at **`http://www.webreference.com/html/tutorials/`**
- *Getting Started with HTML*, by me, available at **`http://www.hut.fi/u/jkorpela/html-primer.html`**
- *Introduction to HTML* at **`http://www.cwru.edu/help/introHTML/`**
- *NCSA Beginner's Guide to HTML*, a "classical" introduction. Many people have found it very readable. The original version described HTML 2.0, but it has later been extended. It is available at **`http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html`**
- Yahoo!'s computer area, **`http://www.yahoo.com/Computers/`**, which contains, in its World Wide Web section, a list of guides and tutorials on HTML (in several languages, quality varying a lot).

Please notice that most introductory texts on HTML do not present the language exactly as defined by HTML 3.2; some of them might differ a lot from it.

Learning HTML 3.2 systematically

When you know the very basics of HTML in general, a suggested order of studying HTML 3.2 is the following:

1. Read *The obligatory structure of a document* and *The recommended structure of a document*. You may wish to compare this information with *The structure of an HTML 3.2 document* on the *Wilbur - HTML 3.2* pages at **`http://www.htmlhelp.com/reference/wilbur/`** by the Web Design Group (the basic content should be the same, but you might prefer WDG's style of describing things to mine)
2. Practise by creating an HTML document with the recommended structure but no contents so far; store this document under a name like **`template.html`** and use it as the basis for your HTML documents in the future; create a copy of it, add some plain text into the body and check that the document is readable using a Web browser.
3. Read *Fundamental structures in HTML 3.2, with examples* of this document. Concentrate on studying (and possibly enjoying) the ideas and their application, not on memorizing technical details.
4. Study the general remarks on the syntax of HTML in this document. You will need that information when writing HTML. However, you may at this phase ignore the subsection *Miscellaneous notes*
5. Practise by creating useful HTML documents of your own, using the tags you have learned so far.
6. Browse through the short descriptions part of this document, to get a picture of what is available in HTML 3.2, and following the links to get more information about elements that seem potentially useful to you.
7. Then the world is open to further practising and studying. But beware: there are false prophets and a lot of misuse of HTML around. May the Structure be with you!

The official HTML 3.2 specification

When you have doubts about the exact form, meaning, and limitations of an HTML tag, you should consult the **most official** documents on HTML available: the World Wide Web Consortium documents at
`http://www.w3.org/pub/WWW/MarkUp/Wilbur/`
especially the W3C Recommendation *HTML 3.2 Reference Specification*

There are some minor internal inconsistencies in the HTML 3.2 specification.

The specification is relatively short and technical, and consulting the older HTML 2.0 specification (also known as RFC 1866) can be useful, since the current HTML 3.2 specifications can sometimes be understood only be assuming HTML 2.0 as a background document.

In order to understand the HTML specifications exactly, some fluency in reading **SGML** (the metalanguage used to describe the syntax of HTML formally) is required. SGML as a whole is rather complicated, and the SGML standard is only available in printed form. However, for the purpose of understanding the SGML descriptions of the syntax of HTML (that is, HTML DTDs), the following material usually gives you enough information:

- *A Little Bit of SGML* by Dianne Gorman (nice concise presentation of the basics).
- *HTML Unleashed: SGML and the HTML DTD* by Dmitry Kirsanov (more detailed, yet very readable)
- *Hyvin lyhyt johdatus SGML:ään* by me, in Finnish
- *Gentle Introduction to SGML*, which is rather verbose and originally written for a specific context, but it really explains well the ideas behind SGML
- *Background on Generalized Markup* by Benoît Marchal; a nice introduction which helps you understand the difference between *procedural markup* and *generalized markup* (generic coding, structural markup)
- *The SGML Web Page*, especially the page *SGML: General Introductions and Overviews*
- SGML pages of W3C

Probably the most comprehensive source of information about SGML is the following book (which contains the SGML standard with annotations as well as tutorial material): Goldfarb, Charles F.: *The SGML handbook*. Oxford, 1990. Clarendon Press. ISBN 0-19-853737-9. UDK: 681.3.06.

Additional sources of information

There is a large number of **good documents on HTML authoring in general**. To mention a few of them:

- *Hints for Web Authors* by Warren Steel. More than hints; this is really a good practical summary.
- *Style guide for online hypertext* by Arnoud "Galactus" Engelfriet. Another good summary which covers most basic issues and gives concrete recommendations.
- World Wide Web Consortium pages in general. A lot of information there, although not always in good order.
- *The Web Design Group's Web Authoring FAQ*; a very valuable document containing answers to several practical problems.
- WDG's checklist for HTML authors: *Frequently Encountered Problems: HTML*
- *Web Site Development Information* by Tobias C. Brown; a large collection of links to carefully

selected high-quality documents
- *Publishing on the Web Is Different* by me
- Dan's Web Tips - a collection of enjoyable articles on Web authoring.

The World Wide Web FAQ is mentioned here mainly for historical reasons. In the early years of the Web, it was very important, but the last update is from 1996, so that it has become, as its author puts it, "dusty". However, you might still learn some useful ideas from its section on authoring.

Notice that documents on HTML very often contain information about features which do not belong to HTML 3.2.

Some sources of information on **HTML 3.2 in particular**:

- The *Wilbur - HTML 3.2 pages* by WDG contain a lot of information; they are much more explicit than the official specifications in describing HTML elements.
- *Hyper Text Markup Language v3.2 Reference* (by Sean Bolt).
- *HTML 3.2 and Netscape 4.0* (by Andrew B. King) compares the standard with a popular browser.
- *Quickie Reference for HTML tags*, which is a compact overview of basic tags
- *Bare Bones Guide to HTML* (available in several languages, but the translations can be sloppy and based on old versions).

You may encounter **strange HTML tags** or attributes in other people's documents, especially if you are given the task of maintaining documents written by other people. It's often difficult to find out what they are intended to do and widely they can be expected to work (there is a lot of variation in this!). It is not possible to write a description of "all HTML tags", since the situation keeps changing all the time and many proprietary tags are poorly documented. However, there are some extensive documents which may help you in getting quickly at least a *rough* idea of what a tag *might* stand for:

- *HTML Tag List* by Rob Schlüter; also available as a version which uses frames.
- *HTML Tag Support History* by Brian Wilson.
- *HTML Compendium* by Ron Woodall. This material has serious accessibility problems (e.g. using color as the only means of transmitting essential information and requiring frames), but it is an extensive collection of information. It includes also tags which have been proposed (especially in the HTML 3.0 draft) but not actually implemented except experimentally.
- *HTML Quick List* in Mikodocs by Miko O'Sullivan.

The *HTML Elements List* by Sandia National Laboratories was traditionally referred to as a description of various HTML elements and support for them in some popular browsers. However, that document became out of date, and it has now been removed.

Notice that documents in the list above are not authoritative. They may contain errors. And remember that no list really covers all HTML tags.

Checking your HTML

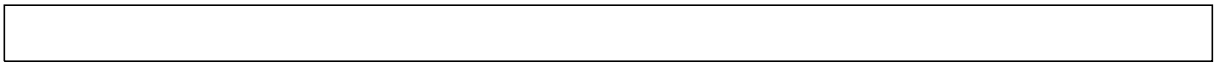
When you have started creating and maintaining important HTML documents, you should learn to use a **validator**, i.e. a program which checks your HTML code *syntactically* against the HTML 3.2 (or some other) specification.

Even if you know HTML 3.2 well, you will occasionally by mistake violate the specification; for instance, just forgetting an ending quote can cause a lot of such violations. Since different browsers have different error handling, you may not notice the error in your environment but your readers may

get confused.

It is **not** sufficient to check that "it works" on your browser. Other people will use that browser in a different environment or with different settings, different versions of the browser, or even quite different browsers. Browsers very often pass invalid HTML without giving error messages, perhaps even handling in such a way that things seem to work fine. For other people, it might be a mess. Looking at your document on a few different browsers may help to detect problems, but it would be too tedious to do that for all important browsing environments.

Therefore, validate your code. There are online services (i.e. Web pages which contain forms which accept a URL of a document and send back the validation report. One of them is the *W3C HTML Validation Service* which you can use for example through the following simple form:



Although all (real) validators do the same basic job, there are some differences e.g. in how understandably they report errors. For such reasons, you might find it useful to check the WDG validator.

Notice that quite often a single error in your document causes a lot of error messages from a validator. The reason is that a structural error may cause a validator to "get out of phase" when analyzing your document. Thus, fix as many validation errors as you can from the beginning of the error report, then submit for validation again.

Passing validation means that there are no violations of HTML syntax (providing that the validator does its job right). It does not imply overall good quality, of course, just *one* component of quality. For some other quality checks, there are some **checkers** or *linters* such as WebLint which can be used to test the document for various common problems - for things which, although technically legal, are likely to provoke known browser bugs, etc. (Unfortunately, Weblint seems to fail to recognize several HTML 3.2 constructs, so it might issue quite confusing and inadequate messages sometimes.) In fact, the KGV validator is able to run Weblint for your document, too, so you can do validation and linting the same time.

Checkers may of course perform an HTML syntax check too, but typically they are rougher than validators. They might declare that a document is OK when it in fact has definite syntax errors, or declare it as erroneous when has correct syntax. Nevertheless, they are useful tools, both for alerting newcomers to potential problems and for picking up errors made by even the most experienced.

Yet another important issue is **checking the links**. One of the worst problems in Web authoring is "linkrot": links easily become invalid. (Well, they might be invalid from the very beginning, too, if you mistyped the URL.) Documents to which you have linked might just be deleted, moved to another location, or be changed so that they become useless as regards your purposes of linking, due to deletion of some information or to not being updated to reflect actual changes in the state of affairs. Automated tools can be of some help here. For instance, NetMechanic allows you to send a URL to be checked for technical validity of links, and it will send you back an E-mail message telling you where you pick up its report.

For more information, see Heikki Kantola’s nice compact list of validators and checkers and WDG’s (annotated) extensive list of validators and checkers.

General remarks on the syntax of HTML

Character set

The *character repertoire* available to the author of HTML documents is not fixed exactly but it should, according to specifications, contain the ISO Latin 1 set, also known as ISO 8859-1, since it belongs to the ISO 8859 set of standards. Notice that the *encoding* of characters may vary, although the default encoding is the one specified in ISO 8859-1, and that encoding is the only one that browsers are required to support. (The HTTP protocol specifies how information about encoding is to passed along with a document.)

In addition to character repertoire and encoding (of characters by bit combinations), there is a special feature which *is* fixed in HTML: the interpretation of numerical character escapes of the form &#*n*; where *n* is a number. Such an escape is to be interpreted as the character corresponding to *n* in ISO 10646 and Unicode. In practice, browsers cannot represent all ISO 10646 characters, but the specifications imply that *if* a browser presents &#*n*; as a character, it must use the ISO 10646 character. (Unfortunately, browsers often violate this.)

In practise, you should **use ISO Latin 1 characters only**. Currently or in the near future you can hardly expect general support for extensions to it, although support to some national alphabets may exist nationally. Support for ISO Latin 1 *should* exist in all browsers, but there are problems even with this. You may of course decide to stick to the ASCII character set, which is a subset of ISO Latin 1, especially if you do not need letters with diacritic marks (or, in general, letters other than English a - z).

The printable characters of ASCII (with code values from 32 to 126 in decimal) are the following:

```
  ! " # $ % & ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ 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 [ \ ] ^ _
` 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 { | } ~
```

The other printable characters of ISO Latin 1 (with code values from 160 to 255 in decimal) are the following:

```
  ¡ ¢ £ ¤ ¥ ¦ § ¨ © ª « ¬ - ® ¯
° ± ² ³ ´ µ ¶ · ¸ ¹ º » ¼ ½ ¾ ¿
À Á Â Ã Ä Å Æ Ç È É Ê Ë Ì Í Î Ï
Ð Ñ Ò Ó Ô Õ Ö × Ø Ù Ú Û Ü Ý Þ ß
à á â ã ä å æ ç è é ê ë ì í î ï
ð ñ ò ó ô õ ö ÷ ø ù ú û ü ý þ ÿ
```

Note: The presentation of some characters in copies of this document may be defective e.g. due to lack of font support. Naturally, the appearance of characters varies from one font to another.

If your *keyboard* or text editor does not allow you to *enter* (i.e. to type directly) some ISO Latin 1 characters such as ä or ñ, you can use the character escape conventions.

Some practical **warnings** to those who create HTML documents on microcomputers:

- The DOS and Macintosh character sets are incompatible with ISO Latin 1 as regards to the use of any characters outside the ASCII character set. In general, some conversion is needed. Some programs can do the necessary conversions automatically, but there can be errors in the conversion tables.
- The Windows character set is mostly compatible with ISO Latin 1, but there are some code positions which are reserved for use as control characters in ISO Latin 1 but used for visible characters in the Windows character set. The most commonly used of them are the two different dashes, "en dash" and "em dash", which should not be mixed up with the hyphen (-) or the underscore (_), which belong to ISO Latin 1 (and even to ASCII). If you use such characters, users on Windows systems will probably see them as intended, but on all other system the document most probably looks more or less messy. (Usually such characters are not displayed at all.) Thus, it is best **not** to use them; see my document *On the use of some MS Windows characters in HTML* for detailed reasons and suggested replacements.

See also

- *A tutorial on character code issues* by me.
- *Character Encoding Standards* in *HTML Unleashed*
- A. J. Flavell's *Notes on ISO 8859-1 in the Web context*

HTML tags

An HTML **tag** consists of the following, in this order:

- the left angle bracket < (same as the "less than" symbol)
- optionally, the slash /, which means that the tag is an *end tag* which closes some structure; thus, in this context you can read the / character as *end of ...*
- the tag name, e.g. **`TITLE`** or **`PRE`**
- optionally, if the tag can have attributes, a blank followed by one or more attribute specifications like **`ALIGN=CENTER`**
- the right angle bracket > (same as the "greater than" symbol).

Examples:

```
<H1>
<H1 ALIGN=LEFT>
```

HTML elements

Most, but not all, HTML tags are paired so that that an opening tag is followed by the corresponding closing tag, and there can be text or tags between them, as in

```
<H1>Foreword</H1>
```

In such cases the two tags and the part of the document enclosed by them forms a unit which is called *HTML element*.

The opening tag, or *start tag*, is a tag without the / character, since the presence of that character after the opening < indicates a tag as a closing tag, or *end tag*.

Some tags, e.g. <HR>, are HTML elements by themselves, and for them the corresponding end tag would be illegal. - In the sequel we will usually refer to tags by their name only, omitting the obligatory angle brackets.

For some elements which logically consist of a start tag, some content and an end tag, it is legal to omit the end tag, possibly even the start tag. For example, you can omit the end tag </P> and let browsers and other software imply it when necessary. The exact rules for allowable tag omission are given in the HTML specification, often only in the formal (SGML) syntax, so they can be hard to read. Moreover, some browsers are known to misbehave if you omit some end tags even when the specs allow it, and this can have drastic effects e.g. when nested tables are involved. Thus it is wisest to *use explicit end tags always* for all elements which logically have an end tag.

Attributes

For each element, a set of possible **attributes** is defined. This set can be empty or rather large, but most elements accept one or a few attributes. In almost all cases the attributes are optional.

An **attribute specification** consists of the following, in this order:

- the attribute name, e.g. **`WIDTH`**
- the equals sign **`=`**
- the attribute value, which is a string, e.g. **`"80"`**.

Attributes, if present, are written into the start tag of an element, after the element name. They must be separated from the element name and from each other by blanks or newlines. The *order* of attribute specifications in a tag is insignificant.

It is always safe to enclose the attribute value in **quotes**, using either single quotes (**`'80'`**) or double quotes (**`"80"`**), using matching quotes of course. The string in quotes must not contain the quote, so if the data contains a double quote, use single quotes for quoting, and vice versa. Some browsers do not treat quotes within quotes correctly, so if you need to include a quote as data character into an attribute value, it is probably safest to present them using character escapes: double quote " as `"` and single quote ’ as `'`.

In general, using double quotes is preferable, since for the human eye single quotes are sometimes difficult to distinguish from other characters like accents. Moreover, some browsers do not handle single quotes correctly.

You can also omit the quotes from an attribute value if the value consists of the following characters only (cf to the technical concept of name):

- letters of the English alphabet (**`A`** to **`Z`**, **`a`** to **`z`**)
- digits (**`0`** to **`9`**)
- periods **`.`**
- hyphens **`-`**

Thus, **`WIDTH=80`** and **`ALIGN=CENTER`** are legal shorthands for **`WIDTH="80"`** and **`ALIGN="CENTER"`**. A reference to a URL like **`HREF=foo.html`** is acceptable, but **in general URLs must be quoted when used in attributes**, e.g. **`HREF="http://www.hut.fi/"`**. - Some browsers are more permissive. Some browsers may even accept elements with a starting quote but without any closing quote. Such use is very bad practise.

Often an attribute value is keyword-like: it must be one of set of particular strings (case-sensitively and allowing quotes around). For example, if a P element contains an ALIGN attribute, its value must be one of LEFT, CENTER, RIGHT. There are also attributes for which **a string in general** can be used as value; technically speaking, such attributes are declared as having CDATA value in the formal SGML description (DTD) for HTML. Even within such attribute values, **no HTML tags** are recognized. On the other hand, **escape sequences are recognized** and interpreted within attribute values. (At least they should. Some browsers don't get this right.) This in turn means that if any of the characters &<> is to appear within an attribute value, it is safest to present it using the escape notation & or < or > respectively.

There is a **minimized syntax** for attributes when the attribute value is the same as the attribute name. For instance, <UL COMPACT="COMPACT"> can be abbreviated as <UL COMPACT> (and it is common practise to do so). Some browsers even require minization for some attributes (COMPACT, ISMAP, CHECKED, NOWRAP, NOSHADE, NOHREF), so perhaps it is best to use the minimized syntax when applicable.

URLs

Several HTML elements, most notably the A element, may contain an attribute which takes a URL as value. URLs, Uniform Resource Locators, are addresses of Web documents. More generally, URLs can be used on the Web to refer to "objects" on the Web or in other information systems.

Absolute URLs

The general syntax of absolute URLs is the following:

scheme`:`//*host*`:`*port*/*path*/*filename*

where

scheme
: specifies the information system (technically speaking, the *protocol*) to be used to access the resource; possible values include the following:

`http`	a Web document (to be accessed using Hypertext Transfer Protocol, HTTP)
`ftp`	a resource to be retrieved using FTP (File Transfer Protocol), usually a file in a so-called FTP server,
`file`	a file on a particular computer; a `file` URL is hardly useful on the Web
`gopher`	a file in a Gopher server
`mailto`	electronic mail address
`news`	a newsgroup or an article in Usenet news
`telnet`	for starting an interactive session via the Telnet protocol (which is part of TCP/IP)

host
: is the Internet host name in the domain notation, e.g. `www.hut.fi` (or sometimes a numerical TCP/IP address); notice that typically, but not necessarily, Web servers have domain names starting with `www`

`:`*port*
: is the port number part, which can usually be omitted since it has a reasonable default; that is, omit it, unless it is a part of a URL which you got somewhere (or you really know what you are doing)

path
: is a directory path within the host

filename
: is a file name within the directory.

Actually, this pattern is mainly for Web documents, i.e. `http` URLs. For other URLs, simplifications and special interpretations are applied. For example, a `mailto` URL is just of the form `mailto:`*address* where *address* is a normal Internet E-mail address like `Jukka.Korpela@hut.fi` (as specified in RFC 822). Please notice that appending anything to the E-mail address in a `mailto` URL is unsafe and may result in lost mail without anyone noticing! (See also the discussion of `mailto:` URLs in the description of the A element.)

Notes and warnings

URLs are generally **case sensitive**. Some parts of URLs (such as server name) can be case insensitive. But for example, in a URL, `foo.htm` is quite different from `FOO.HTM`. Although a server *might* accept both, and treat them as referring to the same resource, this is server-specific.

The separator character between hierarchic parts of URLs is the **slash**, or solidus, character `/`, not backslash (`\`). This does *not* depend on the operating system of the server or the browser. (If the file system on the server uses backslash in hierarchic file names, then the server software is responsible for handling this when mapping URLs to file names, invisibly to users and authors.)

It is safest (and in many cases obligatory) to enclose URLs in **quotes** when writing them as attribute values in HTML.

Although many browsers allow you, as a "Web surfer", to omit the part `http://` when specifying the URL of a document to be visited, you, as an author, must not omit it in when writing a normal URL into an HTML document. Otherwise browsers will try to interpret it as a relative URL. See below.

Relative URLs

Basically anything that appears where a URL is expected and does not begin with a scheme part such as `http://` or `ftp://` should be interpreted as a relative URL. (However, when a URL is given directly to a browser (in a Location or URL field or something like that), browsers tend to imply `http://` rather than interpret the data as a relative URL. This is to some extent understandable, but it tends to confuse people who write HTML documents.)

A relative URL is an abbreviated form of `http` URLs, and it is interpreted as relative to the *base URL* of the document. The base URL is by default the URL of the document itself, but it can be changed using the BASE element.

Given a base URL, say `http://www.server.example/xyz/bar/zap.html`, and a relative URL, say `foo.html`, the browser acts as follows: it takes the base URL, deletes its trailing characters up to (but not including) the last slash (`/`), then appends the relative URL; the result is the absolute URL `http://www.server.example/xyz/bar/foo.html` which is then used normally by the browser.

If a relative URL begins with the slash `/`, it is interpreted as relative to the server root. In our example, `/foo.html` would mean `http://www.server.example/foo.html`

If a relative URL contains the special notation `..`, it means that one hierarchic part (a part between two slashes) is removed from the base URL when constructing the absolute URL. In our example, `../foo.html` would thus mean `http://www.server.example/xyz/foo.html` (note that the part `/bar` was "wiped out"). This principle was included into URL syntax to imitate operations like `cd ..` ("go one directory upwards in a tree") in some systems, but it is logically independent of them; it's technically just a formal operation on URLs as strings, and it need not correspond to anything in a file system (though it usually does).

Fragment identifiers

An `http` URL (absolute or relative) can have a fragment identifier appended to it, to construct an address that refers to a particular **location** or **part** in a particular document. The fragment identifier is separated from the URL proper by a number sign character `#` See the description of the A element for more information.

Normally you set a destination anchor in HTML using an A element with an attribute like NAME="xyz", and you refer to it using a fragment identifier like #xyz. But it might be possible to set destination anchors in other than HTML documents too, e.g. in a PDF document. See *Two-Way Linking of HTML and Acrobat Files* by Don Lancaster.

Technically, by URL specifications, a fragment identifier is not part of the URL proper. Instead, a construct of the form *URL#fragment-identifier* is called a *URL reference*. But HTML specifications generally don't make this distinction, so normally whenever a URL is permitted in HTML constructs, a fragment identifier can be appended too. The distinction still matters; in particular, the URL encoding rules discussed below do *not* apply to fragment identifiers.

More information about URLs

The description of URLs in *Dan's Web Tips* is a very readable discussion of several fundamental principles as well as practical issues.

As regards to the technical specifications of the syntax of URLs, see RFC 1738 (absolute URLs) and RFC 1808 (relative URLs) as well as RFC 2396 which supersedes them as far as the generic URL syntax is considered. Note that in addition to the URL schemes defined in RFC 1738, various new schemes and modifications to the old schemes have been defined and proposed. See especially W3C material on addressing, which contains (an attempt at) an exhaustive list of URI schemes.

URL encodings, or what to do e.g. with spaces

Within a URL *only a limited set of characters can be used as such*:

- alphanumeric characters (`A` to `Z`, `a` to `z`, `0` to `9`)
- the characters `-_.!*'()`
- the characters `;/?:@=&#+$,` provided that they are used in the *special meaning* reserved for them in the RFCs mentioned above.

Other characters must be encoded. (The characters `;/?:@=&#+$,` must also be encoded, if they are not used in the special meaning.) This encoding (which is defined by URL specifications, not HTML specifications) consists of using the percent sign followed by two hexadecimal digits, presenting the code position. See e.g. my list of ISO Latin 1 characters to find the hexadecimal codes for characters. (In principle, only Ascii characters, code positions 20 through 7E in hexadecimal, should be used; other characters may or may not work.) For example, tilde (~) should be presented as `%7E` and space as `%20`. (Violating the rules causes problems much more likely in the latter case than in the former.)

When a URL occurs as an attribute value in HTML, there is another complication caused by the **&** character which may have special use in query form submissions. That character should be escaped as & or as & (there is a footnote in the HTML 2.0 specification about this) and browsers should process it so that the actual URL passed to the processing CGI script has that notation replaced by plain & character. (Notice that it must *not* be encoded using the `%` notation. This is a confusing issue, and CGI scripts should really be written so that semicolon ; and not ampersand & is used as field separator.)

Case sensitivity

As regards to tag and attribute names and most keyword-like attribute values, HTML is **case insensitive**. You can, for example, type **`TITLE`** or **`Title`** or **`title`** or even **`tItLE`** if you like. As an exception, the value of a TYPE attribute in an OL element is case sensitive.

In this document, upper case letters are used for the above-mentioned constructs. This may help the reader distinguish HTML code from normal text. The modern trend is to use lower case, and this is a requirement in XHTML.

Beware that the following constructs are (in general) **case sensitive**:

- **escape sequences** (more officially called *character entities*), which begin with **`&`** (e.g. **`<`**)
- **URL**s although they may contain parts that are case insensitive, it is safest to use consistent spelling as regards to case of letters
- other attribute values which are not keyword-like but strings in general, such as the value of an ALT attribute in an IMG element and the value of a NAME attribute of an A element.

Division into lines and the use of blanks and tabs

With the exception of text enclosed in PRE tags (preformatted text) or TEXTAREA tags, **blanks and newlines are not preserved when displaying the document**. More technically, any sequence of blanks, tabs, and newlines is equivalent to a single blank in HTML file. On the other hand, a blank in the HTML file may be rendered using any number of empty space or replaced by newline(s).

The term *newline* is used to denote an end of line designation. Theoretically, the SGML declaration for HTML specifies that line feed (LF, ASCII code 10 in decimal) acts as a record (line) start character and carriage return (CR, ASCII code 13 in decimal) as a record end character. In practise, HTML documents are presented and transmitted using a newline presentation convention of the computer system used. Therefore, HTML browsers are encouraged to accept any of the three common representations, namely CR LF sequence, CR only, and LF only, as line separators and to infer the missing record end and start characters.

Thus, it does not matter how you divide the text into a lines, since a newline is equivalent to a blank. Notice, however, that you *must not divide a word into two lines in HTML*. If you e.g. divide the word `international` into two lines as follows:

```
inter-
national
```

it will be interpreted as equivalent to

```
inter- national
```

and the result is not what you want.

Thus, you must use HTML tags such as P or BR to force line breaks, if they are necessary for the logical representation of your document.

Browsers usually do not divide words into two lines, except possibly when a word contains a hyphen. The *HTML 3.2 Reference Specification* is not very explicit in this matter; it just says, in the discussion of tables, the following:

> For some browsers it may be necessary or desirable to break text lines within words. In such cases a visual indication that this has occurred is advised.

Beware that the **line length is outside your control**. It depends on the browser, device, and settings used by the people who look at your document. You can *force* line breaks but *not prevent* line breaks between words, in general. (You can *try* to prevent line breaks by using non-breaking spaces.)

As regards to **newlines in conjunction with HTML tags**, there are special rules:

- A newline immediately following a start tag is ignored. For example,

  ```
  <P>
  Text
  ```

  is equivalent to

  ```
  <P>Text
  ```

- Similarly, a newline immediately preceding an end tag is ignored. For example,

  ```
  Text
  </P>
  ```

  is equivalent to

  ```
  Text</P>
  ```

However, popular browsers (such as Netscape and Internet Explorer) are known to violate these official rules. For example, if you write an A element as follows:
`<A HREF="foo.html">bar </A>`
then many browsers incorrectly display it as if the link text had a blank appended. Since browsers often indicate links with underlining, there could be an extra underlined space. Thus, *in some cases removing a newline before an end tag may help* in improving the presentation on popular but buggy browsers. See the document *White Space Bugs in Browsers* for more detailed explanation with examples.

The *horizontal tab* character (HT) can appear in the HTML source. Within PRE elements, tabs have a special interpretation. Otherwise a tab is equivalent to a space. Thus, it does not imply tabulation of any kind. (In order to present tabular data, use the TABLE element.) It is best to avoid tabs in HTML code and to use a suitable number of spaces instead, if one wants to format the HTML source code into tabular form.

Classification of elements

The ways in which HTML tags can be combined are defined in terms of elements and their classification. It is much more convenient to define e.g. that an H1 element may contain (only) text elements than to give a long list of allowable elements, especially since the same list would appear in many contexts and it may change when new text elements are added to HTML in its future revisions.

Apart from the elements at the topmost levels, namely HTML, HEAD and BODY, the HTML elements are classified into three major categories:

- **head elements**, i.e. elements used in the HEAD element, to specify information *about the document as a whole*: TITLE, ISINDEX, BASE, META, LINK, SCRIPT, STYLE
- elements which specify the *structure of the document*, e.g. division into parts and paragraphs: H1, H2, H3, H4, H5, H6, ADDRESS and the following **block elements**: P, UL, OL, DL, PRE, DIV, CENTER, BLOCKQUOTE, FORM, ISINDEX, HR, TABLE; sometimes the term *block level element* is used to refer to block elements and heading elements (H1 - H6) and ADDRESS element, but that is confusing
- **text elements**, specifying *text segments and their properties*:
  - plain text, possibly containing escape sequences (such as &)
  - phrase markup: EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE
  - font markup: TT, I, B, U, STRIKE, BIG, SMALL, SUB, SUP
  - special elements: A, IMG, APPLET, FONT, BASEFONT, BR, SCRIPT, MAP
  - form field elements: INPUT, SELECT, TEXTAREA

Any text element (including plain text) can appear wherever a block element is allowed.

A rule of thumb which may help in remembering which elements are block elements and which are text elements: block elements cause paragraph breaks, text elements do not.

Note: Often block elements can contain both text elements and other block elements, i.e. blocks can be nested. Text elements can be nested, too. On the other hand, *text elements may not contain block elements*. For example,
`<CITE><H3>Origin of Species</H3></CITE>`
is invalid (since CITE is text element and H3 is block element) and also illogical (you don't really mean that the heading *as a structure* is a citation, do you?) whereas
`<H3><CITE>Origin of Species</CITE></H3>`
would be legal, although different browsers might treat it differently (letting either H3 or CITE determine the rendering, or possibly using a mixture of the two). Similarly, don't embed headings into A NAME tags but vice versa. It is also illegal to have a paragraph break (P tag) within e.g. a STRONG element; although several browsers can handle it, the semantics is ambiguous and you should use separate start and end STRONG tags within each paragraph (if you really want to emphasize such large portions of text!).

Allowed nesting of elements

This section describes how elements may be nested in HTML 3.2. It does not describe the rules for the ordering or repeatability of elements. It simply answers questions of the form *may element X appear within element Y?*

The same information is presented in the individual tag descriptions, in their *Allowed context* and *Contents* parts. Here it is presented in a compact form. This form does not cover all details but might be more illustrative.

Legend:

- An uppercase word stands for the corresponding element.
- A lowercase word is a term which describes a collection of HTML elements
- Each entry is followed by an indented list of elements which may appear within the elements specified by the entry. If there is no such list, no nested elements are allowed. However, for **block** and **text** the allowed contents is as described under that title
- #PCDATA means "parsed character data" (without HTML tags, but escape sequences such as ä are allowed)
- body.content means the elements which are listed under BODY

HTML

- HEAD
  - TITLE, SCRIPT, STYLE
    - #PCDATA
  - ISINDEX, BASE, META, LINK
- BODY
  - H1, H2, H3, H4, H5, H6
    - text
  - **block**
    - P
      - text
    - UL, OL, DIR, MENU
      - LI
        - text
        - block

        (within DIR or MENU, LI may not contain a block)
    - DL
      - DT
        - text
      - DD
        - text
        - block
    - PRE
      - text *without IMG, BIG, SMALL, SUB, SUP, FONT*
    - DIV, CENTER, BLOCKQUOTE
      - body.content
    - FORM
      - body.content without FORM

        - ISINDEX
        - HR
        - TABLE
            - CAPTION
                - text
            - TR
                - TH, TD
                    - body.content
    - ADDRESS
        - text
        - P
            - text
    - **text**
        - #PCDATA
        - TT, I, B, U, STRIKE, BIG, SMALL, SUB, SUP
            - text
        - EM, STRONG, DFN, CODE, SAMP, KBD, VAR, CITE
            - text
        - A
            - text
        - IMG
        - APPLET
            - text
            - PARAM
        - FONT
            - text
        - BASEFONT, BR
        - SCRIPT
            - #PCDATA
        - MAP
            - AREA
        - INPUT
        - SELECT
            - OPTION
                - #PCDATA
        - TEXTAREA
            - #PCDATA

In order to simplify element descriptions, I will use the term *text container* to denote any element which *may* contain a text element directly (as opposite to containing an element which contains a text element). The following elements are text containers:

A, ADDRESS, APPLET, B, BIG, BLOCKQUOTE, BODY, CAPTION, CENTER, CITE, CODE, DD, DFN, DIV, DT, EM, FONT, FORM, H1, H2, H3, H4, H5, H6, HTML, I, KBD, LI, P, PRE (with restrictions), SAMP, SMALL, STRIKE, STRONG, SUB, SUP, TD, TH, TT, U, VAR.

The following are not text containers but may contain text elements indirectly, i.e. contain elements which are text containers:

DIR, DL, MENU, OL, TABLE, TR, UL.

The following may not contain text elements at all:

AREA, BASE, BASEFONT, BR, HEAD, HR, IMG, INPUT, ISINDEX, LINK, MAP, META, OPTION, PARAM, SCRIPT, SELECT, STYLE, TEXTAREA, TITLE,

Similarly I will use the term *block container* to denote any element which *may* contain a block element directly (as opposite to containing an element which contains a block element). Block containers are: BLOCKQUOTE, BODY, CENTER, DD, DIV, FORM, HTML, LI (when within UL or OL), TD, TH.

Miscellaneous notes: about escape sequences (character entities), names, colors, widths, pixels, non-breaking spaces ( ), comments

This subsection discusses some technical issues which are related to some HTML tags. Rather than presenting them in the descriptions of individual tags, they have been collected here. Please *feel free to skip them in first reading*, and consult them later when needed; the tag descriptions contain links to the relevant information here.

Escape sequences (character entities)

Escape sequences, more formally known as character entities, are a method of presenting special characters. For example, the escape sequence < denotes the less than character (<).

Obviously, since some characters such as < are used with a very special meaning in HTML, there must be some way of expressing them as data characters, i.e. when they should appear e.g. as part of the document itself or in a URL. The convention is that the following notations are used:

character	notation	usual name(s) of the character
<	<	less than character, left angle bracket
>	>	greater than character, right angle bracket
&	&	ampersand

Technically speaking, it is not always necessary to use the escape notations for characters listed above It is, however, easier and safer to follow the simple rules which work always.

There was notation " for the double quote (") in HTML 2.0, but it does not belong to HTML 3.2 (for certain technical reasons). The double quote can be typed as such within normal text, and (in principle at least) within quoted strings as well if the single quotes are used as the outermost quotes.

Notice that the *semicolon* is part of the escape sequence. In principle, it is necessary only if the following character would otherwise be recognized as part of the name. In practice, it is best to adopt the habit of **always terminating an escape sequence with a semicolon**.

In escape sequences, the case of letters is significant. For example, the ampersand & may not be represented as & (this escape sequence is undefined), and the escape sequences ä and Ä denote two distinct characters, *a* umlaut (*a* dieresis, the letter *a* with two dots above it) in lower case and in upper case (*ä* and *Ä*); notice the principle of uppercasing only the first letter in the escape notation (&AUML; is undefined).

The need for the above-mentioned escape sequences arises from the syntax of HTML. In fact there are escape sequences for *all* characters in the ISO Latin 1 character set. There are

- notations like

  © copyright sign, ©

  ® registered trademark sign, ®

    non-breaking space

- notations such as Æ (for AE ligature, Æ) for various non-ASCII letters
- notations of the form &#*n*; where *n* is the code position of a character, in decimal (in the range from 0 to 255); these shall be interpreted as referring to the *ISO Latin 1 character* with code value n (but notice that some browsers are not conformant in this respect)

For a full list, see the appendix *Character Entities for ISO Latin-1* of the *HTML 3.2 Reference Specification*. There is also perhaps slightly more readable presentation of that information: *Table of Character Entities for ISO Latin-1*.

However, there is usually little reason to use other escape sequences than < and > and &. Using ä instead of ä might seem to give some character code independency, but it does not; if a browser can display ä correctly, it can also display correctly a document in which the character ä is specified directly. But notice that sometimes you cannot *input* some special characters directly due to keyboard restrictions, and in such cases you can have use for notations like ä.

And please notice that "character ä" means the ISO Latin 1 character with name "small letter a with diaeresis" (diaeresis = umlaut), with code 344 in octal, 228 in decimal. It can be entered into an HTML document in various ways. It is possible that pressing a key labeled with ä or Ä is *not* among those ways. For instance, on a Macintosh with Scandinavian keyboard the ä key normally produces a character quite different from ä in ISO Latin 1. Various programs may or may not handle this by performing character code conversions.

Some browsers support other escape sequences than those mentioned above, for example ™ and &cbsp;. The use of such notations is *strongly discouraged*. (Notation ™ refers to a symbol which does not belong to ISO Latin 1 at all; you may wish to use the HTML 3.2 conformant notation ^(TM) instead. Notation &cbsp; stands for "conditional breaking space", not in ISO Latin 1 and possibly not intended to be a character at all.)

Names

In some contexts in the definition of HTML, the word *name* appears as a technical term. (Perhaps a more appropriate term would be *identifier*, since the concept bears resemblance to identifiers in programming languages). A name is a sequence of characters containing only

- letters of the English alphabet (**A** to **Z**, **a** to **z**)
- digits (**0** to **9**)
- periods **.**
- hyphens **–**

and beginning with a letter.

This name concept occurs in the description of HTTP-EQUIV and NAME attributes of the META element and in the description of NAME attribute of the PARAM element.

In other contexts, a string which is used to name something may contain other characters as well but then it must be quoted.

Colors

Some HTML constructs can be used to specify colors: by using an explicit BODY element one can specify the background color, default text color, and colors of link texts; and the FONT element can be used to set text color locally.

It is of course possible that due to software or hardware limitations all colors cannot be presented. On some devices, the actual rendering might be just black and white or different shades of grey.

When a color is specified as the value of an attribute, there are two possibilities:

- A symbolic notation such as RED. There are sixteen such names defined (see below). It can be written in upper or lower case, with or without quotes.
- A numerical designation in hexadecimal notation, such as "#FF0000", which controls how the color is formed from some basic colors - more specifically, from red, green and blue in the so-called sRGB color space. Notice that the designation must be within quotes.

Of course, the symbolic notations are much easier to use and more self-explanatory. On the other hand, many authors prefer numerical designations for one or more of the following reasons:

- the set of predefined color names is much smaller than the set of colors definable numerically
- the predefined color names refer to color which are too strong (bright) especially when used as background or otherwise in large amounts
- there are browsers which do not understand color names and which might even interpret them in strange ways.

The following table lists the predefined color names and their numerical equivalents.

Color names and sRGB values

Black = "#000000"	Green = "#008000"
Silver = "#C0C0C0"	Lime = "#00FF00"
Gray = "#808080"	Olive = "#808000"
White = "#FFFFFF"	Yellow = "#FFFF00"
Maroon = "#800000"	Navy = "#000080"
Red = "#FF0000"	Blue = "#0000FF"
Purple = "#800080"	Teal = "#008080"
Fuchsia = "#FF00FF"	Aqua = "#00FFFF"

These colors were originally picked as being the standard 16 colors supported with the Windows VGA palette. The HTML 3.2 Reference Specification contains a section on colors with sample images in each of the 16 colors.

See also

- *Setting Background and Text Colors*
- *The ~~256~~, oops, 216 colors of Netscape*
- section *Color Palettes and Usage* in *Design Elements* by WDG.

Widths

The value of the WIDTH attribute in e.g. an HR or TABLE tag can specified in two alternative ways:

- as a *percentage* of the space between the current left and right margins; in this case the attribute value must be within quotes and the percentage number must be immediately followed by the percent sign, e.g. WIDTH="80%"
- in *pixels*, in which case a plain integer number is used (and no quotes are necessary), e.g. WIDTH=212.

The former, relative specification is more recommendable in general, since the author of a document cannot know the pixel size of the reader's screen.

Pixels

Pixel can be defined as "the smallest element on a screen that can be controlled by a computer in terms of light intensity and colour" (from the entry for "pixel" in a glossary by MDA). The number of pixels in the horizontal and vertical direction constitute the *resolution* of a screen.

Pixel values used in several contexts like width specifications refer to *screen* pixels. The physical size of a pixel depends on the user's screen.

People often ask "for what resolution should I write". See *WDG Web Authoring FAQ*, question *For what screen size should I write?* for a short answer.

A browser should multiply the pixel values by an appropriate factor when rendering to very high resolution devices such as laser *printers*. For instance if a browser has a display with 75 pixels per inch and is rendering to a laser printer with 600 dots per inch, then it should multiply the pixel values given in HTML attributes by a factor of 8.

Non-breaking spaces ( )

The notation   is the escape notation for the the no-break space - a character which is often called non-breaking space, or NBSP for short. According to ISO 8859, this character should be presented as a normal space (blank) but so that it is not replaced by a newline (as normal spaces often are in text processing). This means that a   between two words causes them to be presented at the same line with some inter-word space between them. (The actual width of inter-word space may vary and need not relate to the number of spaces in an HTML file.) Typical examples of use would be "5 m" (meaning "five meters") and "J. Korpela" (where "J." is the given name initial).

The HTML 2.0 specification says:

> Use of the non-breaking space and soft hyphen indicator characters is discouraged because support for them is not widely deployed.

This is somewhat misleading. The soft hyphen should really be avoided; it serves no useful purpose in HTML. But as regards to non-breaking space, it seems to be honored rather well *in its basic meaning* described above. And although the HTML 3.2 Reference Specification is not explicit about the matter in general, it suggests, in the discussion of the NOWRAP attribute of TH and TD elements, that   should act as non-breaking space within table cells at least.

If you use non-breaking spaces, use them *instead of* normal spaces, not in addition to them. For instance, if you wish to prevent a line break between `version` and `3`, type `version 3` (not `version  3`).

On the other hand, within a table in HTML 3.2,   can have quite different meaning, which can be described as *non-empty space*: on several browsers, when a table is presented with borders, cells with empty contents are drawn without them, and spaces only do not constitute contents - but   does! So there is a difference between <TD></TD> and <TD> </TD>. (Netscape also ignores background color suggestions for a table cell unless there is some content, at least , in the cell.) Notice that there can be better ways to deal with empty cells than to use no-break spaces.

For further confusion, some people use   to **force spaces** into the visible presentation of a document, e.g. by putting an   or a few of them into the beginning of a paragraph to get its *first line indented*. This actually works on most browsers, but it is unwise to rely on that, and it is normally useless to try to enforce such presentation features anyway. Indentation can be rather successfully suggested using stylesheets. (And consider what happens when a user has carefully designed a *user stylesheet* which makes paragraphs presented that way. If you use the   hack, that user - who assumably *really cares* about the presentation of paragraphs - will see first lines of paragraphs on your pages *doubly* indented!) The trick of using   between words inside a paragraph to create *wider spacing* is probably less risky. Other tricks which utilize the common but non-guaranteed treatment of   by browsers include using it to create a "flexible pseudo-table" and to try to make options in a SELECT menu be of equal width.

See also notes on the no-break space in *ISO-8859 briefing and resources* by Alan Flavell.

Comments

An HTML file can contain comments, which give explanations to human readers of the HTML code. Comments do not affect the rendering of a document in any way, i.e. they are ignored by a browser.

You can begin a comment with the four-character sequence **<!--** (less than sign, exclamation sign, two hyphens) and terminate it with the three-character sequence **-->** (two hyphens, greater than sign). Don't use the character pair -- or the character > within a comment. For example:

```
<!-- Written by Jukka Korpela -->
```

The reason for the above rule for not using > within a comment is not the syntax of HTML but known deficiencies of popular browser. A practical consequence is that *you should not try to "comment out" parts of your document*; any HTML markup in such parts would confuse many browsers.

For a more thorough discussion of comment syntax, see document *HTML comments* by WDG.

It is generally preferable to include metainformation about the document into HTML elements, such as META. Consider making information about purpose, author, creation and last update time etc a visible part of the document itself, too.

Thus, comments should be inserted in rare cases only, e.g. to comment the HTML code itself to explain things that may look odd. Remember that a comment is part of an HTML file, to be transmitted whenever the document is delivered. Therefore, to avoid wasting bandwidth, if you have a long story to tell, put it into a separate document and insert just its URL into a comment.

HTML editors and converters often insert a few comment lines into the beginning of an HTML file. Such indications can be helpful and should not be removed.

Media types

An Internet media type is, generally speaking, a property of a data set, describing both the general type of data (such as "text" or "image" or "application"; the last one refers to program-specific internal data formats) and, as a subtype, a specific format for the data. The concept was originally defined as "MIME content types".

Media types relate to HTML as follows:

- When a Web server sends an HTML document, it should specify the correct media type (`text/html`) in the HTTP header `Content-Type` it sends along with the document. Normally servers are configured to do this by default when the file name ends with `.html` or `.htm` (depending on the system; please consult local documentation).
- In a FORM element, the value of the ENCTYPE attribute specifies the media type to be used when encoding and sending the content of the form.
- When referring to various resources, such as embedding images using IMG elements or linking to binary files using an A element, there is no way to tell the media type in HTML. (In HTML 4.0, a TYPE attribute is allowed for that element, but its meaning has been defined vaguely, and it has not been implemented in browsers.) Things must be handled in the server. Typically, a Web server uses some mapping table to map file name extensions to media types (e.g. mapping extension `.zip` to media type `application/zip`), and it may provide users some tools for

overriding such mappings or otherwise specifying the media type to be associated with a file or set of files. The description of the A element contains some additional notes related to audio and video and binary files in general.

The *HTML 3.2 Reference Specification* refers to RFC 1521 but that specification was superseded by RFC 2046 (in November 1996). The procedure for registering types in given in RFC 2048; according to it, the registry is kept at
`ftp://ftp.isi.edu/in-notes/iana/assignments/media-types/`

For less authoritative but more readably presented information, see document *MIME Types* by Chris Herborth.

In addition to standardized media types, there are media types which are in fact supported by popular servers and browsers. Appendix B of *Special Edition Using CGI* (by QUE) lists many of them. For an online list, see *Multipart Internet Mail Extensions (MIME)* in *The HTML Sourcebook, 3Ed*, by Ian S. Graham.

You can *check* what is the media type information sent by a server by using Delorie's HTTP Header Viewer.

There is an additional complication caused by the fact that Internet Explorer does not work according to the protocols in this area. It often ignores the media type announced in the `Content-Type` and uses the last few characters of the URL instead to determine the method to be used. (IE may also apply some "heuristics" based on the actual *content* of the data!) This means that *in addition to* making sure that the server sends the correct media type information one should try to name the file so that things might work on IE, too. Thus, one should try to stick to commonly used conventional file name suffixes like `.DOC` for MS Word documents, `.XLS` for MS Excel documents, `.TXT` for plain text documents, etc.

Fundamental structures in HTML 3.2, with examples

The obligatory structure of a document

First of all, let us start with an extremely primitive HTML document: one that only contains the words *Hello world* as plain text. In an HTML file, the contents must be preceded by a head section which minimally consists of two constructs. Our HTML code would be as follows:

Example hello.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<TITLE>Hello</TITLE>
Hello world
```

In fact, this document *implicitly* has the following structure, i.e. it is equivalent to the following:

Example hello2.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>Hello</TITLE>
</HEAD>
<BODY>
Hello world
</BODY>
</HTML>
```

This means that apart from the first line, the entire file is an HTML element which contains a HEAD element, with the TITLE element as contents, and a BODY element, with the plain text as contents.

Thus, in the absence of HTML, HEAD, and TITLE tags a browser implicitly assumes them in suitable places. Therefore, your document always contains a head and a body.

(In fact, the example above does not conform to the requirements of HTML 4.0 Strict: the text `Hello world` should be enclosed into a P element, for example. See notes in the description of the BODY element.)

The recommended structure of a document

In addition to the *obligatory* structure, there are various structural features which are highly *recommendable*. There are various local recommendations at different sites, and you should study the applicable documents carefully.

Here we will simply emphasize that every HTML document should contain certain basic **information about its origin**. The local recommendations may specify in detail the **form** in which that information should be provided.

The importance of providing origin information becomes evident if we think how people find documents using search engines or link lists in an increasing amount. In such contexts the document pops up as such, *in isolation*, even if you may have intended that people find out following links which you have carefully designed so that they give background information. When a user has e.g. found your document using AltaVista, he most probably wants to know what kind of document it is. Therefore, each HTML file should provide the very basic information (or link to information) about its origin and nature. For example, in a book-like document collection divided into small files, every file should contain at least a link to the "front page" of the "book".

At least the following origin information should be provided:

- The **author** of the document, specified so that the author can be identified uniquely. Providing a link to the author's home page is usually a good idea. If there are several authors, specify them all and the role of each of them; this may involve e.g. the original writer, the later editors, the current maintainer, and the person who is formally in charge of the document.
- The **date** of creation of the document, or the date of last update, or both. The date presentation should be uniquely understandable throughout the world; in particular, specifying months by their names is preferable.
- The **context** of the document and its **status**, such as being part of official documentation by a company about one of its products, or part of a private person's information about his hobbies, or whatever the case may be.
- The **address** (URL) of the document. Such information is often redundant, but it can be very valuable e.g. when someone sees just a paper copy of the document. It is better not to rely on a

browser (and a user) adding such information when paper copies are made.

The following document presents, in the form of a skeleton sample, one way of **implementing** such information; please study the applicable local recommendations before adopting this or some other particular style.

Example skel.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<HTML>
<HEAD>
<TITLE>A sample HTML document</TITLE>
<LINK REV="made" HREF="mailto:jukka.korpela@hut.fi">
</HEAD>
<BODY>
<H1>A sample HTML document</H1>
<P>
This is a sample HTML document exemplifying a suggested way
of presenting basic origin information.
</P>
<HR>
<P>
<A HREF="http://www.hut.fi/u/jkorpela/">Jukka Korpela</A>,
<a href="mailto:Jukka.Korpela@hut.fi">Jukka.Korpela@hut.fi</a>
<BR>
This document belongs to the context of
<a href="index.html"><cite>Learning HTML 3.2 by Examples</cite></a>
<BR>
The URL for this document is
<KBD>
http://www.hut.fi/u/jkorpela/HTML3.2/skel.html
</KBD>
<BR>
Created: 1997
</P>
</BODY>
</HTML>
```

Information about the document - the HEAD section

As mentioned, there are **two obligatory constructs** in HTML 3.2 and they must appear in this order:

- the construct

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

  (where you theoretically should have `HTML 3.2 Final` instead of `HTML 3.2`)
- the TITLE element, e.g.

  ```
  <TITLE>Introduction to General Absurdity</TITLE>
  ```

Most browsers don't complain if you omit these, but they are required by the HTML 3.2 definition. More importantly, there are good practical reasons to include them:

- The !DOCTYPE clause, which is a reference to a document type definition (DTD) in the SGML metalanguage, is very relevant when the document is processed by a general SGML browser (instead of a much more specialized program, an HTML browser, such as a typical WWW

browser). Moreover, specifying the *version* of HTML used in the document is useful to people who study your HTML code, and it might be relevant to WWW browsers and editors, too.
- The document name in the TITLE element is used for several useful purposes by browsers and other software. Typically, it is displayed in hotlists, results returned by search engines, etc.

Formally, the TITLE element is (at least implicitly) part of a HEAD element whereas the !DOCTYPE clause precedes all HTML constructs.

Optionally, the HEAD element may contain the following elements in addition to a TITLE element:

- an ISINDEX element (not used much any more)
- a BASE element, specifying the implicit base address of URL references
- META elements providing various metainformation, for example document expiration date
- LINK elements, which also provide metainformation but about the relationships of the document with other documents
- STYLE and SCRIPT elements; they are expected to be very important in the future but they are not usable yet (since both standardization and implementation is in progress).

Organizing the contents - headings, paragraphs, lists, etc

Generally, you divide your document into parts, which may in turn be divided into parts etc. In HTML, such division is expressed using **headings** of different level. The lowest-level parts in this hierarchy consist of one or more **paragraphs**. In addition to normal paragraphs and some special kinds of paragraphs like (long) quotations, HTML 3.2 supports **lists** and **tables**, which can be regarded as paragraph-like. The internal structure of paragraphs and paragraph-like elements is expressed using **text level tags**, to be discussed later.

The tags for expressing major structural features, so-called **block level tags**, are the following:

- headings of different levels: H1, H2, H3, H4, H5, H6
- paragraph level tags:
  - normal paragraph: P
  - quotation presented as separate paragraph: BLOCKQUOTE
  - author's address information as separate paragraph: ADDRESS
  - preformatted text to be displayed as such, preserving layout (lines, blanks): PRE
- lists:
  - normal unordered list: UL, LI
  - compact list of one-line items: MENU, LI
  - list of small items: DIR, LI
  - ordered list: OL, LI
  - definition list (labelled list): DL, DT, DD
- tables: TABLE, CAPTION, TR, TH, TD
- division of document into parts which may have their own layout properties (such as centering): DIV, CENTER
- change of topic: HR
- fill-out forms: FORM, ISINDEX.

A **recommendable approach**, which may need adjustments to fit your local recommendations, is the following:

1. Write a descriptive heading for the entire document and use H1 element with ALIGN=CENTER attribute for it.
2. Divide the document into major parts (sections), write suitable titles for them, using H1 with ALIGN=LEFT. In this and further divisions, try to avoid having more than seven parts.
3. If necessary, divide each major part into smaller parts with H2 headings, and if needed divide each of these subsections into subsubsections with H3 headings. Avoid using H4 headings and especially H5 and H6 headings, both because they are often rendered with a very small font and because more than three levels of structure tends to make the document hard to read. (If you still feel tempted to use H4, consider dividing the entire document into smaller documents.)
4. If you have a section with, say, H2 heading and containing H3 headings, avoid inserting text between the H2 heading and the first H3 heading. Such "homeless" text can be acceptable if it only contains *very short* notes such as general orientation, some remarks about the section, or a motto. Long homeless texts confuse the reader who does not see your good intentions; therefore, use a subsection with a heading of the appropriate level and with text like *Introductory remarks*, *Generalities* or *Summary*.
5. Divide the smallest parts of the above-mentioned structure into paragraphs or paragraph-like blocks (namely lists or tables). as described below. Notice that in HTML you must explicitly indicate paragraph division by HTML elements; leaving just an empty line does *not* cause a paragraph break.
6. Within paragraphs, use text level markup, normally phrase markup, to distinguish special text segments from normal text, e.g. to indicate quotations of computer output or to emphasize key words.
7. Add links and, if applicable, images or other illustrations.

As regards to the **paragraph level**, there are quite a many alternatives. The following list is intended to give some practical guidelines for selecting a suitable alternative:

- For *normal text* paragraphs, use the P element.
- However, if the text of the entire paragraph is literally *quoted* from some source, use the BLOCKQUOTE element or, if it is program code, computer output or some other text that shall be presented exactly honoring the division into lines, use the PRE element. In the latter case, if monospaced font is not suitable (e.g. the text is a poem), use BLOCKQUOTE and append a BR element to each line.
- As a special case, if the paragraph is intended for providing *information about the author* (that's you), use the ADDRESS element.
- For *itemized* information, which logically consists of separate cases or items, use various elements as follows:
    - For list of items where the order is not significant, such as a list of ingredients in a recipe, use the UL element, or the MENU element (for a list of rather small items), or the DIR element (for a large list of small items, suitable for presentation in multi-column format). However, since most browsers do not present MENU and DIR in the manner suggested in the specifications but as identical to UL, you may wish to consider other possibilities as well, such as using tables to represent menus. - Notice that e.g. in an alphabetical list the order is usually not significant in the relevant sense; if the order is so explicit, there is hardly any need to make it more explicit e.g. by using numbering.
    - For list of items where the order is significant and this significance needs to be made explicit, such as a sequence of instructions to be obeyed in that particular order, use the OL

element.
    - For a list of items with short titles or tags, such as a list of definitions for terms or abbreviations, use the DL element. However, you may like to consider using a TABLE element to present a definition list as an alternative.

  Notice that in a typical implementation MENU and DIR elements are rendered similar to UL elements. Moreover, DL element rendering can be awkward, too. Please browse a separate file *Examples of various list elements in HTML* to see what renderings of lists look like in your implementation. - The UL, MENU, DIR, OL, and DL elements are "plain lists" with no such structural feature as the CAPTION in a TABLE element. It is usually desirable to have some sort of heading or explanation before the list, but from the HTML viewpoint it is a separate paragraph.
- For *tabular information*, use normally the TABLE element, but consider the possibilities provided by PRE and DL elements, which may be suitable in special cases.

List can be nested in the sense that an item in a list, i.e. an LI (or DD) element, may in turn contain a list element.

Notice that the basic paragraph element P is not nestable, i.e. you cannot have P elements within a P element to create subparagraphs. However, the various list elements effectively provide an itemization structure which essentially corresponds to subparagraph division. Moreover, the list elements are nestable.

Text markup - emphasis, citations, code, etc

Logical vs physical markup

There are two major classes of text markup: logical and physical. Logical markup indicates the *role* of a text segment, such as being more important than normal text or being a citation. Physical markup is an instruction to present text in a particular manner, such as using a font of some specific kind or underlining.

Logical markup shall be preferred. Use physical markup only if it is really relevant that part of a text displayed in a particular physical way (if possible). The need for physical markup may arise when *referring to information in fixed presentation form*, such as text in a book or in an image. Such situations occur rarely.

For instance, use the STRONG element for strong emphasis, letting the various Web browsers express the emphasis in the way which is the best in the environment where they are used. Do not use the B element (indicating bolding), except in the rare occasions where you are writing *about* some text appearing in boldface somewhere or e.g. writing about mathematical vectors, for which is no adequate markup exists in current HTML.

When style sheets will be generally usable, both authors and readers will be able to affect the rendering (e.g. font, color, and background) of elements. For instance, someone might wish to have all program code extracts presented with yellow background and larger than normal font whereas someone might prefer some quite different methods of distinguishing them from normal text. Such operations will be much easier if logical markup has been used consistently.

In addition to being more flexible with respect to various browsers and rendering environments, logical markup has the following advantage over physical markup: In an increasing amount, computer programs are used for extracting information from HTML documents for various purposes like indexing. For this to work, it is much better to have logical markup indicating e.g. that some text is more important than the rest or a quotation of computer printout, rather than having designations of physical fonts.

Both logical and physical markup is done using HTML elements with start and end tags. It follows from the nature of HTML language that markups *must not overlap*. For instance, the following is in error:

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

On the other hand, markup elements *can be nested*. Browsers should do their best when rendering structures like the following:

Example nest.html:

```
This is <I>italic text which contains <U>underlined text</U>
within in </I> whereas <U>this is normal underlined text</U>.
```

Obviously, browsers with limited font repertoire can have difficulties in presenting text markup.

Phrase elements (logical text markup)

There are two phrase element for **emphasis**: EM and STRONG, and naturally STRONG is used for stronger emphasis. The HTML 2.0 specification requires that these elements be rendered as distinct from plain text and from each other; most browsers (excluding Lynx) seem to obey this.

Avoid emphasizing too much, since emphasizing everything is tantamount to saying everything with the same emphasis, i.e. not emphasizing anything! (The proverbial student who underlines everything in his textbook has not grasped the idea of emphasizing.)

Unfortunately there is no phrase element for "de-emphasis", i.e. for indicating segments of text as less important. If you really need that, you may consider using the SMALL element. But especially if the less important text is relatively long, it might often be a better idea to put it "behind hyperlinks", into separate documents to which there are links in the main document. A person who follows such a link is probably interested in the text, so he probably prefers seeing it as normal text, and there is no need for any de-emphasis.

The DFN element can be regarded as a special kind of emphasis, too, but logically it indicates that a term is used in a context where it is defined. This is a very useful element in principle but unfortunately many browsers, including Netscape, do not effectively support it.

The VAR element indicates that a piece of text (typically, a word) is a **variable**, i.e. a generic notation to be replaced by different actual expressions.

The other phrase elements involve different kinds of **citations or quotations**:

CITE	citation (title of a book or article or equivalent)
CODE	program code or equivalent (e.g. HTML code)
SAMP	sample output from programs, scripts, commands etc
KBD	text to be typed from a keyboard by a user; typically used when giving instructions

Please do not identify e.g. the concept of emphasis with its physical representation on your browser (or even its typical representation on several browsers). See below for notes and examples on rendering markup.

Font elements (physical text markup)

The available font elements - to be used very sparingly! - are:

TT	"teletype" text, i.e. monospaced text
I	italics
B	bold
U	underlined
STRIKE	strike-through text
BIG	large font
SMALL	small font
SUB	subscript
SUP	superscript

The HTML 2.0 specification says about the B, I and TT elements that where bold or italic typography or teletype font, respectively, is unavailable, "an alternative representation may be used". There is no explicit description of what this might mean, but there seems to be a general tendency to compare B to STRONG and I to EM

The FONT (and BASEFONT) element offers more possibilities to control font sizes than BIG and SMALL. However, all use of font size control in HTML should be avoided.

Rendering of markup

You may wish to view a separate file to see the visual appearance of the different markup elements on your browser. But please do not assume that the rendering which you see is universal or the correct one.

For example, some browsers (e.g. Internet Explorer) render TT (and CODE) so that the font is significantly smaller than normal text font, and this disproportion is preserved when the setting for font size is changed; moreover, Internet Explorer 3.0 renders VAR with monospaced font whereas most graphical browsers use (much more naturally) italics. On the other hand, in Netscape these font sizes are separately settable and by default the same font size is used for both, but "the same" is the technical size in points - in practise monospaced font looks bigger than normal proportional font!

Thus, *avoid messing up with font sizes*; use phrase markup and other structural elements and let the users, if they dislike the font sizes, define fonts in their browser settings the best they can.

The following table is intended for giving an idea of the variation. It (verbally) presents the rendering of markup elements in Netscape Navigator, Microsoft Internet Explorer, and Lynx. Notice that there is variation even within each of these programs - depending on version, platform, and system-wide or user's own configuration, so this is just a typical situation. Thus, consider this as *what different things might happen* rather than as a description of what actually happens in some particular program.

element	Netscape	Internet Explorer	Lynx
EM	italics	italics	underlined
STRONG	bold	bold	underlined
DFN	normal text	italics	normal (monospaced)
CODE	monospaced	monospaced small	normal (monospaced)
SAMP	monospaced	monospaced small	normal (monospaced)
KBD	monospaced	monospaced small	normal (monospaced)
VAR	italics	monospaced small	normal (monospaced)
CITE	italics	italics	underlined
TT	monospaced	monospaced small	normal (monospaced)
I	italics	italics	underlined
B	bold	bold	underlined
U	normal text	underlined	underlined
STRIKE	strike-through	strike-through	text between `[DEL:` and `:DEL]`
BIG	larger than normal	larger than normal	normal text
SMALL	smaller than normal	slightly smaller than normal	normal text
SUB	lowered, slightly smaller	lowered	normal text
SUP	raised, slightly smaller	raised	normal text

These relate to unnested elements. Nesting of text elements may affect the rendering.

Presenting interaction with computer

In order to present text-based interaction between a human being and a computer, or similar situations, the following approach can be used:

- computer output (whether it is prompts, normal output, or error messages) is within SAMP elements
- generic terms describing user input are within VAR elements
- actual user input is within KBD elements
- if computer program (source) code is quoted, it is within CODE elements.

In all cases, the principles on division into lines and the use of blanks and tabs must be taken into account, and this may require the insertion of BR elements or the use of PRE elements. Notice that logical markup is allowed within a PRE element (although possibly not implemented in a quite satisfactory way).

The following example illustrates the approach in the context of an introduction to the Perl programming language.

Example interact.html:

```
<P>The following Perl script prints out its input so that each line begins with
a running line number:</P>
<PRE><CODE>
#!/usr/bin/perl
$line = 1;
while (&lt;&gt;) {
  print $line++, " ", $_; }
</CODE></PRE>
<P>The scalar variable <CODE>$line</CODE> is of course the line counter.<P>
<P>The loop construct is of the form<BR>
<CODE>while (&lt;&gt;) {</CODE><BR>
<VAR>process one line of input</VAR> <CODE>}</CODE><BR>
</P>
<P>Assuming that you have written this script (the simpler version of it) into a
file named <KBD>lines</KBD>, you could test it using a command of the form<BR>
<KBD>./lines</KBD> <VAR>datafile</VAR><BR>
In particular, using the script as input to itself, you would do as follows
(the details of system output vary from one system to another):
</P>
<PRE>
<SAMP>lk-hp-23 perl 251 % </SAMP><KBD>./lines lines</KBD>
<SAMP>1 #!/usr/bin/perl
2 $line = 1;
3 while (<>) {
4   print $line++, " ", $_; }
lk-hp-23 perl 252 % </SAMP>
</PRE>
```

Notes on the example:

- nesting of text markup has been avoided
- although having the program code within a CODE element may seem unnecessary when it is within a PRE element, it is logical to do so, it should cause no harm, and it might one day prove useful (in a browser which uses different monospaced fonts for different purposes).
- similarly, using SAMP and KBD within the sample run *might* cause user input to be presented differently from computer output; using style sheets, you might even be able to specify the font, color, background and other properties differently for these logically different elements.

Controlling the layout

First, get the structure of your document right. Then, if needed, consider making the layout better. Notice that different browsers use different layouts, and even the same browser may display the same document differently in different environments. For instance, when the user changes the size of his Netscape window, the layout may change radically.

Thus, on the Web there is no such thing as *the* layout of a document. As an author **you cannot dictate layout**, just make some efforts to *affect* it. The following notes, and all information related to layout-oriented features of HTML, should be read with this in mind.

Several HTML elements have optional attributes which can be used to affect the way in which the element is rendered. Consult the detailed descriptions of individual HTML tags to see the possibilities and to read notes about them.

In particular, you may wish to *center* parts of the text to make them more distinguishable from normal text. You can use the ALIGN=CENTER attribute in several elements like P or DIV (or the separate CENTER element).

If you wish to separate major portions of your document visually from each other, you can use the HR element. Typically it is rendered as a full width horizontal line. But please use this in addition to structuring tools like headings, not as a substitute for them.

As regards to detailed layout issues such as forcing or preventing line breaks, see section *Division into lines and the use of blanks and tabs*. Font issues were discussed above.

Links

In this section:

- The link concept
- Normal links
- Expressing the nature of a link
- Link types (REL values)
- Practical guidelines for setting up normal links
- Audio and video resources
- Links to binary and plain text files
- Links to other non-HTML resources

The link concept

Links (often called *hyperlinks*) are the feature which justifies the *HT* in *HTML* (HyperText Markup Language).

Technically links are specified using A (anchor) elements, and some technical issues are discussed in the description of the A element. In principle, some other HTML constructs define links, too. In particular, the LINK element defines a link between two documents.

A link is a directed connection between a particular point in a document and another particular point in the same or another document or another resource on the Internet. The points are often called *anchors* in HTML terminology.

The two ends of a link (the anchors) are in different logical positions: the link is *from* one point *to* another. The latter, called the *target* of the link, is very often the beginning of a document or, perhaps more logically speaking, an entire document.

Normal links

In the simplest case, you create a link from one point of your HTML document to another HTML document (which could be your own or written by someone else, perhaps physically located at the other side of the globe). You have to decide which words act as a visual representation of the link, i.e. as the phrase which refers to the other document, and you need to know the Web address (the URL) of that document. Then you just put the pieces together into a suitable A element. For instance:

```
I work at <A HREF="http://www.hut.fi/english.html">HUT</a>.
```

This might, in one environment, be rendered as follows:

I work at HUT.

The *link text*, here the abbreviation HUT, acts as a link to a Web document which explains what the abbreviation means and also provides a lot of information about it. The rendering of links varies a lot - the link text might be underlined, colored, or otherwise distinguishable from normal text. In particular, underlining is not part of the link concept, just one possible rendering. The user(reader) is assumed to know how links are rendered in the particular environment.

The user should also be assumed to know how to use his browser in order to *follow a link*, i.e. to access the linked resource. There is hardly any reason to tell the user to click something, especially since clicking is by no means the only possible method for following links. (A keyboard or some other pointing device than a mouse or spoken commands might be used in different environments.)

It is also browser-dependent whether and how the linked resource can be viewed in a new window, stored into a file, passed as data to some application, or processed in some other way.

In particular, as an HTML author you need to do nothing to allow the user to *open a document in a new window*. In a typical graphical browser, such as Internet Explorer or Netscape, the user can use the rightmost button of the mouse for the purpose when selecting the link (instead of the normal use of the leftmost button), then select the alternative `Open in New Window` in the pulldown menu opened. Naturally, the user could then move that window to another position on the screen and resize it suitably. If you think that most of your users don't know such things, you may give a note about them, when you have a link for which such a treatment might be especially suitable (such as in the *To whom?* section of this document). But any references to such issues of browsing techniques should be kept to minimum in Web documents, unless such techniques are their main topic!

Expressing the nature of a link

Typically people assume that a link refers to some *general information* about the thing expressed as the link text. Thus, the link HUT in our example leads to the home page of HUT (which hopefully explains what the abbreviation stands for, what the institution is, etc).

If the nature of a link is different, you can try to help the user understand it, allowing him to make a rational decision whether to follow the link or not. There are actually several ways for doing that, and in some cases you might use all of them:

- Include explanatory text e.g. in parentheses after the link. This works more universally than the other ways, which are not very widely supported by browsers yet, but it often looks clumsy and redundant.
- Include a REL attribute into your A element. This is a good idea if there is a suitable REL value for the purpose, e.g. REL=Glossary when the linked document is a glossary for the current document.
- Include a TITLE attribute into the A element. See Jacob Nielsen's excellent alertbox *Using Link Titles to Help Users Predict Where They Are Going*.

At least one the methods described above should be used at least if the link is of *unexpected* nature, such as referring to very detailed technical reference in a tutorial or referring to an example only in a context where the reader would expect the link to point to general information. Examples:

Example rfcref.html:

```
<P>The expiration time must be expressed in one of a few strictly
defined formats (as defined formally in
<A TITLE="The official specification of the Internet E-mail protocol"
   HREF="ftp://ds.internic.net/pub/doc/rfc/rfc822.txt">RFC 822</a>
and
<A TITLE="Requirements for Internet hosts"
   HREF="ftp://ds.internic.net/pub/doc/rfc/rfc1123.txt">RFC 1123</a>).
</P>
```

Example hobbies.html:

```
My interests include comic books (especially
<A TITLE="A large Calvin and Hobbes site, with a daily strip
  and a lot more"
  HREF="http://www.calvinandhobbes.com/">Calvin &amp;amp; Hobbes</a>),
<A TITLE="Some material, by me and by others, about the Finnish
  language, mostly in English"
  HREF="http://www.hut.fi/u/jkorpela/Finnish.html">Finnish language</A>,
and
<A TITLE="My experiences on making beer and wine at home;
  in Finnish, with a short summary in English"
  HREF="http://www.hut.fi/u/jkorpela/beerwine.html">making beer and wine</A>.
```

Note: Internet Explorer 4.0 shows the TITLE texts when cursor is on a link, but only after a delay of a second or two.

Link types (REL values)

As regards to REL and REV values in general, there is no official requirement on browsers to support any particular set of values. The idea of specifying the nature of a link with REL and/or REV has been in HTML from the early days, but it has not been deployed much. Probably, and hopefully, things will change.

The following table lists some REL values which authors can use, expecting them to be supported by at least some browsers now or in the near future. They were listed in an Internet Draft (*draft-ietf-html-relrev-00.txt*, now expired) in 1996 as well as in the *HTML 3.2 Reference Specification*, and they have been included into the list of link type values in the HTML 4.0 specification. (See also W3C working draft *Hypertext Links in HTML*.)

attribute setting	type of link (role of linked resource)
REL=CONTENTS	A document serving as a table of contents.
REL=INDEX	A document providing an index for the current document.
REL=GLOSSARY	A document providing a glossary of terms that pertain to the current document.
REL=COPYRIGHT	A copyright statement for the current document.
REL=NEXT	The next document to visit in a guided tour.
REL=PREV	The previous document in a guided tour. (This value was originally named PREVIOUS.)
REL=HELP	A document offering help, e.g. describing the wider context and offering further links to relevant documents. This is aimed at reorienting users who have lost their way.
REL=BOOKMARK	A bookmark, used to provide direct links to key entry points into an extended document. The TITLE attribute may be used to label the bookmark. Several bookmarks may be defined in each document, and provide a means for orienting users in extended documents.

In conjunction with style sheets, a LINK element with **REL=STYLESHEET** can be used.

Practical guidelines for setting up normal links

Although it is technically easy to set up links, it is pragmatically often difficult to use them the right way. Here are some practical guidelines:

- *Avoid excessive linking*. If every word in your document is a link, the reader does not know which are the *useful* links.
- When you use e.g. an *abbreviation or technical term* which is not explained in your document, try to find a suitable document which gives some explanation to which you can link. (In this context, you might use a TITLE text which is simply an expansion of an abbreviation. This might suffice to many readers, so that they need not follow the link.) Whether this should be made at the first occurrence only or at every occurrence depends on the circumstances.
- Similarly, it is often a good idea to link to *organizational and personal home pages* (if available) when mentioning an organization or person.
- Naturally, when *citing a document*, provide a link to its online version (or at least to online bibliographic information about it) if such information exists on the Web.
- Often you have *"footnote-like"* information which you wish to make available via the Web but which is of less importance (to most readers at least) than your main document. As an alternative to using the SMALL element, consider making it a separate HTML file (or set of files) and attach e.g. a *Further reading* section to the main document, providing suitable links. This applies especially to technical details which might be useless and even irritating to the majority of your readers, yet valuable to some readers.
- If you would like to *link to several documents* from a point (e.g. when mentioning a computer program name, you would like to link to a short description of it, to a full manual, to an FTP site for downloading etc.), create a small file containing those links with suitable explanations and

link to it. This gives you the additional option of providing links to copies of the same information, such as downloading links to alternate FTP sites.
- Try to make the link text *short but descriptive*.
- Link to *relevant and reliable* information only. Linking to bad-quality documents might be worse than no links, since people would waste their time with them instead of searching for newer and better documents on the Web.
- When referring to issues outside your main themes, try to link primarily to *short, clearly written documents* which contain links to more detailed and technical information. For example, avoid linking directly to an ISO standard or an RFC in a document written for a wide audience.

Audio and video resources

From the HTML viewpoint, linking to audio and video resources is easy: simply provide a link to a file which is in a suitable format. (This leaves it to the user to follow the link in order to listen or see the material. In HTML 3.2, there is no way to *force* audio or video to autoload and autostart.)

Example march.html:

```
<P>This page is so solemn that you may wish to listen to the
<A TITLE="[0.5 Mb .au]"
 HREF="http://www.tpk.fi/sound/porilaisten_marssi.au">
<CITE>Pori March</CITE></A> while reading.</P>
```

It depends on the browser how references to resources like audio and video files are handled. If a browser supports them, it typically supports some particular repertoire of file formats by initiating ("launching") a separate program for "playing" the file. (It might use a distinct program for each file format or a general-purpose media player program for a large set of formats.)

Thus, for example, in order to listen to `.au` files the user needs, in addition to suitable hardware installed, a program which can produce sounds interpreting that specific format, and user's browser must have settings which instruct it to launch that player program for those files.

In principle, the Web server on which the file resides should be configured to send the correct media type for the file. In practice it mostly does that on the basis of the file name extension (and some browsers incorrectly rely on that extension more than on the media type information).

See also: *Guidelines for HTML Writers of MIDI File Pages* by Charles Kelly.

Links to binary and plain text files

You can make plain text and binary files of various formats available to other people alongside with your HTML files, and you can tell about them and provide links to them in your HTML documents.

Similarly to audio and video files (which are actually special cases of binary files), quite a many things must be set up (outside the HTML document) both in the server and in users' browsers for this to work. If your server does not support the file format involved, you can try to use some widely known format and corresponding file name suffix; see also *WDG Web Authoring FAQ*, questions 5 and 6.

Of course, such links will be useful only to such people who can use a program which processes the particular file format in a meaningful way. A *plain text* file is normally displayed as such by a browser, at least if the line length is reasonable (preferably less than 80 characters). The processing of

a *binary* file might consist of displaying an image or animation, playing music, or doing some spreadsheet calculations, for example. This might take place within a browser or in a separate program launched automatically by a browser (when programmed to do so), or "offline" so that the Web browser is used just to retrieve the file and to save it into a local file, to be opened later by an application.

Example:

```
The budget proposal is available as a
<A HREF="budget.zip">zipped Excel file</A>
```

People using computers on which Excel is available will then be able to view your document on it. It depends on the browser and its settings how smoothly this can take place. Of course they also need some program (e.g. WinZip or WiZ) for unfolding a .zip file, but such software exists for almost all environments and is very useful to have installed anyway. The reason for my suggesting the use of zipped format in problematic cases is twofold:

- Web servers can usually process `.zip` files appropriately, telling browsers that they are binary files. Various application program formats cause trouble much more likely, especially if the particular format is not normally used in the computer system on which the server runs. For instance, the correct media type for an Excel file is `application/vnd.ms-excel`, but either your server or user's browser might not recognize it.
- Zipping can save time and space, and it can be used to pack several closely related files (such as a binary program and its documentation and data files) into a single file, making downloading easier and faster.

Links to other non-HTML resources

In principle, a link can refer to other Internet resources as well, such as a Usenet newsgroup, an E-mail address, or a file in an anonymous FTP server. See the description of URLs for the syntax as well as some remarks. Here we will discuss just one special case, links to E-mail addresses.

You can use a `mailto:` URL in the HREF attribute. of an A element. Example:

```
My E-mail address is <A HREF="mailto:Jukka.Korpela@hut.fi">
Jukka.Korpela@hut.fi</A>.
```

(Please avoid constructs like `<A HREF="mailto:`*address*`">Mail me!</A>` which are useless e.g. when reading a paper copy of the document.) Selecting such a link typically means that the browser invokes an E-mail composer, with the recipient field prefilled. It is *not* possible to prefill other fields in any reliable way. See question *How do I specify a subject for a mailto: link?* in *WDG Web Authoring FAQ* and *Special Notes on the Mailto: argument* in the *HTML Compendium*. Use forms instead of simple `mailto:` links if you want to prefill something. (I have written a very simple example of such a form.)

Images, formulas, etc.

Basically, the image support in HTML is just an *interface* to the world of graphics. The creation and manipulation of images, the graphics formats and other graphics stuff is not part of HTML. In particular, the HTML specification does not pose any requirements or restrictions on the graphics formats supported by Web browsers.

Assuming that we have some graphics in some format in a file, there are two essentially different ways to use it in a Web document. You can either **link** to it or to **embed** it into your document. In the first case, you use an anchor (A) element; in the latter case, an IMG element. In the first case, when a user accesses your document he sees e.g. a verbal phrase which acts as a link, and activating that link causes an image to be displayed, either in the same window or in another, depending on the browser and its settings. On the other hand, an embedded image is part of your document; when a user accesses your document, the image is loaded along with it and displayed as part of it.

In both cases, the user will see the image only if the browser supports the particular graphics format. The most commonly supported formats are GIF and JPEG. They are often the only formats supported for embedded images. As regards to choosing between these formats, the document *Image Use on the Web* by WDG gives the following rule of thumb: "Lots of colors, JPEG... Solid colors or no gradations, GIF."

For linked images, the support is typically wider (it might include e.g. PostScript, PDF, and PNG) and extensible by the user (by installing new viewers and making suitable additions to the settings of the browser). The reason is that linked images are typically implemented so that the browser knows nothing of the graphics format itself but only knows how to launch a separate program to present it. It is even possible that a browser cannot process embedded images at all but can launch a separate viewer for linked images.

Thus, *referencing a graphic by linking (A element) provides better accessibility than using embedding (IMG element)*. If the graphic is essential, this might be important. In fact, the HTML 2.0 specification recommends using A for essential graphics, IMG for non-essential. Note: on browsers which support embedded images, it is usually possible to open a document in a new window; thus, the usefulness (or necessity) of seeing an image alongside with the rest of the document need not be an objection to using linking.

As a special case, it is possible to combine linking and embedding in a sense: you can create a document which contains an image which acts (instead of verbal link text) as a link to another image. Typically, the embedded image ("thumbnail") is rather small, stamp-like, often a small coarse version of the image to which it points as a link.

Linking to an image is usually permitted without specific permission. On the other hand, embedding an image means using it in a way which requires the author's permission, and the author must be mentioned. (See *Web Law FAQ*.) Obviously, some images are so simple that copyright is not applicable. Moreover, there is a large number of collections of images, some of which are in the public domain.

To illustrate linking to images and embedding images, let us consider a GIF image which has been put onto a suitable place so that it is accessible using the URL http://www.hut.fi/%7elsarakon/sae.gif. Now I could refer to it in the following way:

Example sae.html:

```
<A HREF="http://www.hut.fi/%7elsarakon/">Liisa Sarakontu</A> has drawn
<A HREF="http://www.hut.fi/%7elsarakon/sae.gif">a picture of
Siamese algae eater</A>.
```

On the other hand, since Liisa has given me the permission to do so, I could embed the image into a document of mine as follows:

Example sae-2.html:

```
The Siamese algae eater (<I>Crossocheilus siamensis</I>) is often
mixed up with another algae eating fish, the "false Siamensis"
(<I>Garra taeniata</I> or <I>Epalzeorhynchus sp.</I>). Below you
can see drawings of them by
<A HREF="http://www.hut.fi/%7elsarakon/">Liisa Sarakontu</A>.
<P>
<IMG SRC="http://www.hut.fi/%7elsarakon/sae.gif" ALT="[Picture of Siamese
algae eater]">
<P>
<IMG SRC="http://www.hut.fi/%7elsarakon/false.gif" ALT='[Picture of "false
Siamensis"]'>
```

The issue of *good* use of images is very difficult any many-faceted. No attempt to cover it will be made here. I have written a separate treatise *How to use images in communication in general and on the Web in particular*.

There is no general support in HTML 3.2 to presenting **mathematical formulas**. Consult the W3C document on *Math Markup* to see what work is in progress in this respect. However, you can use some software (e.g. TeX) to produce the representation of a formula as an image, e.g. in PostScript form, and use the IMG tag to embed it into your document or the A tag to create link to it. The latter method is often worth considering, especially for large formulas. The reader may prefer reading the text without distractions and looking at the formula (image) at the very moment he is prepared to do so. Moreover, he may prefer looking at it in a separate window (which is separately adjustable in size and positionable on the screen).

In some cases, when just *a few separate symbols* are needed within the text and they have reasonable textual alternatives, the following kind of approach can be suitable:

Example sigma.html:

```
The Greek letter <IMG SRC="http://www.ece.cmu.edu/icons/Sigma.xbm"
ALT="capital sigma"> is often used to denote summation.
```

There is a problem, however: since an image has fixed dimensions whereas the size of letters is browser-dependent, there might be an unesthetic disproportion. - Notice that HTML 4.0 contains a relatively large set of escape sequences for various symbols, including Greek letters as used in mathematics. However, it will take time before browsers in use generally support them.

Sometimes it is best to present mathematical expressions in *linearized notation*. For example, instead of trying to find a way of presenting the square root of 2 in the normal mathematical way, you might write just sqrt(2). It depends on intended audience whether you need to explain such notations.

If you use images for presenting mathematical formulas, try to formulate an alternate purely textual (linearized) presentation, which you would then include as the value of the ALT attribute. A simple example:

Example math.html:

```
Assignment 42. Compute
<IMG SRC="integral.gif" ALIGN=MIDDLE
 ALT="the integral of exp(x**2) from 0 to infinity.">
```

More information: *Math in HTML (and CSS)*.

Tables (Not in HTML 2.0!)

Index:

- The table concept in HTML 3.2
- Tags used to represent tables
- The very basic table structure
- Additional features; a typical table with text cells
- Parallel texts
- Using a table to present a definition list
- Numerical tables
- Using tables to represent menus
- Table elements occupying several rows or columns
- Nested tables
- Alignment of cells
- Fonts in table elements

See also Dianne Gorman's excellent *Introduction to Tables* (part of her *Introduction to HTML*) and Joe Barta's helpful Table Tutor, which carefully explains the basics but later discusses advanced issues too.

The table concept in HTML 3.2

In HTML, a table is a structure consisting of *rows* and *columns*, which can have *headers* (names, titles, explanations). A table is typically rendered in some natural way corresponding to the structure, with columns adjusted accordingly. The components, or *cells*, of a table may contain any text elements or even block elements and headings. Thus, table element might be a number, a word, a text paragraph, an image, or something more complicated.

Table cells are often called table elements, but it is best to avoid that in the HTML context, since it might cause confusion e.g. with the TABLE element, which is the HTML description of an entire table.

Tables are the most important improvement in HTML 3.2 in comparison with HTML 2.0. On the other hand, the table constructs of HTML 3.2 are only a subset of the draft *The HTML3 Table Model* (RFC 1942).

Tables are supported by most browsers, but there are many problems with the support, especially if you use large and complicated tables.

Text-only browsers and speech-based user agents will always have difficulties with complicated tables, for obvious reasons. See Alan Flavell's review *Tables on non-table browser* for information about making tables look somewhat reasonable, if possible, also on browsers which do not support tables. See also Web Accessibility Initiative.

Authors often use table elements just to get a desired *layout* of pages, not to represent data which is logically matrix-like in structure. This reduces accessibility and causes other problems. The HTML 4.0 specification explicitly says that "tables should not be used purely as a means to layout document". For detailed arguments, see e.g. *Tables* in *Dan's Web Tips*.

Tags used to represent tables

Representing a table involves several kinds of HTML tags:

- TABLE tags, which surround the entire table specification
- an optional CAPTION element specifying the caption (name) of the table
- TR tags, which specify the table rows
- TH tags, which specify table row and column headers
- TD tags, which specify the data in the table, i.e. the contents of table cells.

The very basic table structure

Let us start with a very simple example. It consists of a 2 by 2 table of numbers (a unit matrix), with no headers whatsoever. The HTML code is as follows:

Example table1.html:

```
<TABLE>
<TR> <TD> 1 </TD> <TD> 0 </TD> </TR>
<TR> <TD> 0 </TD> <TD> 1 </TD> </TR>
</TABLE>
```

and it looks like the following on a typical browser:

1	0
0	1

Thus, the TABLE tags enclose the table rows, each of which is enclosed by TR tags and enclose table cells enclosed by TD tags. This corresponds to the logical structure of a table as a set of rows consisting of cells. You can abbreviate the table structure by omitting the TD and TR end tags (since a browser implicitly assumes them), but at the expense of losing the logical clarity to some extent:

```
<TABLE>
<TR> <TD> 1 <TD> 0
<TR> <TD> 0 <TD> 1
</TABLE>
```

Moreover, although omitting those end tags is legal HTML 3.2, it may in practise confuse some browsers (including Netscape) in some cases.

According to the specifications, the use of blanks and newlines in the HTML code for a table is irrelevant to the visual appearance of a table when viewed with a browser, since that appearance is controlled by HTML tags. However, it is often useful to position table elements suitably in the HTML code so that items in the same column are adjusted to make the structure clear for *you* (or whoever has to maintain the HTML document). On the other hand, Netscape is known to violate the specifications: blanks or newlines between tags may affect the presentation (causing, for example, extra space between the content of a cell and its border)

Additional features; a typical table with text cells

There are several separate features which you will often like to add to this simple table model:

- A *caption* for the table, attached to the table itself (as opposite to telling about the table in the normal text of the document).
- *Headers* (explanations) for table rows or columns or both.
- *Borders* around the table and each table cell.

The following, rather typical, example uses all of the above-mentioned features:

Example table2.html:

```
<P>An illustration of the use of the TABLE element in HTML.</P>
<TABLE BORDER=1>
<CAPTION>Finnish, English, and scientific names for some animals</CAPTION>
<TR><TH>Finnish name</TH><TH>English name</TH><TH>Scientific name</TH></TR>
<TR><TD>hirvi</TD><TD>elk</TD><TD><I>Alces alces</I></TD></TR>
<TR><TD>orava</TD><TD>squirrel</TD><TD><I>Sciurus vulgaris</I></TD></TR>
<TR><TD>susi</TD><TD>wolf</TD><TD><I>Canis lupus</I></TD></TR>
</TABLE>
```

Notice that some table elements in the example contain text markup; in this case, there is a specific reason for using the I element.

Parallel texts

If you have logically parallel texts, such as a document in several languages or several variants of the same text, the TABLE element is probably the best way of presenting them. (Using a PRE element is possible but requires tedious formatting by hand and results in the text being displayed in monospaced font.)

In the simplest case you can just write a TABLE element (with attributes defaulted) which contains a single row which contains two data cells, each of which contains a paragraph.

In a more general case, you should divide the parallel texts into logical parts, such as paragraphs, and make each part a cell of the table. This may require a lot of work (unless you have a suitable program to do the job), since you must take care of "merging" the text: after the first part of the first text, you must have the first part of the second text, etc.

The following example presents a passage from the Bible in three versions and translations:

Example table3.html:

```
<TABLE>
<CAPTION><STRONG>The beginning of Genesis
in three languages</STRONG></CAPTION>
<TR ALIGN=LEFT VALIGN=TOP>
<TH></TH><TH>Latin (Vulgate)</TH><TH>English (King James version)</TH>
<TH>Finnish (1992 version)</TH>
</TR><TR ALIGN=LEFT VALIGN=TOP>
<TH>1</TH>
<TD>In principio creavit Deus caelum et terram.</TD>
<TD>In the beginning God created the heaven and the earth.</TD>
<TD>Alussa Jumala loi taivaan ja maan.</TD>
</TR><TR ALIGN=LEFT VALIGN=TOP>
```

```
<TH>2</TH>
<TD>Terra autem erat inanis et vacua et tenebrae super faciem
abyssi et spiritus Dei ferebatur super aquas.</TD>
<TD>And the earth was without form, and void;
and darkness was upon the face of the deep.
And the Spirit of God moved upon the face
of the waters.</TD>
<TD>Maa oli autio ja tyhjä, pimeys peitti syvyydet,
ja Jumalan henki liikkui vetten yllä. </TD>
</TR><TR ALIGN=LEFT VALIGN=TOP>
<TH>3</TH>
<TD>Dixitque Deus "Fiat lux" et facta est lux.</TD>
<TD>And God said, Let there be light: and there was light.</TD>
<TD>Jumala sanoi: "Tulkoon valo!" Ja valo tuli.</TD>
</TR></TABLE>
```

Notice that the ALIGN and VALIGN attributes can be essential for achieving good rendering. Browsers cannot know the nature of tables from their contents, so there are situations where the document author may need to control formatting issues like alignment.

Using a table to present a definition list

As mentioned in the discussion of list elements like DL, the typical rendering of "definition lists" is not very good. Moreover, there are just a few ways to affect the rendering.

Using a TABLE element for a definition list is perhaps not an intended use of that element but it is often useful, especially since the author can control things like alignment and use of borders. Consult the document *Examples of various list elements in HTML* for a very simple example of presenting a definition list as a table with default attribute settings. Usually you probably want the "definition terms" to be left-aligned, as in the following example:

Example table4.html:

```
<TABLE>
<CAPTION>The first three letters of the Greek alphabet</CAPTION>
<TR><TH ALIGN=LEFT>alpha</TH>
<TD> the first letter of the Greek alphabet </TD> </TR>
<TR><TH ALIGN=LEFT>beta</TH>
<TD> the second letter of the Greek alphabet </TD> </TR>
<TR><TH ALIGN=LEFT>gamma</TH>
<TD> the third letter of the Greek alphabet. </TD> </TR>
</TABLE>
```

Numerical tables

For many people, tables are essentially tables of numerical data. As the preceding examples show, tables have a lot of other use as well.

For numerical tables, proper *alignment* is usually crucial for easily readable rendering. (It is in a sense a structural feature, since it relates to the comparability of items of a column.)

Integer values in a column should be right aligned. This is easy to achieve in principle. There are two alternatives:

- use the ALIGN=RIGHT attribute in every TD element, *or*
- use the ALIGN=RIGHT attribute in every TR element and override it with ALIGN=LEFT or ALIGN=CENTER in TH elements if appropriate.

Values containing a decimal point (or, in many languages, a decimal comma) should be aligned according to that separator, but unfortunately this is not possible in HTML 3.2. (There are suggested ways of expressing such requests, but currently there is little if any support for them.) One solution is to present such values so that there is the same number of digits to the right of the decimal point in every value in a column, and use ALIGN=RIGHT.

However, the rendering might be unsatisfactory if numbers are presented using a proportional font so that digits are of essentially different sizes. It is possible but tedious to overcome this by putting the data in each numerical cell within a TT element. (Notice that it is not legal for a TT element to contain a TABLE element!)

The following example contains first a hand-formatted table presented using the PRE element, then the same data using a TABLE element. In general, it takes more work and care to use a TABLE element but the result is often much better.

Example table5.html:

```
Measurement results:
<PRE>
time     temperature   pressure
12:00       26           12.8
12:15       22.5          9.8
12:30       11            1.65
12:45        3.3          0.03
13:00        0.05         0.002
</PRE>

<TABLE>
<CAPTION>Measurement results</CAPTION>
<TR><TH>time</TH><TH>temperature</TH><TH>pressure</TH></TR>
<TR ALIGN=RIGHT><TD>12:00 </TD><TD>26.00 </TD><TD>12.800 </TD></TR>
<TR ALIGN=RIGHT><TD>12:15 </TD><TD>22.50 </TD><TD> 9.810 </TD></TR>
<TR ALIGN=RIGHT><TD>12:30 </TD><TD>11.00 </TD><TD> 1.650 </TD></TR>
<TR ALIGN=RIGHT><TD>12:45 </TD><TD> 3.30 </TD><TD> 0.030 </TD></TR>
<TR ALIGN=RIGHT><TD>13:00 </TD><TD> 0.05 </TD><TD> 0.002 </TD></TR>
</TABLE>
```

Using tables to represent menus

Very often one needs to present a **relatively large set of relatively small items**. For instance, suppose that we have documents about various countries and we wish to provide a menu of country names, to be used as an index.

The index is implemented in HTML using normal links, e.g.
`<A HREF="af.html">Afghanistan</A>`
What we will discuss here is how to present the link names, or some other pieces of text, as a list, table, or some other structure.

If you only read HTML specifications, the obvious answer is to use the DIR or MENU construct. However, as mentioned and exemplified in the general discussion of lists, this is not practically feasible. Thus, if we prefer having the menu in multicolumn format, as we usually do, we must use

other constructs.

Pseudo-table: preformatted text

One possibility is to format the menu by hand and enclose it into a PRE element. If the menu items are link texts, you should first format it as text only, then add the anchor (A) tags, since adding them obscures the layout. For clarity, therefore, the following example is presented without links (unlike the other alternatives):

Example menu1.html:

```
<PRE>
Afghanistan           Albania               Algeria
American Samoa        Andorra               Angola
Anguilla              Antarctica            Antigua and Barbuda
Arctic Ocean          Argentina             Armenia
</PRE>
```

Using just characters as separators

Another possibility, which should be the normal one, is to present the items simply as a text paragraph, using e.g. a blank or a blank and a comma as separator. Thus, you would *plain text characters* and not HTML markup to separate the items; not very structural, but it often works well. In this approach, the browser takes care of dividing the text into lines and the presentation is very compact:

Example menu2.html:

```
<BASE HREF="http://www.odci.gov/cia/publications/factbook/">
<P>
<A HREF="af.html">Afghanistan</A>,
<A HREF="al.html">Albania</A>,
<A HREF="ag.html">Algeria</A>,
<A HREF="aq.html">American Samoa</A>,
<A HREF="an.html">Andorra</A>,
<A HREF="ao.html">Angola</A>,
<A HREF="av.html">Anguilla</A>,
<A HREF="ay.html">Antarctica</A>,
<A HREF="ac.html">Antigua and Barbuda</A>,
<A HREF="xq.html">Arctic Ocean</A>,
<A HREF="ar.html">Argentina</A>,
<A HREF="am.html">Armenia</A>
</P>
```

Of course, it is possible to force line breaks by using a BR element (e.g. to make a change in the initial letter cause a new line in an example like above). If you think the items are not distinguishable enough in the rendering, consider prefixing each item with a special character like * (and using just spaces as separator) or using | and spaces around it as separators.

If we'd strongly prefer a presentation where **items occupy the same amount of horizontal space**, then one can either use the PRE method described above or take the effort of designing a suitable TABLE element. Example of the latter:

Example menu3.html:

```
<BASE HREF="http://www.odci.gov/cia/publications/factbook/">
<TABLE><TR>
<TD WIDTH=160><A HREF="af.html">Afghanistan</A></TD>
<TD WIDTH=160><A HREF="al.html">Albania</A></TD>
<TD WIDTH=160><A HREF="ag.html">Algeria</A></TD>
<TD WIDTH=160><A HREF="aq.html">American Samoa</A></TD>
</TR><TR>
<TD WIDTH=160><A HREF="an.html">Andorra</A></TD>
<TD WIDTH=160><A HREF="ao.html">Angola</A></TD>
<TD WIDTH=160><A HREF="av.html">Anguilla</A></TD>
<TD WIDTH=160><A HREF="ay.html">Antarctica</A></TD>
</TR><TR>
<TD WIDTH=160><A HREF="ac.html">Antigua and Barbuda</A></TD>
<TD WIDTH=160><A HREF="xq.html">Arctic Ocean</A></TD>
<TD WIDTH=160><A HREF="ar.html">Argentina</A></TD>
<TD WIDTH=160><A HREF="am.html">Armenia</A></TD>
</TR></TABLE>
```

Alternatively, you might wish to consider the effect of using a table with borders.

Notice that this solution is rather **unclean**. It involves a TABLE structure where the division into lines is (normally) made for layout purposes only, and *adding new items usually requires complete restructuring of the table*. You typically need to insert WIDTH attributes to ensure that table columns are of the same width, and the specification is inherently device-dependent since it must be given in pixels. In particular, the presentation might not be the desired one if the physical font size in pixels differs too much from what you think it should be. Moreover, the larger the sum of WIDTH attribute values for a row, the more probable it is that the presentation does not fit into a browser window without horizontal scrolling or, depending on the browser, without the browser deviating from the WIDTH suggestions, messing up the table.

Thus, this approach should be avoided in general, especially since it makes the document window width dependent. Hopefully future browsers will support the UL element in a more advanced way, automatically selecting a compact multicolumn presentation when applicable, or at least support the DIR element in the intended way.

A flexible pseudo-table

There is a trick which is a modification of the using just characters as separators between items but creates, in most browsing situations, the appearance of a table. The "table" even **adapts to the browser window width** so that the division of items to rows changes. Applied to our example, this works rather well. (I use this technique for the menu of element names in my *Quick index to HTML 4.0 specification*).

The idea is to **pad the items with trailing no-break spaces** so that each item has the same number of characters and use normal spaces (or newlines) between the items. Additionally, use some markup which causes monospace font to be used, such as TT. As a consequence, most browsers will treat the items as chunks of equal size and format the paragraph rather nicely. **Drawbacks**: monospace, or "teletype", font is not that nice, and implementing the trick is tedious (but you could use some authoring tool to generate the HTML document from some simpler format).

I call this a "trick" because it does use logical markup and because there is no *guarantee* that it works. On the other hand, it is syntactically valid HTML, and in the rare cases where it does *not* work (since a browser "collapses" no-break spaces), the situation is no worse than when using just spaces as separators.

Table elements occupying several rows or columns

Sometimes we would like to make a table element occupy the space for two or more elements, horizontally or vertically or both. As an example, consider the following information (the declination of a Latin pronoun):

```
      neut. masc. fem.

nom.  id    is    ea
acc.  id    eum   eam
gen.  eius  eius  eius
dat.  ei    ei    ei
abl.  eo    eo    ea
```

Obviously this calls for using a table in HTML, and using the above-explained constructs you can write a simple table presentation for the data. However, if you would like to make it more explicit that there are identical entries in adjacent cells, you can use the ROWSPAN and COLSPAN attributes as follows:

Example span.html:

```
<TABLE BORDER=1 ALIGN=CENTER CELLPADDING=3>
<CAPTION>Declination of <I>is</I> in singular</CAPTION>
<TR><TH></TH><TH>neuter</TH><TH>masc.</TH><TH>fem.</TH></TR>
<TR><TH>nom.</TH><TD ROWSPAN=2 VALIGN=MIDDLE><I>id</I></TD>
 <TD><I>is</I></TD><TD><I>ea</I></TD></TR>
<TR><TH>acc.</TH><TD><I>eum</I></TD><TD><I>eam</I></TD></TR>
<TR><TH>gen.</TH><TD COLSPAN=3 ALIGN=CENTER><I>eius</I></TD></TR>
<TR><TH>dat.</TH><TD COLSPAN=3 ALIGN=CENTER><I>ei</I></TD></TR>
<TR><TH>abl.</TH><TD COLSPAN=2 ALIGN=CENTER><I>eo</I></TD>
 <TD><I>ea</I></TD></TR>
</TABLE>
```

For example, the first cell is specified to have ROWSPAN=2, which effectively means that two adjacent cells in the same column are combined into one cell. Notice that when writing the HTML code for the next row (the second TR element) we simply leave out a cell element corresponding to the location which has already been taken into use.

Nested tables

Tables can be nested, because a TD element (and a TH element) may contain a block element and therefore a table element in particular.

Nested tables easily become confusing. Moreover, there are browsers which cannot handle nested tables in general or which get confused with complicated nested tables. Of course, nested tables can be the natural way of expressing information, when it is logically an array of something which may in turn be an array.

Basically you just need to be *very careful* in writing HTML code for nested tables. No new elements or other features are needed, just a combination of those which have already been described. But due to deep nesting one easily makes mistakes, and the results can be really messy, and locating the error may take time.

The simplest case is probably a table with a single row consisting of two elements, each of which is a table. This might be used for presenting two similar tables in parallel for comparison. To proceed with our grammatical example, here is a table containing two tables, one for declination in singular and one for declination in plural:

Example nt.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<TITLE>tbl</TITLE>
<TABLE ALIGN=CENTER>
<CAPTION>Declination of <I>is</I></CAPTION>
<TR><TD>
<TABLE BORDER=1 ALIGN=CENTER CELLPADDING=3>
<CAPTION>Singular</CAPTION>
<TR><TH></TH><TH>neuter</TH><TH>masc.</TH><TH>fem.</TH></TR>
<TR><TH>nom.</TH><TD ROWSPAN=2 VALIGN=MIDDLE><I>id</I></TD>
 <TD><I>is</I></TD><TD><I>ea</I></TD></TR>
<TR><TH>acc.</TH><TD><I>eum</I></TD><TD><I>eam</I></TD></TR>
<TR><TH>gen.</TH><TD COLSPAN=3 ALIGN=CENTER><I>eius</I></TD></TR>
<TR><TH>dat.</TH><TD COLSPAN=3 ALIGN=CENTER><I>ei</I></TD></TR>
<TR><TH>abl.</TH><TD COLSPAN=2 ALIGN=CENTER><I>eo</I></TD>
 <TD><I>ea</I></TD></TR>
</TABLE>
</TD>
<TD>
<TABLE BORDER=1 ALIGN=CENTER CELLPADDING=3>
<CAPTION>Plural</CAPTION>
<TR><TH></TH><TH>neuter</TH><TH>masc.</TH><TH>fem.</TH></TR>
<TR><TH>nom.</TH><TD ROWSPAN=2 VALIGN=MIDDLE><I>ea</I></TD>
 <TD><I>ii (ei)</I></TD><TD><I>eae</I></TD></TR>
<TR><TH>acc.</TH><TD><I>eos</I></TD><TD><I>eas</I></TD></TR>
<TR><TH>gen.</TH><TD COLSPAN=2 ALIGN=CENTER><I>eorum</I></TD>
 <TD><I>earum</I></TD></TR>
<TR><TH>dat.</TH><TD COLSPAN=3 ROWSPAN=3 ALIGN=CENTER VALIGN=MIDDLE>
 <I>iis (eis)</I></TD></TR>
<TR><TH>abl.</TH></TR>
</TABLE>
</TD>
</TABLE>
```

Notice the explicit use of end tags like </TD>. The same code with omissible tags omitted is equivalent according to HTML 3.2 specification, but Netscape has a bug which can make it present a nested table incorrectly in the absence of end tags.

Alignment of cells

Alignment of cells, i.e. the positioning of the contents of a table cell (within the space reserved for the cell by a browser), is important in tables containing numerical data. You may wish to control it in other contexts as well.

The *default* alignment is the following:

- in *horizontal* direction,
  - *heading* cells (TH elements) are *centered*
  - normal *data* cells (TD elements) are aligned to the *left*
- in *vertical* direction, the contents is *centered* with respect to the middle of the cell.

There is no way to set different defaults for an entire table. (Although the TABLE element accepts an ALIGN attribute, it affects the positioning of the *entire* table!)

However, you can use the ALIGN and VALIGN attributes in TH and TD elements to set the alignments for an individual cell, and you can use the same attribute in a TR element to set the alignment defaults for the cells within that element (i.e. within one row); naturally, such defaults can be overridden in individual elements.

The possible values of ALIGN (in TH, TD and TR elements) are LEFT, RIGHT, and CENTER, for aligning the contents of a cell horizontally with respect to the left, center or right within the space for the cell. Notice that when aligning to the left or right, there can still be some space between the content of a cell and the left or right lower border of the cell, depending on the setting of the CELLPADDING attribute of the enclosing TABLE element.

The possible values of VALIGN (in TH, TD and TR elements) are TOP, MIDDLE, and BOTTOM, for aligning the contents of a cell vertically with respect to the top, center or bottom of the space for the cell. As stated above, the default is VALIGN=MIDDLE. Notice that when VALIGN=TOP or VALIGN=BOTTOM is used, there can still be some space between the content of a cell and the upper or lower border of the cell, depending on the setting of the CELLPADDING attribute of the enclosing TABLE element.

Fonts in table elements

People often ask how to designate font face, size and color for data within tables.

The short answer is: *Don't*. When necessary, use logical markup for text elements within tables as well as elsewhere. (Previous discussion contained a simple example of this.)

Assuming that you really need to designate font face, size and color (or just insist on doing so), the laborious way of doing it elementwise is the only portable way. Here *portable* means that you can, with some confidence, expect the HTML code to work on most browsers (assuming that they have table support at all, of course). This is not just a standards issue. In particular, in Netscape the BASEFONT element does not affect text in tables (it is disputable whether it should, according to the standard).

To summarize the situation, as regards to portable solutions in the above-mentioned sense:

font face
: Cannot be set in HTML 3.2 at all. You can only use a few markup elements to *suggest* that a font of a specific *kind* (e.g. italics, monospaced, bold) be used. These cannot be set globally, i.e. if you want them to apply to all elements of a table, they must appear separately in each TH or TD element. (The FACE attribute of the FONT element is not allowed in HTML 3.2, though mentioned as a feature which is supported by some browsers. It is valid, though deprecated as the entire FONT element is, in HTML 4.0. And it is "local", text level markup, so it really needs to be put into each table cell separately.)

font size
: Locally (e.g. within a table cell) you can use SMALL, BIG, or FONT SIZE=... You can set the global (default) font size with BASEFONT but this usually does not affect table cell contents, as explained above.

font color
: Locally (e.g. within a table cell) you can use FONT COLOR=... You can set the default text color globally - in the absolute sense, for the *entire* document - with BODY TEXT=... But you cannot

set the default color for a table to be different from that of the entire document.

Style sheets provide tools for affecting the rendering in a rather detailed manner, but support for them in browsers is still under development.

Style sheets

Style sheets are not part of HTML. They can be used even in conjunction with HTML 2.0 despite the fact that HTML 2.0 contains no specific constructs related to style sheets. On the other hand, HTML 3.2 contains such constructs, and assumably future versions of HTML will have more support.

The **basic idea** of style sheets is to provide tools for specifying features of the visible (or audible) **representation** of HTML documents without introducing new HTML tags and attributes for the purpose. The presentation style is specified in a manner which allows several style specifications (by the author and by users, as well as browser defaults) to be taken into account when rendering a document. This will allow control over indentation, colors, fonts, etc in a sophisticated manner. For more information about style sheets in general, consult the W3C pages on style sheets and WDG pages on style sheets. There is also a CSS FAQ by The HTML Writers Guild. For criticism of style sheets, see my *Why style sheets are harmful*.

Almost at the same time as the *HTML 3.2 Reference Specification* was accepted as a W3C Recommendation, a recommendation with similar status was accepted concerning style sheets: *Cascading Style Sheets, level 1*, abbreviated CSS1. The two recommendations are, however, separate in the sense that the combination of style sheet specifications with HTML documents has not been defined exactly. In particular, CSS1 mentions the ID and CLASS attributes for selecting specific pieces of text, but these attributes are not in HTML 3.2. The same applies to attributes of STYLE element and the proposed SPAN element.

The HTML 3.2 language provides two ways of referring to style sheets in HTML documents:

- one can use a LINK element with the REL=STYLESHEET attribute; the style sheet itself is in a separate file, and the LINK element specifies its name
- one can use a STYLE element; in this case, the style sheet itself can appear as the contents of the STYLE element or it can reside in a separate file.

In both cases you can e.g. define the visible representation of H1 elements in your documents but you cannot specify that some H1 elements are presented in some way and some other H1 elements (in the same document) in another manner. However, a browser which supports style sheets at all very likely supports some mechanisms (outside HTML 3.2) for the latter situation.

Additional methods of referring to style sheets in HTML will probably be possible, and some of them are already supported. For a short general discussion, see *Linking Style Sheets to HTML* by WDG. There is also a W3C Working Draft *HTML3 and Style Sheets* which discusses these issues.

An HTML 3.2 conforming browser need not support style sheets in any way (except by recognizing the STYLE element and hiding its contents). However, there is increasing support for some features of CSS1 in browsers.

Descriptions of HTML 3.2 tags

Index and legend

*A *ADDRESS *APPLET *AREA *B *BASE *BASEFONT *BIG *BLOCKQUOTE *BODY *BR *CAPTION *CENTER *CITE *CODE *DD *DFN *DIR *DIV *DL *DT *EM *FONT *FORM *H1 *H2 *H3 *H4 *H5 *H6 *HEAD *HR *HTML *I *IMG *INPUT *ISINDEX *KBD *LI *LINK *MAP *MENU *META *OL *OPTION *P *PARAM *PRE *SAMP *SCRIPT *SELECT *SMALL *STRIKE *STRONG *STYLE *SUB *SUP *TABLE *TD *TEXTAREA *TH *TITLE *TR *TT *U *UL *VAR

The structure of the element descriptions is as follows:

- A *heading*, containing the element name and a short description of its meaning, and, if needed, a warning that the element is not in HTML 2.0.
- A short description of the *purpose* of the element.
- A verbal description of a *typical rendering* by a (graphical) Web browser.
- A description of the *basic syntax* (without attributes, except obligatory or very common attributes).
- Possible *attributes* with their meanings and possible values, in the form of a table.
- The allowed *context*, i.e. a specification which says where the element may occur.
- The allowed *contents* of the element, i.e. the elements (or other constructs) which may occur between the start tag and the end tag. If the content is specified as being none, the element is a so-called *empty element* which neither requires nor allows an end tag or any contents.
- *Examples*, usually first a simple example showing the very basic and primitive use, with "everything defaulted", then a more complicated example (if possible), showing options etc. Most example HTML codes, displayed as a separate paragraph in monospaced font, are preceded by names like Example PRE-1.html which act as links to documents containing the code, allowing the reader check easily what the example looks like on his browser and environment. Notice that the renderings themselves are not included in this document; this is intentional, in order to make explicit the difference between an HTML structure and its visual appearance when using a particular browser.
- Pragmatic *notes* about the usage of the element. The ordering of these notes proceeds from questions like "should I use this element at all, or should I use some other instead" to various practical aspects of using it properly, then to more and more technical issues. The notes may include warnings about typical abuse or common errors.

This presentation does not discuss the XMP, LISTING, and PLAINTEXT elements. They have been obsolete for a long time, and PRE should be used instead.

A - anchors, hyperlinks, etc

Purpose

To set up hyperlinks and "anchors" for them, i.e.

- to define that a word or other construct in the document acts a link to a resource (e.g. another HTML file, or an image file, or an audio file), or

- to specify that the current location can be used, with a given name, as the target of such links (in the same or another document).

In principle, the A element can also be used for some other purposes which are currently of little practical value.

Typical rendering

An A element of the form <A HREF="*target*">*anchor text*</A> is displayed so that *anchor text* is presented in a distinguished manner (e.g. underlined or highlighted). The reader might even tell his browser not to present links in any distinguished manner, especially when the document is printed on paper. (On paper output, links might also be presented using footnotes.)

There are no automatic newlines or similar phenomena involved in presenting the anchor text; this means that the anchor text can be part of normal text flow in the document.

The user may *select* the anchor text (in a browser-dependent manner, using e.g. arrow keys for moving the cursor and enter key for selecting, or the mouse for moving the cursor and a mouse button click for selecting). In that case the document or location in a document as specified by the *target*, if existent and accessible, will be fetched and presented to the user. A browser may allow the user to select whether the document is to be displayed in the same or in another window on the screen.

The *visual look of anchor texts* is settable by user options in many browsers. It can depend on whether the target has been visited by the user or not. It is also affected by eventual LINK and VLINK attributes in a BODY element. When a document is *printed*, anchor texts might be, depending on the browser and its settings, e.g. normal text or underlined text or footnotes (indicating the target URLs) might be attached to them.

If *anchor text* is (or contains) an IMG element, a browser generally indicates the *image as a link* by drawing a colored (typically blue) border around the image. The width (and existence) of such a border can be controlled by the BORDER attribute of the IMG element. (Notice that if you suppress the border by using BORDER=0, you should separately indicate the image as a link, since otherwise the user won't know it's a link.)

Other A elements than those containing an HREF attribute have no effect on the rendering of a document. (Exception: a few browsers present the text within an A element with a NAME attribute in some highlighted manner.)

Basic syntax

<A HREF="*target*">*anchor text*</A>

or

<A NAME="*name*"></A>

Possible attributes

attribute name	possible values	meaning	notes
NAME	*string*	a name for a link end	must be unique within the document; case sensitive
HREF	*URL*	network address for the linked resource	could be another HTML document, a PDF file, an image etc
REL	*string*	the forward relationship also known as the "link type"; see notes on REL and REV values in the general description of links	in principle, could be used by browsers and other softwarein several ways, e.g. to determine to how to deal with the linked resource when printing out a collection of linked resources
REV	*string*	the reverse relationship:	a link from document *A* to document *B* with REV=*relation* expresses the same relationship as a link from *B* to *A* with REL=*relation*.
TITLE	*string*	a title for the linked resource	advisory; see remarks below

The value of a TITLE attribute might be used e.g.

- for display prior to accessing the destination resource, e.g. as margin note or on a small box, while the mouse is over the anchor or the document is being loaded; thus, the attribute can be used as a hint about the nature of the link
- as a window title for such resources that do not include a title, such as graphics or plain text
- as the subject of an E-mail message when an A element refers to a `mailto:` URL

(Not all browsers actually use the attribute in manners like those described above)

Allowed context

Text container, i.e. any element that may contain text elements, except an A element. This includes most HTML elements.

Contents

Text elements, except A elements. Notice that this includes IMG (you can have an image as the "anchor text") but excludes headings (you can have A elements within headings but not vice versa).

Examples

Example A.html:

```
<P>A hyperlink referring to a document in the same directory
as the current one:
<A HREF="ADDRESS.html">Examples of using ADDRESS tag</A>.
<P>A hyperlink referring to a document elsewhere:
<A HREF="http://www.hut.fi/english.html">HUT</A>.
<P>A hyperlink in which the link text contains markup:
<a href="http://www.iki.fi/oa/HTML/"><cite>The HTML test set</cite></a>
<p>A hyperlink referring to a label in the same document:
```

```
<A HREF="#final">final example</A>.
<P>A hyperlink referring to a label in another document:
<A HREF="http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimerP2.html#UR">
URL info in HTML Primer</A>
<P>A link to an image:
<A HREF="http://www.hut.fi/~jkorpela/perhe.jpg"
   TITLE="Yucca's family picture, by Minna">a family picture</A>.
<P><A NAME="final">Finally, this is just text to which you can
refer with a hyperlink.</A>
```

Notes

See the general discussion of links, which contains additional examples as well as notes on normal links links to audio and video resources, links to binary and plain text files, and links to other non-HTML resources (such as `mailto:` links).

Don't use anchor texts like *Click here*. They look stupid e.g. in a paper copy of a document. Warren Steel says in *Hints for Web Authors*:

> You don't need to say "Click **here** for information on our graduate programs;" just insert the link into what you were saying: "Our excellent **graduate programs** ..." Links to large files or unusual formats should be so marked, perhaps in a parenthetical note. "Our stirring **fight song** (400k .au) ..."

It is a rather common **error** to omit quotes or the closing quote in an HREF attribute. Some browsers are permissive, others may get very confused, so that the link may not work at all.

You *cannot nest* A elements, but you can write a dual-purpose A element which has both an HREF and a NAME attribute, e.g. `<A NAME="foo" HREF="#bar">zap</A>`

An A element with NAME attribute specifies a location which can be referred to by URLs with fragment identifiers. For example, `<A NAME="foo">` specifies a target which can be referred to by `<A HREF="#foo">` in the same document. Notice that other authors cannot refer to specific locations in your documents unless you include suitable anchors. Thus, it is often advisable to use an A NAME element at least for (technically, within) the major heading elements.

It is not obvious what exactly is the entity named in A NAME element. The most natural interpretation seems to be that it is a *part* of the document, namely the part between the start and end tags. However, notice that only text elements are allowed within the contents and that most browsers seem to interpret things so that an A NAME element just names a *location* (a point) in the document, namely the location of the start tag, leaving the position of the end tag meaningless. (However, an end tag </A> is obligatory!)

It is syntactically legal to have an A element with empty content, such as `<A NAME="foo"></A>`, but this has been observed to confuse some browsers. The simple solution is include a few words from the text into the A NAME element, e.g.

```
<P><A NAME="summary">To summarize</A>, it is legal but not advisable
to have an A element with empty content.</P>
```

If an A element contains an IMG element with the ISMAP attribute, they constitute a server-side image map.

ADDRESS - document author information

Purpose

To provide contact information about the author of the current document (i.e. the document in which the element is used).

Typical rendering

Typical rendering should involve paragraph breaks before and after. This is, however, not the case in Netscape, for example (see notes below). A browser may or may not use some special font like italics.

Basic syntax

<ADDRESS>*address information*</ADDRESS>

Possible attributes

None.

Allowed context

Block container.

Contents

Text elements and P elements.

In HTML 4.0 Strict, P elements are not allowed within an ADDRESS element.

Examples

Very simple address information, containing just the author's E-mail address:

Example ADDRESS-1.html:

```
<ADDRESS>
Jukka.Korpela@hut.fi
</ADDRESS>
```

One idea is to provide just the author's name but so that it is a link to a home page containing more information. This is typically suitable for short documents to be viewed on the screen only.

Example ADDRESS-2.html:

```
<ADDRESS>
<A HREF="http://www.hut.fi/u/jkorpela/">Jukka Korpela</A>
</ADDRESS>
```

A longer example which uses several ADDRESS elements, to specify different kinds of addresses (notice that unfortunately Netscape may not distinguish them visually from each other):

Example ADDRESS-3.html:

```
<ADDRESS>
Jukka Korpela, M.S. (Math.)<BR>
Helsinki University of Technology Computing Centre<BR>
FIN-02150 Espoo<BR>
Finland
</ADDRESS>
<ADDRESS>
Telephone International +358 9 451 4319
</ADDRESS>
<ADDRESS>
Electronic mail (Internet):
<A HREF="mailto:Jukka.Korpela@hut.fi">Jukka.Korpela@hut.fi</A><BR>
WWW home page:
<A HREF="http://www.hut.fi/u/jkorpela/">http://www.hut.fi/u/jkorpela/</A>
</ADDRESS>
```

Notes

Typically an ADDRESS element is *placed* either under the main heading of the document or at the end of the document (perhaps preceded by an HR element to separate the address information from the end of the document text).

NCSA Beginner's Guide to HTML says that the ADDRESS element "is not used for postal addresses", but the *HTML 2.0 specification* contains no such statement; on the contrary, its example of ADDRESS illustrates using it for a postal address. Notice that the ADDRESS element obeys the normal rules for division into lines; thus, if you want the components of a postal address appear on lines of their own, use the BR element for line breaks.

Several browsers, including Netscape, do not use normal paragraph breaks when rendering ADDRESS. You might therefore consider using explicit P tags within an ADDRESS element around the address information, although this does not conform to HTML 4.0 Strict. (Notice that in HTML 3.2, a P element is allowed within ADDRESS but not vice versa.)

It is advisable to obey applicable standards when writing address information. In particular, when providing telephone numbers, please apply CCITT recommendation E.123.

The ADDRESS element itself *creates no links*; to provide e.g. a link to author's home page or a `mailto` link to author's E-mail address, use the normal A element with HREF attribute (within the ADDRESS structure or outside it); see also: META element and LINK element with REV attribute.

Don't forget to add BR tags for line breaks.

APPLET - Java applets (Not in HTML 2.0!)

Purpose

To embed a Java applet into an HTML document.

Typical rendering

If the browser is Java enabled, it runs the applet; this typically means that some animation (perhaps an interactive one) is shown in area within the browser window. If not, it displays the contents (after PARAM elements) of the applet, or the string specified in the ALT attribute.

Basic syntax

<APPLET CODE="*appletfile*" WIDTH=*m* HEIGHT=*n*> *textual description* </APPLET>

Possible attributes

attribute name	possible values	meaning	notes
CODEBASE	*URL*	the base URL of the applet; this typically refers to the directory or folder containing the code of the applet	default is the URL of the document
CODE	*string*	class file, i.e. the name of the file that contains the compiled Applet subclass of the applet	obligatory; interpreted as relative to the base specified by the CODEBASE attribute; cannot be absolute
ALT	*string*	a textual description, to be displayed in place of applet	the contents of the element can be used for the same purpose, with more flexibility
NAME	*string*	a name for the applet instance	such names make it possible for applets in the same document to find (and communicate with) each other.
WIDTH	*integer*	suggested width, in pixels, not counting any windows or dialogs which the applet brings up	obligatory
HEIGHT	*integer*	suggested height, in pixels, not counting any windows or dialogs which the applet brings up	obligatory
ALIGN	TOP, MIDDLE, BOTTOM, LEFT, RIGHT	positioning of the applet display area	similar to ALIGN attribute of IMG
HSPACE	*integer*	suggested horizontal gutter (width of white space to the immediate left and right of the applet display area), in pixels	cf. to HSPACE attribute of IMG
VSPACE	*integer*	suggested vertical gutter (height of white space above and below the applet display area), in pixels	cf. to VSPACE attribute of IMG

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements.

Contents

Zero or more PARAM elements followed by zero or more text elements.

The exact meaning and intended use of text elements in the contents is somewhat obscure. The following is the wording of the *HTML 3.2 Reference Specification*:

> Following the PARAM elements, the content of APPLET elements should be used to provide an alternative to the applet for user agents that don't support Java. - - Java-compatible browsers ignore this extra HTML code. You can use it to show a snapshot of the applet running, with text explaining what the applet does. Other possibilities for this area are a link to a page that is more useful for the Java-ignorant browser, or text that taunts the user for not having a Java-compatible browser.

Notice that text elements in the contents and ALT attribute in the start tag are two ways of having something displayed in place of the applet. There are two differences: the value of ALT is a plain string, whereas the elements may contain text markup; and an ALT attribute has no effect if the browser does not know an APPLET element at all, whereas such a browser probably processes the text elements in the contents - it simply ignores the APPLET (and PARAM) start and end tags.

Examples

A simple example: an applet which draws animated bubbles.

Example APPLET-1.html:

```
<APPLET CODEBASE="http://java.sun.com/applets/other/Bubbles/classes"
   CODE="Bubbles.class" WIDTH=500 HEIGHT=500 ALIGN=MIDDLE>
<P><IMG SRC="bubbles.gif" ALT="[GIF image (2k)]"><BR>
An extract from a snapshot of <CITE>Bubbles</CITE> animation
by <A HREF="http://java.sun.com/">java.sun.com</A>.</P>
</APPLET>
```

A more complicated example, using parameter passing (PARAM element):

```
<APPLET CODE="AudioItem" WIDTH=15 HEIGHT=15 ALIGN=TOP>
<PARAM NAME=snd VALUE="Hello.au|Welcome.au">
<STRONG>Welcome!</STRONG>
</APPLET>
```

An example of typical (?) use of Java applets for "decorative" purposes (which many people find annoying):

```
<APPLET CODEBASE="applets/NervousText"
     CODE="NervousText.class"
     WIDTH=300
     HEIGHT=50>
<PARAM NAME=TEXT VALUE="Java is Cool!">
<IMG SRC="sorry.gif" ALT="This looks better with Java support">
</APPLET>
```

Our final example, as well as the first one, uses Java demo code from java.sun.com. It runs an applet for a nice game. Of course, the essential thing is the applet itself. The HTML constructs needed are simple and similar to the ones above. However, here the alternative texts just inform the user about Java being not in use. In cases like this, where the applet is *essential*, such notes might be the best you

can do, although you might consider providing a snapshot for illustration (as in the first example) to help the reader to decide whether he really wants to see the applet in action.

Example APPLET-4.html:

```
<H3>Multilingual Word Match Game</H3>
<P>This demonstration uses a Java applet
(from <A HREF="http://java.sun.com/">java.sun.com</A>),
so you need Java enabled on your browser to see it.</P>
<P>First select a language on the left. The match words with
pictures, clicking first on a picture, then on a word. Click
on "score" to see how well you did.</P>
<APPLET
  CODEBASE="."
  CODE="WordMatch.class"
  WIDTH=500 HEIGHT=350
  ALT="(Your browser recognizes the APPLET element but does
not run the applet.)">
<EM>Your browser either has no Java support at all or
it has Java support disabled.</EM>
</APPLET>
```

Notes

In HTML 4.0, the APPLET element is deprecated in favor of the new OBJECT element. However, the implementation of OBJECT is broken in many popular browsers.

Java is an object-oriented programming language (somewhat similar to C++) developed by Sun. A *Java applet* is a Java program which is executed by an interpreter invoked by (some) Web browsers upon encountering an APPLET element. Java applets can be used for animations, games, etc.

Even if a browser supports Java, the support can be disabled by system administration or by individual users, and people often do this because they think Java has too many security risks. Therefore, if you use Java applets, try to design your documents so that they work (although perhaps unimpressively) with Java disabled, too.

There is a very large number of Java applet examples on the Web. For some collections, see *FREE Java and JavaScript* at TheFreeSite.com. (But make sure you understand the difference between Java and JavaScript!)

AREA - area in a clickable map (Not in HTML 2.0!)

Purpose

To define an area ("hotzone") in a (client-side) clickable map.

Typical rendering

No direct visual effect, but when the user clicks in the specified area, the document mentioned in the AREA element is visited.

To help the user, a browser may display, in the status line, the contents of the ALT attribute as the mouse or other pointing device is moved over an area.

Basic syntax

<AREA HREF="*URL*" COORDS="*x1*,*y1*,*x2*,*y2*">

Possible attributes

attribute name	possible values	meaning	notes
SHAPE	RECT, CIRCLE, POLY	shape of the area	default is RECT
COORDS	*string* of a form which depends on SHAPE	coordinates for the area	obligatory except for defaulted SHAPE
HREF	*URL*	address of a document	acts as a hypertext link
NOHREF	NOHREF	means that this region has no action	useful when you want to cut a hole in a hotzone region
ALT	string	textual description of the area	obligatory

The meanings of SHAPE and the syntax and semantics of COORDS for each shape is the following:

SHAPE value	form of area	syntax of COORDS	meaning of COORDS
SHAPE=RECT	rectangle	COORDS="*x1*,*y1*,*x2*,*y2*"	the *x* and *y* coordinates of the upper left and lower right corner
SHAPE=CIRCLE	circle	COORDS="*x0*,*y0*,*r*"	the *x* and *y* coordinates of the center and length of the radius
SHAPE=POLY	polygon	COORDS="*x1*,*y1*,*x2*,*y2*,*x3*,*y3*,..."	the *x* and *y* coordinates of the vertices

The *x* and *y* coordinate values are measured in pixels from the *upper left* corner of the associated image. This means that the *y values increase downwards*.

Examples of various shapes:

`SHAPE=RECT COORDS="0,0,9,9"`	a rectangle of 10 by 10 pixels in the top left corner of the image
`SHAPE=CIRCLE COORDS="10,10,5"`	a circle with radius of 5 pixels and center at location (10,10)
`SHAPE=POLY COORDS="10,50,15,20,20,50"`	a polygon (in this case, a triangle) with edge locations (10,50), (15,20), and (20,50)

Allowed context

MAP element.

Contents

None.

Examples

```
<AREA HREF="guide.html" ALT="Guide" COORDS="0,0,118,28">
```

Notes

If two or more regions **overlap**, the region defined *first* in the map definition takes precedence over subsequent regions. Thus, to make part of an area defined by an AREA element inactive, put an AREA element with the NOHREF attribute before it.

A draft version of HTML 3.2 contained DEFAULT as a possible value of SHAPE, to be used to specify what happens if the user selects a point which does not belong to any area specified in other AREA elements. This was removed. The same effect can be achieved by using `AREA SHAPE=RECT COORDS="0,0,`*`width`*`,`*`height`*`"` as the *last* one within a MAP element. (Here *width* and *height* are the dimensions of the entire image in pixels.)

The HTML specifications allow **percentage** values for coordinates too, so that e.g. `COORDS="0,0,100%,100%"` could be used when specifying a rectangle which covers the entire image. However, many popular browsers incorrecly treat such values as pixels, i.e. ignore the `%` sign. Thus, *don't* use percentages. This isn't a serious restriction, since you (or the program you use) need to work with pixel values anyway when setting up a useful image map.

The ALT attribute is used to provide text labels which can be displayed in the status line as the mouse or other pointing device is moved over hotzones, or for constructing a textual menu for non-graphical browsers. Authors are strongly recommended to provide meaningful ALT attributes to support interoperability with speech-based or text-only user agents. But notice that the value must be just a string with no text markup.

B - bolding

Purpose

To present text in a boldface font.

Typical rendering

Bolded. See general notes on rendering markup.

Basic syntax

<B>*text*</B>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example B-1.html:

```
In the main menu of the program, one letter, usually the
initial, of each word is in bold face, e.g. <B>s</B>ave.
This indicates the letter that can be used as a shortcut.
```

Example B-2.html:

```
<P>In mathematics, vectors are usually denoted with boldface
letters like <B>x</B>.</P>
<P>When presenting computer source programs in printed form,
one often uses boldface for reserved words like <B>int</B> in C.</P>
```

Notes

Avoid using B; use logical markup instead. In particular, for **emphasis** use EM or STRONG.

See general notes on text markup, which provide additional examples.

BASE - base for URLs

Purpose

To define base URL for relative URLs in the document (e.g. in HREF attributes of A elements). This is typically used when mirroring documents.

For example, given
`<BASE href="http://foo.com/index.html">`
the IMG element
`<IMG SRC="images/bar.gif">`
refers to image
`http://foo.com/images/bar.gif`

Typical rendering

None. The BASE element has no direct effect on the rendering of a document.

Basic syntax

<BASE HREF="*URL*">

Possible attributes

attribute name	possible values	meaning	notes
HREF	*URL*	base URL to be used	obligatory; must be absolute

Allowed context

The head element, in which at most one BASE element may appear.

Contents

None.

Example

```
<BASE HREF="http://www.hut.fi/u/jkorpela/">
```

This implies that e.g. the link
`<A HREF="lists.html">list examples</A>`
is equivalent to
`<A HREF="http://www.hut.fi/u/jkorpela/lists.html">list examples</A>`

Notes

The BASE element is, with few exceptions, useful only for to make mirroring easier. Suppose there is a document which contains link tags like <A HREF="foo.html"> and suppose the document is copied to another server without the documents to which it refers that way. Then you can add a BASE element (referring to the original document) to the copy.

Since only one BASE element per document is allowed, you cannot have different base URLs in different parts of an HTML file.

In the absence of a BASE element in a document, the URL of the document itself is the base URL within it. (This is not necessarily the same as the URL used to request the document, since the base URL may be overridden by an HTTP header accompanying the document.)

It is advisable to *enclose the URL into quotes*, although this is not always mandatory.

Don't forget the slash "/". Anything that follows the last slash in the URL in a BASE element is interpreted as belonging to the filename part and ignored. The following is equivalent to the BASE element in the example above:
`<BASE HREF="http://www.hut.fi/u/jkorpela/foobar">`

whereas the following are equivalent to each other, so the meaning of the first one is probably not what was intended:

```
<BASE HREF="http://www.hut.fi/u/jkorpela">
<BASE HREF="http://www.hut.fi/u/">
```

BASEFONT - base font size (Not in HTML 2.0!)

Purpose

To specify the base font size (relatively to other sizes).

Typical rendering

BASEFONT sets the base (default) font size. The base font size applies to normal and preformatted text but not to headings, except where these are modified using the FONT element with a relative font size (e.g. FONT SIZE="+1").

It is not obvious whether it applies to tables. In Netscape, for example, BASEFONT does not affect the font size within tables. (Thus, to affect the font size within tables you must insert font changing elements into each cell!)

The actual font sizes used depend on the browser. See rendering notes about the FONT element.

Basic syntax

<BASEFONT SIZE=*n*>

Possible attributes

attribute name	possible values	meaning
SIZE	*string*	size of the font (1 - 7)

It is not obvious from the *HTML 3.2 Reference Specification* whether the SIZE attribute here follows the same rules as in the FONT element or has to be just an unsigned integer.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

None.

Examples

Example BASEFONT-1.html:

```
<P>This is text with default font size (3).</P>
<BASEFONT SIZE=5>
<P>This is text with font size 5 with <FONT SIZE=1>some text</FONT>
inserted with font size 1.</P>
```

Notes

Avoid using BASEFONT, for reasons explained in the discussion of text markup in general. In HTML 4.0, the BASEFONT element is deprecated in favor of style sheets.

Use FONT or, more preferably, SMALL or BIG to set font size locally (but notice that paragraph breaks are not allowed within FONT.)

BASEFONT can be regarded as a global counterpart for FONT with SIZE. In a sense, BODY with TEXT is a global counterpart for FONT with COLOR.

BIG - big font (Not in HTML 2.0!)

Purpose

To present text in a large font.

Typical rendering

Larger than normal font. See general notes on rendering markup.

Basic syntax

<BIG>*text*</BIG>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example BIG-1.html:

```
That was a <BIG>big</BIG> mistake!
```

Notes

Avoid using BIG; use logical markup instead. In particular, for **emphasis** use EM or STRONG.

See general notes on text markup, which provide additional examples.

It is unspecified what happens if BIG elements are nested; it might or might not result in using a font which is larger than you get with a single BIG.

The FONT element may provide more alternatives for specifying different font sizes.

BLOCKQUOTE - long quotation

Purpose

To present a (typically long) quotation to be rendered as a block of its own (in contrast to shorter quotations embedded into text paragraphs).

Typical rendering

As a separate paragraph (or sequence of paragraphs). Often indented (perhaps both at the left and at the right). Often in a font different from that of normal text, typically in italics.

Basic syntax

<BLOCKQUOTE>
quoted text
</BLOCKQUOTE>

Possible attributes

None.

Allowed context

Block container.

Contents

Headings, text elements, block elements, and ADDRESS elements.

In HTML 4.0 Strict, text elements may not occur directly within a BLOCKQUOTE element but must be enclosed into e.g. a P element.

Examples

Example BLOCKQUOTE.html:

```
<P>The original context of the saying <I>O tempora, o mores</I> is
the following:</P>
<BLOCKQUOTE>
<P>
O tempora, o mores!
Senatus haec intellegit. consul videt; hic tamen vivit.
Vivit? immo vero etiam in senatum venit, fit publici consilii particeps,
notat et designat oculis ad caedem unum quemque nostrum.
</P>
<P ALIGN=RIGHT>
<A HREF="http://www.utexas.edu/depts/classics/documents/Cic.html">
Cicero</A>,
<A HREF="http://www.utexas.edu/depts/classics/documents/cat1.html">
<CITE>Oratio in Catilinam Prima</CITE></A>, 2
</P>
</BLOCKQUOTE>
```

Notes

Basically, a *quotation* is an *exact* copy of somebody's words. (However, exactness does not normally imply using the same layout and fonts.) If you explain somebody's opinions or reports in your own words, it is not a quotation and should be presented as normal text (without any special markup).

Since BLOCKQUOTE is a block element, it is normally used for relatively long quotations. As regards to *short quotations* to be presented with no paragraph breaks around them, present them using text level markup. In special cases, you might use CODE, SAMP, KBD or CITE, but in the general case you have to resort to specifying the physical presentation, e.g. using italics (I element) or quotes according to your preferences and the norms of the language you use. (There is no generic text-level element for quotations in HTML 3.2, mainly because the rules for presenting such quotations are different in different languages.)

If it is essential to have the text displayed as it is written (with respect to division into lines and the use of blanks and tabs), consider using PRE.

When describing man-machine interaction, use the specific elements CODE, SAMP and KBD for quotations of program code, program output, and keyboard input.

Do **not** use BLOCKQUOTE to achieve *indentation*. A browser may or may not use indentation to present BLOCKQUOTE.

It belongs to proper manners to specify the **source** of quotation in some suitable way. In several cases this is even required by the law (copyright legislation). If possible, provide a hyperlink to the source document on the Web in addition to specifying the source in the text.

The BLOCKQUOTE element itself provides no structured way of presenting source information. The example above presents one method of doing so.

If you do not like the font used by browsers for BLOCKQUOTE, there is not very much to be done; however, style sheets may change this. If you wish to enforce e.g. italics font to be used (if possible), using the I element, remember that as a text element it does not allow e.g. paragraph breaks (or a BLOCKQUOTE) within it, so you must use a separate I element within each paragraph (P element).

As an *exception* to quotations being *exact* reproductions of the quoted text, you may leave out words which are irrelevant in the context of the quotation even if they appear in the middle of the quoted text; in such cases you should indicate the omission clearly (the notations - - and ... are the most common ways of doing this). Be very careful in such omissions; it is easy, but quite inappropriate, to quote someone selectively so that he seems to say something very different from what he really said - perhaps even just the opposite. As another exception, when necessary you may add clarifying words but only to convey the original meaning appropriately, not to change it to conform to your own thoughts. Typically, you add the correlate of a pronoun like *it*. You should clearly indicate such clarifications as not being part of the original; the most common way to do this is to put them into square brackets.

BODY - document body

Purpose

The basic structure of an HTML document always consists of a head and a body. It is not necessary to explicitly enclose the body into a BODY element, but by doing so one can specify attributes which affect the document as a whole (e.g. by setting background image or color).

Typical rendering

Using an explicit BODY element does not affect the document rendering, unless the element contains attributes.

Basic syntax

<BODY>*document body*</BODY>

Possible attributes (Not in HTML 2.0!)

attribute name	possible values	meaning
BGCOLOR	*color specification*	background color for the document
TEXT	*color specification*	color for the text of the document
LINK	*color specification*	color for *unvisited* hypertext links
VLINK	*color specification*	color for *visited* hypertext links
ALINK	*color specification*	color for *active* hypertext links; used to stroke the text for a link at the moment the user selects (e.g. clicks on) the link
BACKGROUND	*URL*	URL for an image to be used to tile the background.

All of these attributes are deprecated in HTML 4.0

Allowed context

The HTML element, which can be either implicit or explicit. Only one BODY element is allowed in a document, and it must appear after the document head (which can be implicit or explicit).

Contents

Headings, text elements, block elements, and ADDRESS elements.

In HTML 4.0 Strict, text elements may not occur directly within a BODY element but must be enclosed into e.g. a P element.

Examples

Example BODY-1.html:

```
<BODY>
<H1>Sample document</H1>
<P>
This is just a trivial sample document. Its body contains first
a heading, then a paragraph, and nothing else.
</P>
</BODY>
```

Example BODY-2.html:

```
<BODY
BGCOLOR=AQUA
TEXT="#848484"
LINK=RED
VLINK=PURPLE
ALINK=GREEN
>
<H1>Sample document</H1>
<P>
This is also a trivial sample document. Its body contains first
a heading, then a paragraph, and then a paragraph containing a link.
However, the BODY element uses attributes to affect the
visual rendering.
</P>
<P>
This document was written by
<A HREF="http://www.hut.fi/u/jkorpela/">Jukka Korpela</A>.
</P>
</BODY>
```

Example BODY-3.html:

```
<BODY
TEXT=BLUE
LINK=RED
VLINK=BLUE
ALINK=PINK
BGCOLOR=WHITE
BACKGROUND="wave.gif"
```

```
>
<H1>Sample document</H1>
<P>
This document contains first
a heading, then a paragraph, and then a paragraph containing a link.
However, the BODY element uses attributes to affect the
visual rendering, including a background image.
</P>
<P>
This document was written by
<A HREF="http://www.hut.fi/u/jkorpela/">Jukka Korpela</A>.
</P>
</BODY>
```

Notes

Only one BODY element is allowed in a document.

Be careful when playing with background images and colors. What looks cool on your screen might be disgusting on some other (or in someone else's opinion).

If you set some of the attributes BGCOLOR, TEXT, LINK, VLINK and ALINK, set them all. Otherwise e.g. your specified background color might coincide with user's default color for text. (See discussion of background in the *Frequently Encountered Problems* document by WDG.

Select the text color so that it works together with the background color or the colors of the background image. For instance, red on green can cause serious problems, because a significant number of people have difficulties in distinguishing them.

The text color can be affected locally by FONT elements with COLOR attribute. Background color cannot be set locally in HTML 3.2; if you want to use different backgrounds, you have to write separate HTML files (or use style sheets).

You can set both BGCOLOR and BACKGROUND. If you do, browsers typically give preference to BACKGROUND, but if the background image cannot be loaded, BGCOLOR is used.

For more information about backgrounds, see *The background FAQ* by Mark Koenen.

BR - line break

Purpose

To force a line break.

Typical rendering

A line break (but not paragraph break).

Basic syntax

Possible attributes (Not in HTML 2.0!)

attribute name	possible values	meaning	notes
CLEAR	LEFT, RIGHT, ALL, NONE	control of text flow	default is NONE; this attribute is deprecated in HTML 4.0

The attribute can be used to move down past floating images on either margin. <BR CLEAR=LEFT> moves down past floating images on the left margin, <BR CLEAR=RIGHT> does the same for floating images on the right margin, while <BR CLEAR=ALL> does the same for such images on both left and right margins.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements.

Contents

None.

Examples

A rather typical example where BR is used for getting some text into a line of its own:

Example BR-1.html:

```
<P>
You should always end the terminal session with the command
<BR>
<KBD>logout</KBD>
<BR>
or some other operation with the same effect.
</P>
```

Notes

Typical uses of BR:

- To simulate subparagraphs as in the example above (see also the description of the P element).
- To present poetic text where the division into lines (verses) is essential. See example 2 in the the description of the DIV element).
- To affect the positioning of images. BR elements with CLEAR attribute are often needed when embedded images are used; see the description of the IMG element.

You should **not** use BR elements just in order to force the text to be of some particular width which you regard as suitable. It is much better to let browsers do such formatting according to each browsing situation.

See notes on division into lines and the use of blanks and tabs.

Some people use multiple BR elements to force vertical white space. This need not work in all browsers. If you wish to force empty vertical space, consider using a suitable PRE element.

CAPTION - caption for a table (Not in HTML 2.0!)

Purpose

To present a caption (title) for a table.

Typical rendering

Above or under the table itself, often but not necessarily using some special, more prominent font.

Usually the caption is horizontally centered. (HTML 3.2 provides no tool for changing the browser behavior in this respect.)

Basic syntax

<CAPTION>*text*</CAPTION>

Possible attributes

attribute name	possible values	meaning	notes
ALIGN	TOP, BOTTOM	placement of the caption relative to the table	usually the default is TOP

The use of this attribute is deprecated in HTML 4.0.

Allowed context

TABLE element. If present, the CAPTION element must appear first, before the TR elements.

Contents

Text elements.

Examples

```
<CAPTION>Summary of measurement results</CAPTION>
<CAPTION><EM>Mean temperatures</EM></CAPTION>
```

Notes

You should normally include a caption into each table. The caption text should be relatively short, yet informative. Avoid inserting explanations into a caption. Give the explanations within normal text paragraphs. A caption should tell *what* the table is about. In normal text should tell *why* the table is presented, i.e. how the table relates to the text of the document.

See the discussion of tables, which contains additional examples, too.

Some browsers (e.g. Netscape) do not render the caption in a visually distinctive manner. Using phrase markup such as EM or STRONG within the CAPTION element may therefore be desirable.

CENTER - centering (Not in HTML 2.0!)

Purpose

To specify that part of a document to be centered in the rendering.

Typical rendering

Centered.

Basic syntax

<CENTER>
a section of the document
</CENTER>

Possible attributes

None.

Allowed context

Block container.

Contents

Headings, text elements, block elements, and ADDRESS elements.

Examples

Example CENTER.html:

```
<P>
This is a normal paragraph which will be rendered according to
default alignments, which usually means left alignment.
</P>
<CENTER>
<P>
This is text which will be centered.
</P>
```

```
<P>
This is a longer text paragraph which will be centered.
It is so long that line breaks will most probably occur.
Notice that the division into lines is usually not the same
as in the HTML file.
</P>
</CENTER>
```

Notes

In HTML 4.0, the CENTER element is deprecated in favor of style sheets.

Using ALIGN attribute in P and heading elements is preferable to using DIV.

CENTER is defined as equivalent to DIV with ALIGN=CENTER. CENTER was introduced by Netscape before they added support for the DIV element. It is retained in HTML 3.2 on account of its widespread deployment.

Since CENTER is a block element, it terminates an open P element (i.e. causes the browser to assume an implied </P> tag when necessary). Other than this, browsers are **not** expected to render paragraph breaks before and after CENTER elements. If paragraph breaks are desired, you can use the P element with an ALIGN attribute instead.

CITE - citations

Purpose

To present a citation or reference to other sources, such as a book title. See notes below.

Typical rendering

In italics. When such rendering is impossible, a browser might use underlining (Lynx does so) or quotes around the citation. See general notes on rendering markup.

Basic syntax

<CITE>*text*</CITE>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

A simple example, referring to a book by its title:

Example CITE-1.html:

```
I learned this from <CITE>The Origin of Species</CITE>.
```

Notes

On the basic nature of CITE: There are different opinions and practices on whether CITE is to be used for such citations as titles of books only or for quoting sentences or words in general. The official documents are laconic: for example, *HTML 3.2 Reference Specification* says that CITE is "used for citations or references to other sources". Typically dictionaries say that *citation* is roughly synonymous with *quotation*. However, the intended interpretation seems to be that CITE is for the *names* of external sources (books, articles, documents etc), not for actual extracts (quotations) from them.

Accepting this, the question arises how quotations are to be presented within text. (For quotations to be presented as separate paragraphs, or even sequences of paragraphs, BLOCKQUOTE is the natural choice.) You can either use quotation marks according to the rules of the language in which your own document is written, or some other suitable method, such as italics, i.e. the I element. The latter is often suitable for very short (e.g. single-word) quotations.

CODE - program code

Purpose

To present program code.

Typical rendering

Monospaced. See general notes on rendering markup.

Basic syntax

<CODE>*text*</CODE>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

The following example discusses the C programming language, referring to a particular expression in that language:

Example CODE-1.html:

```
Expressions like <CODE>a[i++] + b[i++]</CODE> should not be used,
since they cause undefined behavior.
```

Notes

As usual in HTML, division into lines and the use of blanks and tabs is selected by the browser, not honoring the one in the HTML file. Thus, large program codes are more suitably presented using the PRE element or as separate text files to which you have links in HTML files.

See also notes on presenting interaction with computer and general remarks on phrase elements.

DD - definition data

Purpose

To provide a definition for a term in a definition list (DL element)

Typical rendering

Indented and presented as a separate piece of text attached to the corresponding definition term.

Basic syntax

<DD>*definition*</DD>

The end tag </DD> can always be omitted, and it usually is omitted.

Possible attributes

None.

Allowed context

DL element.

Contents

Block elements. Notice that heading and ADDRESS elements are not allowed. On the other hand, lists are allowed.

Examples

An example which does not say very much:

```
<DD>See RFC 822.</DD>
```

For more realistic examples, see the description of the DL element.

Notes

Some people use DD as such, outside any DL element, to get some text indented. This violates the specifications and does not work in general.

DFN - defining occurrence (Not in HTML 2.0!)

Purpose

To indicate that a term (or phrase) appears in a context where it is defined.

Typical rendering

Obviously the element should we presented with some kind of distinction from normal text, such as italic or bold italic (as the HTML 2.0 specification suggests). Unfortunately many browsers, including Netscape, do not effectively support it: they present DFN as normal text.

See also general notes on rendering markup.

Basic syntax

<DFN>*text*</DFN>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example DFN-1.html:

```
<DFN>Ichthyology</DFN> is the branch of natural science which
studies fish.
```

Notes

Since current implementations do not effectively support DFN, as explained above, it is probably best to present defining occurrences using either EM or STRONG.

The HTML 2.0 specification does not include DFN but mentions it as an element which "has been deployed to some extent".

See also general remarks on phrase elements.

DIR - unnumbered list in directory-like form

Purpose

To present information in a directory-like format. The HTML 2.0 specification says that DIR represents a list of short items, typically up to 20 characters each.

Typical rendering

In practise, most browsers present a DIR element exactly the same way as an UL element. A few browsers omit the bullets, however.

Theoretically, the recommendation has been and still is that DIR element be rendered as a multicolumn directory list.

Basic syntax

<DIR>
<LI> *list item 1*
<LI> *list item 2*
...
</DIR>

Possible attributes

attribute name	possible values	meaning
COMPACT	COMPACT	reduced interim spacing

Typically, browsers ignore the COMPACT attribute.

Allowed context

Block container.

Contents

LI elements which do not contain block elements.

Examples

A very small list:

Example DIR-1.html:

```
<DIR>
<LI>one
<LI>two
<LI>three
</DIR>
```

A larger list of very small elements (typically this is **not** rendered in a suitable manner):

Example DIR-2.html:

```
<DIR>
<LI>A<LI>B<LI>C<LI>D<LI>E<LI>F<LI>G<LI>H<LI>I<LI>J<LI>K<LI>L<LI>M
<LI>N<LI>O<LI>P<LI>Q<LI>R<LI>S<LI>T<LI>U<LI>V<LI>W<LI>X<LI>Y<LI>Z
</DIR>
```

See also *Examples of various list elements in HTML.*

Notes

In HTML 4.0, the DIR element is deprecated in favor of the UL element; style sheets can be used to suggest features of the presentation of a list.

See general notes about list elements for a discussion of selecting between them.

DIV - document division (Not in HTML 2.0!)

Purpose

To specify document division. The ALIGN attribute allows different alignments (left, center, right) to be used in different parts of the document.

The DIV element can also be used in conjunction with style sheets in order to affect the rendering of parts of a document in various ways.

Typical rendering

The part of document is aligned according to the ALIGN attribute of the element.

Basic syntax

<DIV ALIGN=*alignment*>
a section of the document
</DIV>

Possible attributes

attribute name	possible values	meaning	notes
ALIGN	LEFT, CENTER, RIGHT	alignment of text within the element	deprecated in HTML 4.0

The ALIGN attribute specifies the *default* alignment; it can be overridden by ALIGN attributes in enclosed elements (e.g. P elements).

Allowed context

Block container.

Contents

Headings, text elements, block elements, and ADDRESS elements.

Examples

Example DIV-1.html:

```
<P>
This is a normal paragraph which will be rendered according to
default alignments, which usually means left alignment.
</P>
<DIV ALIGN=CENTER>
<P>
This is text which will be centered.
</P>
<P>
This is a longer text paragraph which will be centered.
It is so long that line breaks will most probably occur.
Notice that the division into lines is usually not the same
as in the HTML file.
</P>
</DIV>
```

The following example shows how to present (poetic) text as centered and with a particular division into lines:

Example DIV-2.html:

```
<DIV ALIGN=CENTER>
Mieleni minun tekevi<BR>
aivoni ajattelevi<BR>
lähteäni laulamahan<BR>
saa'ani sanelemahan.<BR>
<P ALIGN=RIGHT><CITE>Kalevala</CITE></P>
</DIV>
```

Notes

In HTML 4.0, the ALIGN attribute is deprecated .

Since DIV is a block-like element, it terminates an open P element (i.e. causes the browser to assume an implied </P> tag when necessary). Other than this, browsers are **not** expected to render paragraph breaks before and after DIV elements. If paragraph breaks are desired, you can use the P element with an ALIGN attribute instead.

DL - definition list

Purpose

To present a list of definitions for terms.

Typical rendering

A list where the terms are distinguished by means of layout or font usage or both. The rendering should support the association of each definition with the corresponding term. Typically the term is flush left while the definition is somewhat indented, but without bullets of any kind.

Basic syntax

<DL>
<DT>*term 1*<DD>*definition of term 1*
<DT>*term 2*<DD>*definition of term 2*
...
</DL>

Possible attributes

attribute name	possible values	meaning
COMPACT	COMPACT	more compact style of rendering

In practice, browsers often ignore the COMPACT attribute or implement it deficiently. The attribute is deprecated in HTML 4.0.

Allowed context

Block container.

Contents

DT and DD elements.

Normally you have pairs of DT and DD elements, specifying a term and its definition, of course. Multiple DT elements may be paired with a single DD element; this means that several terms share the same definition.

According to the HTML 2.0 specification, a document should not contain multiple consecutive DD elements, although this is not enforced in the formal syntax. On the other hand, there is no such statement in the HTML 3.2 specification, and it has been argued that a term might well have several (alternative) definitions.

Examples

Example DL.html:

```
<DL>
  <DT>Recursion, indirect
  <DD>See <I>indirect recursion</I>.
  <DT>Indirect recursion
  <DD>See <I>recursion, indirect</I>.
</DL>
```

Example DL-2.html:

```
<dl>
  <dt><span class="dt">term</span></dt>
  <dd>a word or expression that has a precise meaning
  in some uses or is peculiar to a science, art,
  profession, or subject (source:
  <cite><a href="http://www.m-w.com/">WWWebster</a></cite>)</dd>
</dl>
```

See also: *Examples of various list elements in HTML.*

Notes

Although DL is basically for *definitions*, it is often used for *descriptions* as well.

Browsers typically present a DL element in a form which is not suitable for presenting lists of *short* definitions, even if you use the COMPACT attribute.

You can use a TABLE element instead of a DL element (but remember that not all browsers support tables). See general notes about list elements.

DT - definition term

Purpose

To present a term in a definition list (DL element).

Typical rendering

Distinguished from normal text by means of layout or font usage or both.

Basic syntax

<DT>*term*</DT>

The end tag </DT> can always be omitted, and it usually is omitted.

Possible attributes

None.

Allowed context

DL element.

Contents

Text elements.

Examples

An example which does not say very much:

```
<DT>Terminus technicus.</DT>
```

For more realistic examples, see the description of the DL element.

EM - emphasis

Purpose

To emphasize.

Typical rendering

In italics. If this is impossible, a browser might use e.g. underlining (Lynx does so). See general notes on rendering markup.

Basic syntax

<EM>*text*</EM>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example EM-1.html:

```
The EM element is <EM>logical</EM> markup as opposite to
<EM>physical</EM> markup such as the I element.
```

Notes

Avoid emphasizing too much; emphasizing everything is tantamount no not emphasizing anything.

You can use STRONG for stronger emphasis.

See also general remarks on phrase elements.

FONT - font size and color (Not in HTML 2.0!)

Purpose

To specify font size (relatively to other sizes) or font color or both.

Typical rendering

The actual font size and color used to present the contents of the FONT element may be affected, but it depends on the browser; see general notes on rendering markup.

A browser may provide a user option for defining which font is to be used and which physical font size shall be used to correspond to the default font size (3) in HTML. Setting the font size in HTML may decrease or increase the actual font size used, in a browser dependent manner.

Basic syntax

<FONT SIZE=*n*>*text*</FONT>

or

<FONT COLOR=*colorspec*>*text*</FONT>

Possible attributes

attribute name	possible values	meaning	notes
SIZE	*string*	size of the font, either a number in the range 1 - 7 or a signed integer like `"+1"` or `"-2"`	signed value is added to the current base font size as set by BASEFONT to produce a size number in the range 1 - 7
COLOR	*color specification*	color to be used for the contents	might clash with background color!

Several browsers also support a FACE attribute which accepts a comma separated list of font names in order of preference. This is used to search for an installed font with the corresponding name.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements.

Examples

Example FONT-1.html:

```
This is some text <FONT SIZE=-1>including text which may appear
in a smaller font</FONT>.
<P>
This is an attempt to present one
<B><U><FONT SIZE=7 COLOR=RED>word</FONT></U></B>
very prominently: in bold face, underlined, in the largest font
available, and in red.
```

Notes

Avoid using FONT, for reasons explained in the discussion of text markup in general. In HTML 4.0, the FONT element is deprecated in favor of style sheets. (As regards to criticism of FONT in particular, see Warren Steel's *What's wrong with the <FONT> element?*.) Specifically, if you need to change font sizes, try to live with SMALL and BIG only.

Use BASEFONT to set font size for a large part of the document. (Notice that paragraph breaks are not allowed within FONT.)

The attributes in the BODY tag can be used to set the *background* color or the default text font color or both. Of course you should not use the background color for text!

A browser need not implement FONT so that SIZE values 1 - 7 all correspond to different font sizes. Some versions of Internet Explorer and Netscape have mapped those values to physical sizes e.g. so that sizes 4 and 5, or sizes 2 and 3, are equal to each other. The trend is that browsers nowadays try to map them to all to different sizes. On the other hand, browsers generally have an option like "ignore font sizes specified in documents", which is useful e.g. against pages which cluelessly set the size of *all* text to 2 or 4. In character cell browsers such as Lynx, a SIZE attribute has no effect, of course.

You may wish to use a separate file for checking the visual appearance of the different markup elements on your browser to see how it displays different font sizes. Consult information about color specifications for color samples, or a separate file containing text in 16 colors corresponding to the predefined color names.

There are two kinds of *relativity* involved in *font sizes*. First, in HTML we refer to font sizes with numbers in the range 1 - 7 which are in some browser and device dependent manner mapped to physical sizes (expressed e.g. in pixels, points or millimeters). The mapping is usually not linear; you should *not* assume that e.g. font size 3 is half of font size 6. Second, the way in which the font size (in the HTML meaning) is specified in the SIZE attribute can be relative; for instance, SIZE="+1" (which is quite different from SIZE="1" or SIZE=1) means the current base font size plus one, and the sum itself is relative in the sense explained above.

FORM - fill-out form

Purpose

To present a fill-out form to be used for user actions such as registration, ordering, or queries. Forms can contain a wide range of HTML markup including several kinds of form fields such as single and multi-line text fields, radio button groups, checkboxes, and menus. Usually forms are processed by CGI scripts.

Typical rendering

Something that more or less resembles a fill-out form on paper.

Basic syntax

<FORM ACTION="*URL*">
contents of the form, including INPUT elements and possibly TEXTAREA and SELECT elements
</FORM>

Possible attributes

attribute name	possible values	meaning	notes
ACTION	*URL*	address of the server-side form handler	an HTTP server (typically, a CGI script) or a `mailto:` URL (which is not supported by all browsers)
METHOD	GET, POST	HTTP method to be used to send the contents of the form to the server	default is GET, but POST is more common
ENCTYPE	*string*	media type used to encode the contents of the form	default is `application/x-www-form-urlencoded`

Allowed context

Block container.

Contents

Anything that is allowed within a document body (i.e. headings, text elements, block elements, and ADDRESS elements), with the exception that no FORM element is allowed within a FORM element.

In HTML 4.0 Strict, text elements may not occur directly within a FORM element but must be enclosed into e.g. a P element.

Notice in particular that there are some elements which may only appear within a FORM element. They can be used for various purposes as follows:

INPUT
: single line text fields, password fields, checkboxes, radio buttons, submit and reset buttons, hidden fields, file upload, image buttons, etc

SELECT
: single or multiple choice menus

TEXTAREA
: multi-line text fields.

Notice that you can enclose these form field elements into any element which allows a text element, provided that they are ultimately within some FORM element. You can, for example, have a FORM element which contains a TABLE which has cells containing form field elements.

Examples

First a trivial example. This is hardly better than a simple `mailto:` link (using A element), but it hopefully illustrates the structure of form specifications in a very simple case.

Example FORM-1.html:

```
Tell me what you think about my document:

<FORM ACTION="http://www.hut.fi/cgi-bin/mailto?Jukka.Korpela@hut.fi"
 METHOD=POST>
<P><TEXTAREA ROWS=5 COLS=72 NAME=Comments></TEXTAREA></P>
<P><INPUT TYPE=SUBMIT VALUE=Send></P>
</FORM>
```

The example above, as well as the the two other examples below, uses a simple CGI script named `mailto` (not to be mixed up with `mailto` URLs!) and accessible using URL of the form `http://www.hut.fi/cgi-bin/mailto?`*addr* where *addr* is an E-mail address. This particular CGI script has been coded to send the contents of the form as an E-mail message containing name-value pairs in a format which is both legible by humans and easy to process automatically. You can *test* these forms if you like, but please notice that they really send your message to the author; and please *do not copy* the ACTION attribute into a form of your own, since the service referred to is not intended to be a public service. (There are such public forms services elsewhere.)

The following more complicated example contains, in addition to an area for free text input, a selection menu. This might be a good way of getting evaluations, since for many people it is easier to fill a simple form than to write free comments.

Example FORM-2.html:

```
<FORM ACTION="http://www.hut.fi/cgi-bin/mailto?Jukka.Korpela@hut.fi"
 METHOD=POST>
<P>Please tell your opinion about the overall quality of
this document:
<SELECT NAME=evaluation>
<OPTION SELECTED>No opinion
<OPTION>Very poor
<OPTION>Rather poor
<OPTION>Average
<OPTION>Rather good
<OPTION>Very good
</SELECT>
</P><P>
You can also be more specific by writing a few comments:
<TEXTAREA NAME=Comments ROWS=5 COLS=72></TEXTAREA>
</P><P>
<INPUT TYPE=SUBMIT VALUE=Send></P>
</FORM>
```

The following example is more realistic, containing several fields of different kinds:

Example FORM-3.html:

```
This is a form for sending your personal evaluation of the document
<CITE>Learning HTML by Examples</CITE> as a whole.
<FORM ACTION="http://www.hut.fi/cgi-bin/mailto?Jukka.Korpela@hut.fi"
 METHOD="POST">
<P>
Your home page URL (if any):
<INPUT TYPE=TEXT SIZE=30 NAME=Home VALUE="http://">
</P><P>
Please rate the overall <EM>usefulness</EM> of the document (to you):<BR>
<INPUT TYPE=RADIO NAME=Useful VALUE="No opinion" CHECKED>No opinion<BR>
```

```
<INPUT TYPE=RADIO NAME=Useful VALUE="Very little">Very little (or none)<BR>
<INPUT TYPE=RADIO NAME=Useful VALUE="Little">Little<BR>
<INPUT TYPE=RADIO NAME=Useful VALUE="Some">Some<BR>
<INPUT TYPE=RADIO NAME=Useful VALUE="Great">Great<BR>
<INPUT TYPE=RADIO NAME=Useful VALUE="Very great">Very great
</P><P>
What about general <EM>understandability</EM>?
<SELECT NAME=Understandability>
<OPTION VALUE=undef SELECTED>(No opinion)
<OPTION VALUE=verydifficult>Very difficult
<OPTION VALUE=difficult>Difficult
<OPTION VALUE=avg>Average
<OPTION VALUE=easy>Easy
<OPTION VALUE=veryeasy>Very easy
</SELECT>
</P><P>Please feel free to add any comments you like:<BR>
<TEXTAREA ROWS=5 COLS=72 NAME=Comments></TEXTAREA>
<INPUT TYPE=HIDDEN NAME="Via" VALUE="FORM-3">
</P><P>
<INPUT TYPE=CHECKBOX NAME="*** Response requested! ***">
Would appreciate a personal answer; E-mail address:
<INPUT TYPE=TEXT SIZE=25 NAME=From>
</P>
<P>When you are finished with filling the form, select this:
<INPUT TYPE=SUBMIT VALUE=Send></P>
</FORM>
<P>You should get a response saying that a message was sent to
Jukka.Korpela@hut.fi. If you want to get back to the page
from which you came to this form, please use the "Back"
function of your browser twice.</P>
```

Notice the use of a HIDDEN field named `Via`. It is invisible to users filling the form but allows the recipient of the E-mail message to recognize the origin (form) from which the message was generated.

The next example has a very different theme. It illustrates how one can easily create a *customized interface to a search engine*, in this case AltaVista.

Example FORM-4.html:

```
<P>You can search for aquarium-related documents on the Web
using search engines like
<A HREF="http://www.altavista.com/">AltaVista</A>.
The following simple form gives you a simple interface for that;
just append, after the second + sign, a keyword and submit the form:</P>
<FORM method=GET  action="http://www.altavista.com/cgi-bin/query">
<INPUT TYPE=hidden NAME=pg VALUE=q>
Search the Web
for
<INPUT NAME=q size=45 maxlength=200 VALUE="+aquar* +">
<INPUT TYPE=SUBMIT VALUE=Submit><BR>
<INPUT TYPE=RESET VALUE="Reset the form">
</FORM>
```

Notice that in a case like this, with a text input field prefilled with useful content, a RESET button can actually be useful.

Notes

The *Intermediate HTML* tutorial contains an excellent presentation of forms. See also *Carlos' FORMS Tutorial*, which has some nice interactive features, and my document which provides additional annotated links to tutorials, references, and specialized documents about HTML forms.

The value of the **METHOD** attribute specifies the HTTP method (as defined in the HTTP specification) to be used to send the contents of the form to the server (when the ACTION attribute specifies an HTTP server). In principle, the GET method is to be used when the processing of the form is "idempotent", i.e. has no lasting observable effect on the state of the world; this applies typically by *query* forms, the processing of which may involve extracting information from a database but no changes to the content of the database. If the processing of the form has side effects (such as modification of a database or subscription to a service), the method should be POST. In practice, the POST method is often used even if the form submission has no side effects. As regards to technicalities,

- For the GET method, a browser starts with the action URL (the value of the ACTION attribute or, by default, the base URL of the document), then appends a ? character and the form data set, in the format specified by ENCTYPE, or in the `application/x-www-form-urlencoded` format by default. Finally the browser processes the resulting URL as if it were a link anchor.
- For the POST method, the browser conducts an HTTP POST transaction using the action URL and a message body in the in the format specified by ENCTYPE, or in the `application/x-www-form-urlencoded` format by default.

In general, you need a **CGI script** in order to use HTML forms.

Writing CGI scripts is probably not difficult to learn to anyone who has written computer programs before. However, many HTML authors do not know (and need not know) about programming, and learning a suitable scripting language takes time. Moreover, Web server maintainers may have strict policies on CGI scripts for security reasons. Thus, please contact your *local Web server documentation* or local webmaster for information about CGI scripts made available at your site, read their documentation, and write your forms so that you take into account the requirements of the script you have chosen to use.

If you decide to write your own CGI scripts or install scripts written elsewhere, there is plenty of material on the Web, for instance:

- The tutorial *How the web works: HTTP and CGI explained* by Lars Marius Garshol
- *Introduction to the Common Gateway Interface (CGI)*
- *CGI Programming FAQ*
- Information about CGI at Yahoo
- The Usenet newsgroup comp.infosystems.www.authoring.cgi

If the possibilities mentioned above are not feasible in your situation, you may wish to consider *using a CGI script on a remote server*. There are some services which allow you to use CGI scripts on their site, usually for some fee, but there are also free services.

Although the HTML 3.2 specification allows the ACTION attribute to refer to a `mailto:` URL, providing an easy way of creating forms for submitting information via E-mail, notice that this facility is not supported by all browsers. For example, a browser might just invoke its internal E-mail composer from scratch, ignoring the way in which the form has been filled! (This applies to Internet Explorer 3.0, for example.) Moreover, even if a browser supports this feature, the generated E-mail message is in the x-www-form-urlencoded form (which is confusing although not completely

illegible). To summarize, avoid using an ACTION which refers to a `mailto:` URL.

The forms concept in HTML 3.2, despite some complicated details, is essentially rather simple. In particular, it provides no way of **checking** form content (such requiring some field to be filled or contain numerical data) before submitting it. Any checking which is considered necessary must take place in the server to which the form content is submitted.

You can have more than one form in the same document.

The ISINDEX element predates the FORM element and was used for simple keyword searches.

Form submission takes place

- when the user selects (typically, with a mouse click) a form element which is an INPUT element with `TYPE=SUBMIT` (which is typically presented as a grey box, with text inside as specified in the HTML code) or
- when the user selects an INPUT element with `TYPE=IMAGE` (which is presented by graphical browsers using an image specified in the HTML code) or
- in some conditions, when the user *terminates a line* (typically by pressing an enter or return key) in a field defined as an INPUT element with `TYPE=TEXT`; one condition for this is that there are no other single-line input elements in the form, but for more details on this, consult Alan Flavell's document *Submit form by hitting ENTER?*
  The practical implication of the last item above is that you may, as an author, provide to *some* users a convenient way of submitting a form if you can organize things so that there is only one single-line text input field. You should not rely on this, however, so you should provide a normal submit field, too. Moreover, if you wish to take precautions against some people *accidentally* submitting a form before they have really finished filling it, organize things so that your form contains either no single-line text input fields or at least two of them. In any case, if you have only one such field, make it the last field in the form in order to minimize the risk of premature submission.

H1, H2, H3, H4, H5, H6 - headings

Purpose

To specify a heading. There are six levels of headers from H1 (the most important) to H6 (the least important).

Typical rendering

In large font and in bold face, often separated with blank lines from the text. More important headings are generally rendered in a larger font than less important ones. H1 headings are often very large font, whereas H6 can be tiny (even smaller than normal text!).

Basic syntax

<H*n*>*heading text*</H*n*>

where *n* is 1, 2, 3, 4, 5, or 6.

Possible attributes (Not in HTML 2.0!)

attribute name	possible values	meaning	notes
ALIGN	LEFT, CENTER, RIGHT	alignment of the heading	deprecated in HTML 4.0

The default is left alignment, but this can be overridden by an enclosing DIV or CENTER element. (HTML 2.0, which has no ALIGN attribute, contained no explicit rule for default alignment. On the other hand, it described "typical renderings" presenting H1 as centered and other headings with different amounts of left indentation.)

Allowed context

Block container.

Contents

Text elements.

Examples

Example H-1.html:

```
<H1>Notes on General Relativity</H1>
```

Example H-2.html:

```
<H1 ALIGN=CENTER>The story of my life</H1>
<H2>Preface</H2>
<H3>General remarks</H3>
```

There is a separate file which contains headings of all levels.

Notes

Documents should not skip heading levels, e.g. from H1 to H3 without intervening H2. This rule is not enforced by the formal syntax of HTML, but it has always been the recommended practice.

Avoid using H5 and H6 at all. More than four levels of headings are rarely needed, and popular browsers may display H5 and H6 in a manner which is *less* prominent than normal text!

See general structure recommendations for a detailed suggestion on heading usage.

In particular, don’t use e.g. H5 or H6 to cause text to be presented in a *small* font just because some browsers present them so. Other browsers - or even future versions of those browsers - may well adopt the more reasonable view that even the lowest level headings should be presented at least as prominently as normal text. If small font is what you really want, use the SMALL (or FONT) element.

Since heading elements are intended to be presented prominently by a browser, don’t make them very long. Normally you should not try add anything to the presentation by using text markup within the heading text. It is the job of a browser to present headings as headings. And for the same reason you should not write a heading in all upper case.

It might be a good idea to *make every heading an anchor*, i.e. a possible target of a link. Use the A element with NAME attribute for this. Example:
<H2><A NAME="intro">Introduction</A></H2>
Other people (or you) may then link to specific sections in your document, not just to the document as a whole. Notice that you must put the A element within the heading element, not vice versa.

HEAD - document head

Purpose

The basic structure of an HTML document always consists of a head and a body. It is not necessary to explicitly enclose the head into a HEAD element.

Typical rendering

Using an explicit HEAD element does not affect the document rendering.

Basic syntax

<HEAD>
TITLE element
</HEAD>

Both the start and end tags can be omitted.

Possible attributes

None.

Allowed context

The HTML element, which can be either implicit or explicit. Only one HEAD element is allowed in a document, and it must appear before the document body (which can be implicit or explicit).

Contents

Exactly one TITLE element, and optionally (in any order)

- an ISINDEX element
- a BASE element
- META elements
- LINK elements
- STYLE and SCRIPT elements

Examples

```
<HEAD>
<TITLE>Getting started with Perl</TITLE>
</HEAD>
```

Notes

The explicit use of a HEAD element has no other effect than making it explicit (to the reader of the HTML code) which part of the document belongs to the head section.

HR - change in topic (horizontal rule)

Purpose

To indicate change in topic, e.g. in order to separate sections of a document.

Typical rendering

A horizontal rule (full-width by default). Not necessarily preceded with or followed by vertical white space; you may wish to consider the effect putting the texts before and after an HR tag into P elements.

In a speech based user agent, the tag could be rendered as a pause.

Basic syntax

<HR>

Possible attributes (Not in HTML 2.0!)

attribute name	possible values	meaning	notes
ALIGN	LEFT, RIGHT, CENTER	horizontal alignment of the rule	default is CENTER
NOSHADE	NOSHADE	requests the rule to be rendered in a solid color	as opposite to the traditional two-color "groove"
SIZE	integer	height of the rule, in pixels	
WIDTH	width specification	width of the rule	

All of these attributes are deprecated in HTML 4.0

Allowed context

Block container.

Contents

None.

Examples

Example HR-1.html:

```
<P>
Some text, followed by a basic (default) horizontal rule.
</P>
<HR>
<P>
Some other text.
</P>
```

Example HR-2.html:

```
<P>
A horizontal rule placed at the right and half the width of
the document layout:
</P>
<HR ALIGN="RIGHT" WIDTH="50%">
<P>
An example with all possible spices: placed at left,
solid rule (no shading), height 5 pixels, width 100 pixels:
</P>
<HR ALIGN="LEFT" NOSHADE SIZE=5 WIDTH=100>
```

Notes

Don't overuse HR. The document may not look good if you have a lot of rules with just a little text between.

It is usually better to use a percentage specification than absolute number of pixels. The user's window might be very different from yours.

HTML - the top-level element in HTML

Purpose

Essentially, an HTML file in its entirety is an HTML element, but usually the start and end tags are omitted. See the description of the basic structure of HTML documents.

Typical rendering

Using an explicit HTML element does not affect the document rendering.

Basic syntax

<HTML>
the document head and body
</HTML>

Possible attributes

attribute name	possible values	meaning	notes
VERSION	*string*	version of HTML	deprecated in HTML 4.0

Allowed context

(The HTML element is the top level element in the HTML language. See the description of the basic structure of HTML documents.)

Contents

HEAD followed by BODY.

Examples

Example hello.html:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<TITLE>Hello</TITLE>
Hello world
```

Notes

If used, the start and end HTML tags must go around the *entire* document but directly after the DOCTYPE declaration.

The VERSION attribute is rarely used, and the information in it is almost never used by browsers or other software. To specify the HTML version used, use a DOCTYPE declaration.

I - text in italics

Purpose

To present text in italics.

Typical rendering

Italics. See general notes on rendering markup.

Basic syntax

<I>*text*</I>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example I-1.html:

```
Usually the dog is said to form the species <I>Canis familiaris</I>,
but genetically dogs belong to the same species as the wolf,
<I>Canis lupus</I>.
```

Notes

Although the I element is physical markup and logical markup is to be preferred in general, there is a lot of use for I, particularly because there is no text-level element for quotations in general in HTML 3.2. See notes about this in the description of CITE.

However, don't overuse the I element. In particular, for *emphasis* use EM or STRONG, and for variables (placeholders) use VAR. See general notes on text markup.

Words and phrases taken as such from *other languages* (than the language in which the document is written), such as *status quo*, *Weltanschauung* or *sauna*, are often presented in italics. However, the more common the word or phrase is (in your text or in your language in general), the less the reader benefits from designating them as foreign and the more he may be disturbed by the frequent occurrence of different fonts in the text.

In linguistics, when referring to words and phrases as in "the plural of *ox* is *oxen*", it is normal to use italics. (HTML 2.0 suggests the use of SAMP for such purposes, but that would be unnatural.)

The rules for *scientific names* for organismssay that the names should be written in italics if possible, so it is natural to write them within I elements. The same applies to *symbols of physical quantities* such as *F* for force; the VAR element might sound suitable, but I elements are rendered in the required way, in italics, more probably than VAR elements are.

IMG - inline images

Purpose

To include an image into the document.

Typical rendering

The image is presented as part of the document. Notice that the quality of presentation may vary a lot. Non-graphical browsers present the value of the ALT attribute instead. Moreover, a graphical browser can be used with automatic image loading off; in that case it may present an IMG element as a small generic symbol of images with the ALT text attached.

The positioning of the image is affected by the attributes of the IMG element.

Basic syntax

<IMG SRC="*URL*" ALT="*text*">

Possible attributes

attribute name	possible values	meaning	notes
SRC	*URL*	address of the image	obligatory; see notes on graphics formats
ALT	*string*	text description of the image	strongly recommendable, and required in HTML 4.0.
ALIGN	TOP, MIDDLE, BOTTOM, LEFT, RIGHT	positioning of the image relative to the current textline	default is BOTTOM
HEIGHT	*integer*	suggested height, in pixels	suggestion only
WIDTH	*integer*	suggested width, in pixels	suggestion only
BORDER	*integer*	suggested line border width, in pixels	relevant when the IMG element appears as an anchor text; BORDER=0 suppresses the border
HSPACE	*integer*	suggested horizontal gutter (width of white space to the immediate left and right of the image), in pixels	default value is a small non-zero number
VSPACE	*integer*	suggested vertical gutter (height of white space above and below the image), in pixels	default value is a small non-zero number
USEMAP	*URL*	fragment identifier for a client-side image map	maps are defined with the MAP element; names of maps are case sensitive
ISMAP	ISMAP	indicates that the image is a server-side image map	when the user clicks on the image, this attribute causes the cursor location to be passed to the server.

The presentational attributes ALIGN, BORDER, HSPACE, and VSPACE are deprecated in HTML 4.0.

Attributes HEIGHT, WIDTH, HSPACE, VSPACE, and USEMAP were not in HTML 2.0! And in HTML 2.0 the allowed values for ALIGN were TOP, MIDDLE, BOTTOM only.

The WIDTH and HEIGHT attributes, when used together, allow browsers to reserve screen space for the image before the image data has arrived over the network. This may imply faster formatting and allow the user start reading while data transfer is still in progress. These attributes were *not* designed for automatic resizing of images by browsers. Although *some* browsers are able to *scale* the image according to WIDTH and HEIGHT attributes, don´t rely on it. Even if a browser does the scaling, it might do it very poorly, distorting the image, and the scaling might slow down page repainting a lot. Thus the attributes, if used, should specify the **true size** of the image. (Use a suitable program, such as xv on many Unix systems, for finding out the size in pixels and for scaling the image if needed.) On

the other hand, many people omit these attributes; one reason to that is that when image loading is off, a graphical browser typically still reserves space according to these attributes, if present, and this may imply that the ALT text does not fit there. Thus it seems advisable **not** to include these attributes if the image is small and the ALT text is important.

The different values of ALIGN have the following meanings:

ALIGN=TOP
: Positions the top of the image with the top of the current text line. Browsers vary in how they interpret this. Some only take into account what has occurred on the text line prior to the IMG element and ignore what happens after it.

ALIGN=MIDDLE
: Aligns the middle of the image with the baseline for the current textline. Warning: several browsers interpret this incorrectly for small images, aligning the middle of the image with the *middle* (vertically) of the text line. This might be what the author wants, but it's not what browsers should do, and it's not what e.g. Netscape does. Thus, for vertically positioning an image with respect to some text, it might be more reliable to put the image and the text to cells of a table row and use the VALIGN attribute for table cells.

ALIGN=BOTTOM (default)
: Aligns the bottom of the image with the baseline.

ALIGN=LEFT
: Floats the image to the current left margin, temporarily changing this margin, so that subsequent text is flowed along the image's right hand side. The rendering depends on whether there is any left aligned text or images that appear earlier than the current image in the markup. Such text (but not images) generally forces left aligned images to wrap to a new line, with the subsequent text continuing on the former line.

ALIGN=RIGHT
: Floats the image to the current right margin, temporarily changing this margin, so that subsequent text is flowed along the image's left hand side. The rendering depends on whether there is any right aligned text or images that appear earlier than the current image in the markup. Such text (but not images) generally forces right aligned images to wrap to a new line, with the subsequent text continuing on the former line.

Note that some browsers (e.g. Internet Explorer 2.0 and 3.0) introduce spurious spacing with multiple left or right aligned images. As a result authors can't depend on this being the same for browsers from different vendors. See BR for ways to control text flow.

When an IMG element contains the ISMAP attribute, the element must be contained in an A element with an HREF attribute. Example:

```
<a href="/cgibin/navbar.map"><img src=navbar.gif ismap border=0></a>
```

The location clicked is passed to the server as follows. The user agent derives a new URL from the URL specified by the HREF attribute by appending a question mark (?), the x coordinate, a comma (,), and and the y coordinate of the location, with coordinates expressed in in pixels. The link is then followed using the new URL. For instance, if the user clicked at at the location x=10, y=27 then the derived URL will be: `"/cgibin/navbar.map?10,27"`. - It is generally a good idea to suppress the border (using the attribute BORDER=0) and explicitly tell that the image is clickable.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements.

Contents

None.

Examples

A basic example:

Example IMG-1.html:

```
<IMG SRC="Yucca.jpg" ALT="[Picture of Yucca]" WIDTH=110 HEIGHT=168>
<P>
<IMG SRC="Yucca.jpg" ALT="[Picture of Yucca]" WIDTH=110 HEIGHT=168
 ALIGN=RIGHT>
This is a simple example of embedding images.
This paragraph should be displayed, in a graphical browser,
with an image at the right,
and before this paragraph the same image should appear
separately, with default alignment.
</P>
```

Using IMG with ISMAP, to create a clickable map:

Example IMG-2.html:

```
<A HREF="http://www.hut.fi/cgi-bin/imagemap/Pictures/English/english.map">
<IMG HEIGHT="400" WIDTH="400"
 SRC="http://www.hut.fi/Pictures/English/english.gif"
 ALT="Helsinki University of Technology" ISMAP>
</A>
```

Notes

See the general discussion of images, formulas, etc, which contains additional examples.

There is no HTML feature specifically intended for a *caption* for an image. One reasonable way of including a caption (when the image appears on its own and not alongside with the text) is the following:

Example imgcaption.html:

```
<P>
<IMG SRC="sae.gif" ALT="[Siamese algae eater]">
<BR>
Siamese algae eater. <SMALL>Drawing by
<A HREF="http://www.hut.fi/u/lsarakon/">Liisa Sarakontu</A>.</SMALL>
</P>
```

If you want a picture appear at the left (or right) of a text paragraph, you should put the IMG element (with ALIGN=LEFT or ALIGN=RIGHT attribute) at the **beginning** of the paragraph (P element). Otherwise the result may look messy. Moreover, it is good practise to have a **BR element with the CLEAR attribute at the end** of such a paragraph, to avoid confusing effects. In general, putting an

image alongside with text is a potential source of problems; for example, a user with a narrow window might not see the text at all.

Dianne Gorman has written an illustrative document *Aligning Images and Text* (part of her *Introduction to HTML*).

The **ALT** attribute is not formally required in HTML 3.2, but it is wise to include it for every image. In some cases it is very obvious what you should put there. For example, if the image presents some text in a specific visual form, it is natural to have that text as the value of an ALT attribute. If the image is purely decorative, you should use ALT="". But there are cases where it can be somewhat difficult to create a suitable ALT attribute.

The semantics and intended use of the ALT attribute are somewhat vague. It might be viewed as a recommended way of providing a textual presentation of the contents of an image, to be used *as a replacement for the image* in text-only browsers, speech-based user agents etc. However, much more typically it contains a verbal *explanation of the image*, such as a *title* or perhaps just a name for the image. This seems suitable in the common situation of using a graphical browser with automatic image loading disabled: the user decides on the basis of the verbal explanation whether to load this particular image. (Graphic browsers vary in their behavior in such situation: treatment of ALT attributes in the situation where the user has turned off some browsers display the ALT value, others may display a small generic image which says very little.) And it is often difficult to say how the ALT text could be a good replacement for the image, since the syntax restricts the value to be just a string with no HTML markup. - Alan Flavell has written an extensive document *Use of ALT texts in IMGs*. See also my *Simple guidelines on using ALT texts in IMG elements*.

There are two ways of implementing clickable **image maps** in HTML documents:

server side image map
: Requires specific support in a Web server, but such support exists in most servers. The client (browser) essentially just sends the coordinates of the clicked location to the server, which then must take care of the rest. To use a server side image map, you use an A element containing an IMG element with ISMAP attribute. The HREF attribute of the A element specifies the address of the server (typically a script named `imagemap` or `htimage`; consult the documentation of the server).

client side image map
: Requires a client (browser) which supports MAP and AREA elements. (Newest versions of most popular browsers support them.) The HTML document uses these elements to specify the correspondence of areas of an image and associated documents (URLs). Usually some special program (image map editor) is used for the purpose.

Since client side image maps are faster and have other benefits as well but are not supported by all browsers, you may wish to *combine* server side and client side image maps in the following way:

```
<A HREF="/cgi-bin/htimage/your.map">
<IMG SRC="image/your.gif" ... ISMAP USEMAP="#yourmap"></A>
```

That way new browsers will use the client side image map, whereas old browsers will ignore the USEMAP attribute and pass the request to the server.

For more information about image maps, see e.g.

- section *How do I set up a clickable image map?* of the *World Wide Web FAQ*
- Imagemap Help Page
- *Imagemaps - Text-friendly? Cache-friendly?*

Image maps can be very useful in association with geographical maps. (See e.g. the "Virtual Tourist" map at `http://www.vtourist.com/webmap/`.) They might conceivably be used in other contexts as well, for instance to allow the user select an item in a display of purchasable objects or a detail in a plan of a house or to request information about a part of a device described by a drawing. In general, an imagemap can be very useful for things which are *inherently visual in two or more dimensions*. However, in actual practice **most use of image maps is abuse**. Example 2 above is a typical case: a natural, simple text menu would be easier to use and more efficient, and it would work fine on text-only browsers, too. (See section *Using tables to represent menus* for various implementations of menus.)

INPUT - input fields in forms

Purpose

To specify, within a form, input fields such as single line text fields, password fields, checkboxes, radio buttons, submit and reset buttons, hidden fields, file upload, image buttons, etc.

Typical rendering

Varies according to the field type.

Basic syntax

<INPUT TYPE=*inputtype other_attributes*>

Possible attributes

attribute name	possible values	meaning	notes
TYPE	TEXT, PASSWORD, CHECKBOX, RADIO, SUBMIT, RESET, FILE, HIDDEN, IMAGE	type of the input field	default is TEXT
NAME	*string*	name to be used to identify the field when submitting the contents to the server	required for all but SUBMIT and RESET
VALUE	*string*	initial value of the input field; when TYPE is SUBMIT or RESET, provides a textual label	obligatory, if TYPE is RADIO or CHECKBOX
CHECKED	CHECKED	when TYPE is RADIO or CHECKBOX, initializes the field to checked state	
SIZE	*integer*	visible size of the field, as number of average character widths	
MAXLENGTH	*integer*	maximum number of characters permitted in a text field	default is: no limit
SRC	*URL*	address of an image	for fields with background images
ALIGN	TOP, MIDDLE, BOTTOM, LEFT, RIGHT	image alignment for graphical submit buttons	as ALIGN in IMG (and HTML 2.0 allows only TOP, MIDDLE, BOTTOM here, too); default is BOTTOM; this attribute is deprecated in HTML 4.0

The different values of the TYPE attribute correspond to different kinds of input fields as follows.

TYPE=TEXT (the default)

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

Notice that text input is restricted to a single line. Use the TEXTAREA element to define multi-line text fields.

Typically browsers display the field content (both the initial content and the content entered by a user) in a fixed font. Although an INPUT element may occur within text level markup, it is thus usually not affected by it.

In some situations, if the user terminates a line in a text input field, it may cause immediate form submission.

Example:

```
<INPUT TYPE=TEXT SIZE=40 NAME=user value="your name">
```

TYPE=PASSWORD

This is like TYPE=TEXT but the browser should not echo the characters, so that people around the user will not see them. Typically, the browser uses a generic character like * to indicate that some character has been sent. The actual input is sent normally (without encryption!). You can use SIZE and MAXLENGTH attributes to control the visible and maximum length exactly as for regular text fields.

Example:

```
<INPUT TYPE=PASSWORD SIZE=12 NAME=pw>
```

TYPE=CHECKBOX

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

Thus, a set of INPUT TYPE=CHECKBOX elements represents an *n-of-many* choice field, whereas a set of INPUT TYPE=RADIO elements represents a 1-of-many choice field. Cf. to the SELECT element.

Example:

```
<INPUT TYPE=CHECKBOX CHECKED NAME=uscitizen VALUE=yes>
```

TYPE=RADIO

Used for attributes which can take a single value from a set of alternatives. Each radio button field in the group should be given the same NAME attribute. Radio buttons require an explicit VALUE attribute. Only the checked radio button in the group generates a name/value pair in the submitted data.

One radio button in each group should be initially checked (thus providing a default value) using the CHECKED attribute. (The HTML 2.0 specification says that if this is not the case, the browser should check the first button initially. But HTML 3.2 requires that the form itself provide a CHECKED attribute for one of the fields.) Normally the default value should be a *neutral* value, often indicating that the user does not want to or cannot give the information requested. For more information about

this, see my document *Choices in HTML forms*.

At all times, exactly one of the radio buttons in a set is checked; so if the user changes the selection, the browser must uncheck the button that was previously checked.

Example:

```
<INPUT TYPE=RADIO NAME=age VALUE="?" CHECKED>unspecified<BR>
<INPUT TYPE=RADIO NAME=age VALUE="0-12">0-12 years<BR>
<INPUT TYPE=RADIO NAME=age VALUE="13-17">13-17 years<BR>
<INPUT TYPE=RADIO NAME=age VALUE="18-25">18-25 years<BR>
<INPUT TYPE=RADIO NAME=age VALUE="26-35">26-35 years<BR>
<INPUT TYPE=RADIO NAME=age VALUE="36-">36 or more years<BR>
```

TYPE=SUBMIT

This defines a button or other presentation that users can select (e.g. by clicking) to submit the contents of the form to the server. A label is set for the button from the VALUE attribute. If the NAME attribute is given, then the name/value pair for the submit button will be included in the submitted data. You can include several submit buttons in the form. See TYPE=IMAGE for graphical submit buttons.

Example:

```
<INPUT TYPE=SUBMIT VALUE="Party on ...">
```

TYPE=RESET

This defines a button or other presentation that users can select (e.g. by clicking) to reset form fields to their initial state when the document was first loaded. You can set a label by providing a VALUE attribute. Reset buttons are never sent as part of the contents of a form.

It is customary to include a reset button after the submit button, but this means that if a user accidentally clicks on the wrong button, all information entered by him using the form is lost. So perhaps you should put the reset button somewhere else (if you include it at all). For a typical form, a reset button is rarely useful, since it is unlikely that the user wishes to clear *all* fields. However, a reset button might be useful if the form has e.g. prefilled text fields; see the fourth example in the description of the FORM element.

Example:

```
<INPUT TYPE=RESET VALUE="Start over ...">
```

TYPE=FILE (Not in HTML 2.0!)

This provides a means for users to attach a file to the contents of the form.

This feature is *not commonly supported yet*. Notice that some browsers support it seemingly only, e.g. including the *name* of the file instead of its *contents*!

The element is generally rendered as a text field and an associated button or other presentation which, when selected (e.g. by clicking), invokes a file browser to select a file name. The file name can also be entered directly in the text field.

Just like for TYPE=TEXT you can use the SIZE attribute to set the visible width of this field in average character widths. You can set an upper limit to the length of file names using the MAXLENGTH attribute.

Some browsers support the ability to *restrict the kinds of files* (that can be attached to the contents of a form) using an ACCEPT attribute. The value of that attribute is a comma-separated list of MIME content types. For example, `ACCEPT="image/*"` would restrict files to images. Notice that the ACCEPT attribute is not defined in HTML 3.2, although it is defined in RFC 1867 to which the *HTML 3.2 Reference Specification* refers in this context for further information.

The value of the ENCTYPE attribute is an Internet media type which in this context specifies the data encoding used. The default is `application/x-www-form-urlencoded` (which is an unregistered type, but one that browsers are required to support). Another type which browsers should support is `multipart/form-data` (see RFC 1867). Browsers may support other media types in this context, too.

Further information on using forms for file upload can be found in RFC 1867. See also item *How can I allow file uploads to my web site?* in the *WDG Web Authoring FAQ* and my detailed document *File input (or "upload") in HTML forms*.

Example:

```
<INPUT TYPE=FILE NAME=photo SIZE=20>
```

TYPE=HIDDEN

This indicates that the field should not be rendered to the user. A hidden field provides a means for servers to store state information with a form. This will be passed back to the server when the form is submitted, using the name/value pair defined by the corresponding attributes. This is a workaround for the statefulness of HTTP and an alternative to using so-called HTTP cookies.

Example:

```
<INPUT TYPE=HIDDEN NAME=customerid VALUE="c2415-345-8563">
```

TYPE=IMAGE

This acts as a submit button (cf. TYPE=SUBMIT), but it is rendered by an image rather than a text string and the form is submitted so that information about the clicked location is passed, too. The URL for the image is specified with the SRC attribute. The image alignment can be specified with the ALIGN attribute. In this respect, graphical submit buttons are treated identically to IMG elements (so you can set ALIGN to LEFT, RIGHT, TOP, MIDDLE or BOTTOM). A NAME attribute is required. When the user clicks on the button, the *x* and *y* coordinates of the location clicked are passed to the server is two name/value pairs. The names are derived by taking the name of the field and appending `.x` for the *x* value and `.y` for the *y* value.

Example:

```
<P>Now choose a point on the map:
<INPUT TYPE=IMAGE SRC="map.gif" NAME="point">
```

Notice that *image fields cause problems to people using text-only or speech-based user agents* or graphical browsers with automatic loading of images disabled.

The specifications do not mention the VALUE attribute for INPUT TYPE=IMAGE, but at least one text-mode browser takes its value it as the substitute for the image. Thus, defining a meaningful VALUE attribute is good idea, if the form makes sense even if the script processing it does not get (meaningful) *x* and *y* values. For more information, see Alan Flavell's *INPUT TYPE=IMAGE for text users?*

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. However, the text container must appear within a FORM element.

Contents

None.

Examples

```
<INPUT TYPE=RESET VALUE="Start over ...">
```

Notes

See the description of the FORM element, which contains some examples of entire forms.

The use of INPUT for text input is restricted to single line fields. Use TEXTAREA to define multi-line text fields.

ISINDEX - simple keyword searches

Purpose

Simple keyword searches. The browser should provide a single line text input field for entering a query string.

The semantics for ISINDEX are currently well defined only when the base URL for the enclosing document is an HTTP URL. Typically, when the user presses the enter (return) key, the query string is sent to the server identified by the base URL for this document. For example, if the query string entered is "ten green apples" and the base URL is:

```
http://www.acme.com/
```

then the query generated is:

```
http://www.acme.com/?ten+green+apples"
```

The ISINDEX element only provides an interface to a program (typically, a CGI script) which interprets the query. Merely inserting an ISINDEX element does not make the document searchable! (On the other hand, notice that most Web browsers provide some "search in this document" feature, so you need not take any special efforts in order to allow your readers perform simple searches within a document.)

Basic syntax

<ISINDEX>

Typical rendering

An input area (in graphical browsers, an input box) prefixed with a prompt string.

Possible attributes (Not in HTML 2.0!)

attribute name	possible values	meaning
PROMPT	*string*	prompt message

The PROMPT attribute can be used to specify a prompt string for the input field, replacing a browser-dependent default prompt string (which might be e.g. *This is a searchable index. Enter search keywords*).

Allowed context

At most one ISINDEX element may appear in a document, either in the head or in the body.

Contents

None.

Examples

This demonstrates the use of ISINDEX for interfacing to a "finger" script. The script itself is not discussed here, but it is of course essential that it can handle the queries generated.

Example ISINDEX.html:

```
<BASE HREF="http://www.hut.fi/cgi-bin/finger">
Searching for a user at <a href="http://www.hut.fi/">HUT</a>.
<ISINDEX PROMPT="User id at HUT:">
```

Notes

For more flexibility, use the newer FORM element instead. In HTML 4.0, the ISINDEX element is deprecated .

There are no restrictions on the number of characters that can be entered in the query string.

In practice, the query string is restricted to Latin-1 as there is no current mechanism for the URL to specify a character set for the query.

When the query is generated from the input, space characters are mapped to "+" characters, and normal URL character escaping mechanisms apply. For further details see the HTTP specification.

KBD - keyboard input

Purpose

To present a particular command or data string to be entered by the user. Typically this is used in instruction manuals.

Typical rendering

Monospaced. See general notes on rendering markup.

Basic syntax

<KBD>*text*</KBD>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example KBD-1.html:

```
Finally, type <KBD>logout</KBD> and press the return key.
```

Notes

Use the KBD element for *fixed* strings only. To indicate input which varies from one case to another, use the VAR element.

Although *program code* might be regarded as keyboard input (to be typed by a programmer), especially in the context of teaching programming, it is more natural to use the CODE element for code fragments.

It is arguable whether one should use the KBD element for *command names* (or names of programs) as well, even when they do not appear in a context which discusses how commands are given. One might say that a command name like **`ls`** (in Unix) is just a name, not keyboard input. But I recommend using KBD, since it is difficult and sometimes quite artificial to distinguish e.g. **`ls`** as keyboard input (or part of it) and as the name of a command (or program). Notice that when a command name appears at the beginning of a statement, grammar rules require a capital initial which might be misleasing (by suggesting to the user that the case of letters in irrelevant on keyboard input); by using KBD - usually rendered using a monospaced font, and therefore distinguishing the command

name from normal text - we make it more acceptable to violate the grammar rule.

As usual in HTML, division into lines and the use of blanks and tabs is selected by the browser, not honoring the one in the HTML file. Be careful in telling the user when he should press the return or enter key, since this may not correspond to the visual layout of your instructions.

See also notes on presenting interaction with computer and general remarks on phrase elements.

LI - list item

Purpose

To present an item in a list.

Typical rendering

The rendering depends on the nature of the enclosing list.

Basic syntax

<LI>*contents of the list item*</LI>

The end tag </LI> can always be omitted, and it usually is omitted.

Possible attributes (Not in HTML 2.0!)

The attributes depend on the context as follows.

When the (innermost) enclosing list element is UL or DIR or MENU:

attribute name	possible values	meaning
TYPE	DISC, SQUARE, CIRCLE	bullet style

When the (innermost) enclosing list element is OL:

attribute name	possible values	meaning
TYPE	1, a, A, i, I	numbering style (as in OL)
VALUE	integer	sequence number (see OL)

In both cases, the attributes are deprecated in HTML 4.0. For VALUE, especially non-positive values are inconsistently supported in browsers.

Allowed context

UL, DIR, MENU, or OL element.

Contents

Block elements and text elements. Notice that heading and ADDRESS elements are not allowed.

Examples

An example which does not say very much:

```
<LI>A list item.</LI>
```

For more realistic examples, see *Examples of various list elements in HTML* and examples given in the descriptions of UL, DIR, MENU,and OL element.

Notes

LI elements may contain lists, producing nested lists.

The list of bullet types was chosen to cater for the original bullet shapes used by Mosaic in 1993. The list is not very logical. Usually the default bullet type in UL lists is DISC, if the list is not within a UL list, and SQUARE and CIRCLE in the next levels of nesting. In Lynx, the situation is similar with the shapes DISC, SQUARE, and CIRCLE presented as star (*), plus (+) and letter ○.

It is hard to imagine any good use for the TYPE attribute in a LI element, as opposite to defining the bullet type for all items of a list in a UL element or other list element.

LINK - relationships with other documents

Purpose

To specify relationships with other documents, i.e. links between documents. . Currently this element is not very useful, since few browsers or other programs make use of it. LINK elements *could* (and perhaps some day will) be used for very important things such as

- for toolbars or menus for navigation in a web of documents (interlinked with LINK elements), thus allowing e.g. different "guided tours" for different users
- to control how collections of HTML files are rendered into printed documents or converted into a single document for some other purpose.
- for using style sheets

Typical rendering

The LINK elements do not directly affect the rendering of the document itself. They might have some effect on the presentation of information *about* the document, e.g. on the browser window elsewhere than in the display of the document itself. Moreover, if a LINK element is used to specify a style sheet, the effect on rendering can be very important.

Basic syntax

<LINK REL=*relation* HREF=*URL*>

or

<LINK REV=*relation* HREF=*URL*>

Possible attributes

attribute name	possible values	meaning
HREF	*URL*	URL for linked resource
REL	*string*	type of "forward" link
REV	*string*	type of "reverse" link
TITLE	*string*	advisory title string for the linked resource

A link from document A to document B with REV=*relation* expresses the same relationship as a link from B to A with REL=*relation*.

Allowed context

The head element, in which any number of LINK elements may appear.

Contents

None.

Examples

A link element which specifies a style sheet to be used:

```
<LINK REL=STYLESHEET HREF="basic.css">
```

A simple LINK element providing authorship information:

```
<LINK REV=MADE HREF="mailto:jukka.korpela@hut.fi">
```

Some LINK elements which might appear in a large document divided into separate but interlinked HTML files:

```
<LINK REL=CONTENTS HREF="toc.html">
<LINK REL=PREV HREF="doc31.html">
<LINK REL=NEXT HREF="doc33.html">
```

Notes

See the general description of links, especially notes on REL and REV values.

A LINK element with **REV=MADE** is sometimes used to identify the document author, either the author's email address with a mailto URL (as in the example above), or a link to the author's home

page. Although only a few programs (most notably Lynx) make any use of such information, it can be useful to include it, since it also works as a comment-like note to a person reading the HTML source. Notice that the information is *not* shown to the reader of the document (unless he specifically requests to see the HTML code, of course), so you should additionally provide such information using the ADDRESS element, for example.

MAP - clickable map (Not in HTML 2.0!)

Purpose

To provide a mechanism for client-side image maps. A MAP element has a name through which it can be referred to in an IMG element. A MAP element contains AREA elements which specify hotzones on the associated image and bind these hotzones to URLs.

Typical rendering

The visual appearance of the document is not directly affected by a MAP element, but the element, together with associated structures, makes an image into a clickable map.

Basic syntax

<MAP NAME=*name*>
AREA elements
</MAP>

Possible attributes

attribute name	possible values	meaning	notes
NAME	*string*	a name for the map, referable to in USEMAP attributes of IMG elements	obligatory; case sensitive

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements.

Contents

AREA elements.

Examples

A simple example for a graphical navigational toolbar:

```
<IMG SRC="navbar.gif" BORDER=0 USEMAP="#map1">

<MAP NAME="map1">
 <AREA HREF="guide.html" ALT="Access Guide" SHAPE=RECT COORDS="0,0,118,28">
 <AREA HREF="search.html" ALT="Search" SHAPE=RECT COORDS="184,0,276,28">
 <AREA HREF="shortcut.html" ALT="Go" SHAPE=RECT COORDS="118,0,184,28">
 <AREA HREF="top10.html" ALT="Top Ten" SHAPE=RECT COORDS="276,0,373,28">
</map>
```

Notes

See general description of server side and client side image maps in the description of the IMG element.

MENU - unnumbered list in menu-like form

Purpose

To present information in a menu-like format.

Typical rendering

In practise, most browsers present a MENU element exactly the same way as an UL element. A few browsers omit the bullets, however.

Theoretically, the recommendation has been and still is that MENU element be rendered as a single column menu list which is more compact than a UL element.

Basic syntax

<MENU>
<LI> *list item 1*
<LI> *list item 2*
...
</MENU>

Possible attributes

attribute name	possible values	meaning
COMPACT	COMPACT	reduced interim spacing

Typically, browsers ignore the COMPACT attribute.

Allowed context

Block container.

Contents

LI elements which do not contain block elements.

Examples

Example MENU.html:

```
<MENU>
<LI> Undo
<LI> Cut
<LI> Copy
<LI> Paste
<LI> Find
<LI> Find Again
</MENU>
```

See also *Examples of various list elements in HTML.*

Notes

In HTML 4.0, the MENU element is deprecated in favor of the UL element; style sheets can be used to suggest features of the presentation of a list.

See general notes about list elements for a discussion of selecting between them.

The name of the element might be misleading. There is no true selection menu involved, just a display of menu keywords. To present a true selection menu you can use hyperlink anchors (A elements). See the section *Using tables to represent menus*.

META - meta info

Purpose

To supply meta info (information *about* the document) as name-value pairs describing properties of the document, such as author, expiry date, a list of key words etc.

It depends on programs (e.g. browsers or search engines) processing HTML files what they do with the info.

Typical rendering

None. The META elements do not affect the rendering of the document itself. They might have some effect on the presentation of information *about* the document, e.g. on the browser window elsewhere than in the display of the document itself, or in the query reports from search engines.

Basic syntax

<META NAME=*info item name* CONTENT=*info contents*>
or
<META HTTP-EQUIV=*info item name* CONTENT=*info contents*>

Possible attributes

attribute name	possible values	meaning	notes
NAME	*name*	meta information item name	alternative to HTTP-EQUIV attribute
HTTP-EQUIV	*name*	meta information item name	alternative to NAME attribute
CONTENT	*string*	meta information contents	a META element *must* contain this attribute

Allowed context

The head element, in which any number of META elements may appear.

Contents

None.

Examples

```
<META NAME=DESCRIPTION CONTENT=
"An extensive guide to writing HTML 3.2 documents,
with examples and practical advice.">
<META NAME=KEYWORDS CONTENT="structural HTML, logical markup">
```

Notes

For detailed information, see *A Dictionary of HTML META Tags* on Vancouver Webpages.

Several Web search engines, such as InfoSeek and AltaVista, recognize META elements with NAME values DESCRIPTION and KEYWORDS. The words listed in the CONTENT attribute of a META NAME=KEYWORDS element might be used (and perhaps emphasized) when indexing documents; however, generally such keywords are useful only if they occur in the normal text of the document too, and in that case you can expect them to be used in indexing anyway! On the other hand, a META NAME=DESCRIPTION is **recommendable**, since many (but not all) search engines show the CONTENT value as the abstract for the document when returning query results. But you should *also* take into account that many search engines just take the first few words of the document, so you might include a short summary into the document body right after the main heading.

For some more information, consult

- *Strategies for Indexing and Search Engines* in *HTML Unleashed, Professional Reference Edition*
- *Submitting Tips* at InfoSeek
- *METAtags* in the online help of AltaVista.
- *Search Engine Tutorial*
- *Search Engine Simulator* by Delorie; very nice for getting an idea of what your page might look like in reports from search engines which do *not* pay attention to `META` elements!

The META tag affects the way your document is *indexed* when it is included into a data base of a search engine. It will not make a robot *find* the document when it searches candidates for inclusion into a data base. Therefore, if you think the document is important, and especially if there are not several links to it in other documents, consider *additionally* using facilities like "Add URL" on the AltaVista main page.

The difference between NAME and HTTP-EQUIV is that the latter has a special significance when documents are retrieved via HTTP, whereas the interpretation of NAME attributes is up to each particular browser or other program which processes HTML files (although some common practices may emerge and might be standardized later). HTTP servers may use the property name specified by the HTTP-EQUIV attribute to create an RFC 822 style header in the HTTP response. (RFC 822 is the electronic mail protocol used on the Internet.) The header name (which is case insensitive) is taken from the HTTP-EQUIV attribute value, and the header value is taken from the value of the CONTENT attribute. For a good introduction to HTTP headers, consult the tutorial *How the web works: HTTP and CGI explained* by Lars Marius Garshol.

A server may disregard any META elements which specify information controlled by the server, such as "Server", "Date", and "Last-modified"; see the HTTP specification for details.

For example,

```
<META HTTP-EQUIV="Expires" CONTENT="Tue, 20 Aug 1996 14:25:27 GMT">
```

will result in the HTTP header
`Expires: Tue, 20 Aug 1996 14:25:27 GMT`
and this might be used by caches to determine when to fetch a fresh copy of the associated document. Notice that according to HTTP 1.0 specification (RFC 1945) the expiration time must be expressed in one of a few strictly defined formats, the preferred one being exemplified above (and formally defined in RFC 822 and RFC 1123).

If an organization enforces authors to include meta information such as authorship information and expiration times in a specific format, special software might be written to scan through the WWW server periodically in order to send automatic reminders to authors.

OL - ordered (numbered) list

Purpose

To present information in the form of an ordered (numbered) list.

Typical rendering

The list items are presented separately, although possibly with less space between them than there is e.g. between paragraphs. The presentation is often indented in a manner which causes nested lists to be indented according to their structure.

In contrast with the UL element, the items are numbered (consecutively by default).

Basic syntax

<OL>
<LI> *list item 1*
<LI> *list item 2*
...
</OL>

Possible attributes

attribute name	possible values	meaning	notes
TYPE	1, a, A, i, I	numbering style	case of letter is significant
START	*integer*	starting sequence number	default is 1; especially non-positive values are inconsistently supported
COMPACT	COMPACT	reduced interim spacing	often ignored by browsers

Attributes TYPE and START where not in HTML 2.0! All of the attributes are deprecated in HTML 4.0

The meanings of the values of TYPE are the following:

Type	Numbering style	The first few numbers
1	normal (Arabic) numbers	1, 2, 3, ...
a	Latin letters in lowercase	a, b, c, ...
A	Latin letters in uppercase	A, B, C, ...
i	Roman numbers in lowercase	i, ii, iii, ...
I	Roman numbers in uppercase	I, II, III, ...

Allowed context

Block container.

Contents

LI elements (one or more).

Examples

A simple example:

Example OL-1.html:

```
<P>
Proceed as follows:
</P>
<OL>
<LI> Try to guess how to use the program.
<LI> If it fails, send lots of questions to Usenet News.
<LI> If they flame you, consider contacting local user support.
<LI> When everything else fails, read the manuals.
</OL>
```

An example where it is natural to use Roman numbers:

Example OL-2.html:

```
<P>
The declinations of nouns in Latin are best distinguished by
the ending of the genitive singular:
</P>
<OL TYPE=I>
<LI> <I>-ae</I>, eg <I>terra:terrae</I>
<LI> <I>-i</I>, eg <I>annus:anni</I>
<LI> <I>-is</I>, eg <I>labor:laboris</I>
<LI> <I>-us</I>, eg <I>fructus:fructus</I>
<LI> <I>-ei</I>, eg <I>dies:diei</I>.
</OL>
```

A contrived example to show the effects of attributes and overriding them in LI elements.

Example OL-3.html:

```
<OL TYPE=a START=3 COMPACT>
<LI> first item
<LI> second item
<LI VALUE=8> item after skipping a few values
<LI> next item
<LI TYPE=A> going on with uppercase
<LI> this is the last item.
</OL>
```

See also *Examples of various list elements in HTML.*

Notes

See general notes about list elements for a discussion of selecting between them. It is natural to use an ordered list if the order of the items *should be emphasized*, e.g. when they are instructions to be followed in that sequence, a description of events in their temporal order, or things in order of importance.

The sequence numbers of the items start from the value of the START attribute (by default 1). You can set it later on with the VALUE attribute on LI elements. Both of these attributes expect integer values. (Even if you have set the TYPE attribute to something else than 1, the values of the VALUE attribute must be specified using the normal notation of numbers as sequences of digits.) You can't indicate that numbering should be continued from a previous list or skip missing values without giving an explicit number.

The START attribute in OL, as well as the VALUE attribute in LI inside OL, is inconsistently supported by browsers. Most browsers support *positive* values, though e.g. Opera 4.0 doesn't do that properly (fixed in version 5.0). But although a few browsers support negative and zero values, popular browsers have problems in that area; some browsers even support negative and positive value but not zero (treating START="0" as START="1"). It is unlikely that you would like to use negative values, but starting numbering from zero is fairly common. The browser inconsistencies however imply that this is not a good idea. If it is essential to make numbering start from zero, consider using a UL so that the numbers are part of the textual content in LI elements.

The alignment of numbers is unspecified. In particular, Roman numbers might be left or right aligned or centered. (This is outside the control of the document author when using the OL element; you may wish to consider the alternative of using a table.)

In nested OL lists, it would be natural to use numbering of the form *m.n* but the specifications are silent about this. In practice, and most browsers use simple numbering which is independent of any nesting.

OPTION - an option in a select menu

Purpose

To present one option in a select menu within a form.

Typical rendering

When the enclosing select menu is activated, the user can see the text of the option, either as part of a list of such text or by scanning through the options.

Basic syntax

<OPTION>*text*</OPTION>

The end tag can always be omitted.

Possible attributes

attribute name	possible values	meaning	notes
SELECTED	SELECTED	the option is selected by default	in a SELECT element without the MULTIPLE attribute, at most one OPTION element may have this set
VALUE	*string*	property value to be used when submitting the contents of the form; this is combined with the property name as given by the NAME attribute of the enclosing SELECT element	defaults to the contents of the element; however, several browsers ignore leading and/or trailing spaces in the content

According to the HTML 2.0 specification, "the initial state has the first option selected, unless a SELECTED attribute is present on any of the OPTION elements". On the other hand, the *HTML 3.2 Reference Specification* leaves the default initial state open, so it is safest to assume that it is browser-dependent (and it actually is). You may wish to deal with this problem by providing a dummy first option (e.g. "No selection") and making it SELECTED, thus ensuring the same behavior from all HTML 3.2 conformant browsers. See *Choices in HTML forms* for more info.

Allowed context

SELECT element.

Contents

A string. Escape sequences are allowed, but no tags are recognized.

Examples

```
<OPTION>female</OPTION>
```

P - normal paragraph

Purpose

To present a normal text paragraph.

Typical rendering

As a text paragraph, suitably separated (normally with some extra white space such as an empty line) from other paragraphs, headings etc. A browser might leave some extra space at the beginning of the first line; most browsers don’t.

Browsers usually format paragraphs to fit into the horizontal space (screen or window width) available.

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

Basic syntax

<P>*paragraph text*</P>

Possible attributes (Not in HTML 2.0!)

attribute name	possible values	meaning	notes
ALIGN	LEFT, CENTER, RIGHT	alignment of the paragraph (flush left, centered, flush right)	deprecated in HTML 4.0

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

Allowed context

Block container.

Contents

Text elements.

Examples

A normal example:

Example P-1.html:

```
<P>
This is a normal text paragraph which contains so many characters
that it will most probably be split into several lines by a browser.
</P>
```

A contrived example:

Example P-2.html:

```
<P>
This is a normal text paragraph with no attribute for horizontal
alignment. Nothing special.
</P>
<P ALIGN=CENTER>
<B>This is a paragraph which should be centered. It should also appear
in bold face but this results from explicit use of a B element.
Centering itself should not affect the font.</B>
</P>
<P ALIGN=RIGHT>
This is a paragraph which should be rendered flush right. It is difficult
to see why you would ever <EM>like</EM> to use this option!
</P>
```

See also the examples about BLOCKQUOTE, one of which makes reasonable use of ALIGN=RIGHT.

Notes

See the general discussion of paragraph-like elements for selecting a suitable HTML element for different kinds of paragraphs. In particular, if you have a collection of closely related small paragraphs, you may wish to consider making them into a list using UL and LI instead of P; this typically results in more compact presentation visually.

If you intend to use P for alignment purposes, such as centering text, remember that a P element may only contain text elements. The DIV element may contain block elements, too.

There is no way in HTML (in HTML 3.2 at least) to make text appear "justified" (solid-right), unless you want to resort to using the PRE element. More exactly, such presentation issues are browser-dependent, and the great majority of browsers use ragged right margin.

The end tag </P> can always be omitted, and it usually *is* omitted. It is, however, **advisable to use explicit </P> tags** for the following reasons:

- In practice, at least one popular browser (Internet Explorer) fails to infer a closing </P> tag in many occasions. This means that if you do not use an explicit </P> tag, then e.g. a table following a paragraph is rendered immediately after the text, with no space between.
- From a more theoretical point of view, omitting the end tag tends to strengthen an incorrect way of thinking: people may regard <P> as a paragraph *separator*, but in fact it *initiates* a paragraph (to be terminated by an explicit </P> or implicitly by tags like <P> or <H1>).
  One reason for regarding <P> as paragraph separator is that it *was* defined that way in an early HTML draft; notice that no *approved* HTML specification ever defined <P> that way.

Paragraphs cannot be nested. (This is the other side of the "nice" feature that </P> can be omitted.) One way of *simulating subparagraphs* is to use BR elements around a piece of text within a P element. Another way is to use list elements (such as UL) instead of P elements.

The division into lines in the rendering usually does not match the HTML source. See the section *Division into lines and the use of blanks and tabs*.

PARAM - applet parameters (Not in HTML 2.0!)

Purpose

To pass parameters to Java applets.

Typical rendering

Not rendered directly, but may affect the behavior of the applet.

Basic syntax

<PARAM NAME=*name* VALUE=*value*>

Possible attributes

attribute name	possible values	meaning	notes
NAME	*name*	name of the parameter	obligatory
VALUE	*string*	value of the parameter	

Allowed context

APPLET element.

Contents

None.

Examples

<PARAM NAME=snd VALUE="Hello.au">

Notes

Character escape sequences such as `é` and `¹` are expanded before the parameter value is passed to the applet. To include an & character use `&`.

PRE - preformatted text

Purpose

To include text to be displayed as such with respect to the use of blanks and newlines. This can be useful when there is information available in text-only form and we wish to put it onto Web, preferring immediate availability to nice layout. The text might also be e.g. computer output to be presented as it stands.

Typical rendering

The text is rendered in *monospaced* font, i.e. using a teletype-like font where all characters occupy the same amount of space horizontally. Use of blanks and newlines exactly corresponds to that of the HTML source within the PRE element.

Basic syntax

<PRE>
preformatted text
</PRE>

Possible attributes

attribute name	possible values	meaning	notes
WIDTH	*integer*	width of text in characters	not supported in general, and deprecated in HTML 4.0

The value of WIDTH should be equal to or greater than the length of the longest line. In principle, the WIDTH attribute is meant for providing a browser information which it can use to select a suitably-sized font or to adjust indentation to make the text fit. Unfortunately this is not usually done by browsers. You should not expect that e.g. text wider than 80 characters gets displayed correctly (even if you use the WIDTH attribute).

Allowed context

Block container.

Contents

Text element, with the **exclusion** of images (IMG) and changes in font size (BIG, SMALL, SUB, SUP, FONT) or any element that contains them.

Examples

The simplest example:

Example PRE-1.html:

```
<PRE>
To be or not to be,
that is the question.
</PRE>
```

A more realistic example:

Example PRE-2.html:

```
The printable characters of ASCII:
<PRE>
  ! " # $ % &amp; ' ( ) * + , - . /
0 1 2 3 4 5 6 7 8 9 : ; &lt; = &gt; ?
@ 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 [ \ ] ^ _
` 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 { | } ~
</PRE>
```

An attempt to present line printer like computer output:

Example PRE-3.html:

```
The printout from the program is the following. Each line contains ten
real numbers, each in a field of ten characters. Notice that when viewing
this document on WWW, the rendering of the printout can be unsatisfactory;
in such a case widen the WWW window, if possible.
<PRE WIDTH=100>
 0.5138707 0.1757256 0.3086337 0.5345317 0.9476302 0.1717277 0.7022309 0.2264168 0.4947661 0.1246986
 0.0838954 0.3896298 0.2772301 0.3680532 0.9834590 0.5353862 0.7656789 0.6464736 0.7671438 0.7802362
 0.8229621 0.1519211 0.6254769 0.3146764 0.3469039 0.9172033 0.5197607 0.4011658 0.6067690 0.7854244
</PRE>
```

In situations like this, you may consider the effect of using BASEFONT before PRE. (This is not a good solution but it might serve as a workaround until browsers begin to support the WIDTH attribute.)

An example of PRE element containing links (this might also be presented using a table):

Example PRE-4.html:

```
Contact information (phone and E-mail):
<PRE>
help desk     4344 <A HREF="mailto:atk-neuvonta@hut.fi">atk-neuvonta@hut.fi</A>
operators     4341 <A HREF="mailto:opr@hut.fi">opr@hut.fi</A>
WWW problems  4331 <A HREF="mailto:webmaster@hut.fi">webmaster@hut.fi</A>
</PRE>
```

The discussion of presenting interaction with computer contains an additional example with embedded text markup.

Notes

As an *alternative* to using PRE, consider using a normal paragraph so that every line is terminated with a BR element. This has the disadvantages of not preventing a browser from dividing lines (but if a browser splits lines, they are probably so long that a PRE element might cause problems too) and not preserving leading spaces or multiple spaces within a line. On the other hand it has the advantage of more flexibility, e.g. allowing the use of proportional fonts.

As *another alternative*, often suitable for large pieces of text or data, consider writing a separate text file to which you have a link in your HTML code.

Previous versions of HTML contained the XMP, LISTING, and PLAINTEXT elements. They are now obsolete, and PRE should be used instead.

One typical use for PRE has been to present **tables**, and this may still be a good idea in some cases (see example 2). However, HTML tables element can be used for much more advanced tabular presentation. (You might still consider the possibility of presenting your tables in two alternative forms, using TABLE as the basic form but providing a PRE form for those readers who use a non-table browser.)

Although A elements and phrase markup (e.g. STRONG) can be used, the capabilities of a browser in presenting them may be more restricted than outside PRE elements.

You can even use **tabs** in the preformatted text, although it is better to use multiple spaces, since you cannot be sure of how tab stops are set in the reader's environment. The language specification says that the tab character should position to the next 8 character boundary but **discourages** its use.

Although a browser must show the document so that line breaks correspond to those in the source code, a browser is not forbidden from using e.g. constant left **indentation** for preformatted paragraphs.

You cannot change font size *within* a PRE element (and you cannot put a PRE element inside a FONT element, for example), but the BASEFONT element affects preformatted text, too.

In principle, a **P** tag is not allowed within a PRE element, since P is block element, not text element. However, HTML 2.0 specification encourages browsers to accept it, with the remark a P within a PRE element should produce only one line break, not a line break plus a blank line.

If character **<** or **>** or **&** occurs in the data, it must be expressed using the escape syntax (as in example 2). In particular you must do so when including HTML code into your document for the purpose of displaying the source code.

The SGML standard requires that the parser remove a newline immediately following the start tag or immediately preceding the end tag. Thus it should not matter whether you have the <PRE> tag on a separate line or as a prefix to the first line of the text. However, some browsers fail in obeying this, so you may consider using the latter presentation to prevent an extra line.

SAMP - sample output

Purpose

To present sample output from programs, commands, scripts etc.

Typical rendering

Monospaced. See general notes on rendering markup.

Basic syntax

<SAMP>*text*</SAMP>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example SAMP-1.html:

```
The fatal error message <SAMP>Bus error - core dumped</SAMP> can be caused
by very different bugs in your program.
```

Notes

As usual in HTML, division into lines and the use of blanks and tabs is selected by the browser, not honoring the one in the HTML file. Thus, large pieces of output are more suitably presented using the PRE element or as separate text files to which you have links in HTML files.

In HTML 2.0 this element was defined as follows:

> The *SAMP* element indicates a sequence of literal characters, typically rendered in a mono-spaced font. For example:

```
The only word containing the letters <samp>mt</samp> is dreamt.
```

However, since the HTML 3.2 description is more specific and restrictive, you should use SAMP only to present sample output, *not* e.g. in the way the example in the HTML 2.0 specification suggests.

See also notes on presenting interaction with computer and general remarks on phrase elements.

SCRIPT - client-side scripting languages (Not in HTML 2.0!)

Purpose

Reserved for future use with scripting languages.

Typical rendering

browsers should hide the contents of SCRIPT elements. However, if the browser supports scripting, the script may affect the rendering of the document in many ways.

Basic syntax

<SCRIPT>*script statements*</SCRIPT>

Possible attributes

None according to HTML 3.2. In HTML 4.0, the SCRIPT element is defined so that it has an obligatory TYPE attribute and some optional attributes. (In some drafts and implementations, the attribute LANGUAGE has been used, but it has been deprecated in favor of the TYPE attribute.)

Allowed context

The head section and any text container. (The text part of the *HTML 3.2 Reference Specification* mentions only the head section as a place where SCRIPT element may occur, but the formal syntax (DTD) allows it in the BODY part as well, classifying SCRIPT as a text element. The latter is obviously the intent.)

Contents

Script statements. The syntax and semantics is to be defined separately.

Technically, these elements are defined with CDATA as the content type. As a result they may contain only SGML characters. All markup characters or delimiters are ignored and passed as data to the application, except for the character pair </ followed immediately by a letter (a - z, A - Z), This means that the end tag of the element (or of an element in which it is nested) is recognized. (Scripts may need to contain e.g. HTML end tags as data. Different scripting languages provide different methods for coping with this.)

Examples

Since there is no semantics defined for the SCRIPT element in HTML 3.2, no meaningful example can be given.

Notes

The SCRIPT element was introduced into HTML 3.2 just a place holder for the introduction of support for scripting languages in future versions of HTML, such as HTML 4.0.

SELECT - menu in a form

Purpose

To specify, within a form, a menu from which the user can select one or more alternatives. Effectively, a SELECT element defined a form field where user's input is restricted to an enumerated list of values.

Typical rendering

A selection menu which can be "activated" in some browser-dependent way; in a typical graphical browser this means a pull-down menu. Depending on the browser and on the element, all alternatives may be visible at the same time or the user may need to scan through the list one at a time.

Basic syntax

<SELECT NAME=*name*>
OPTION elements
</SELECT>

Possible attributes

attribute name	possible values	meaning	notes
NAME	*string*	a property name that is used to identify the menu choice when the form is submitted to the server	obligatory; each selected option results in a name/value pair being included as part of the contents of the form
SIZE	*integer*	sets the number of visible choices	applicable when MULTIPLE is set
MULTIPLE	MULTIPLE	signifies that the user can make multiple selections from the menu	by default only one selection is allowed

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. However, the text container must appear within a FORM element.

Contents

OPTION elements.

Examples

Example:

```
<SELECT NAME="flavor">
<OPTION VALUE=a SELECTED>Vanilla
<OPTION VALUE=b>Strawberry
<OPTION VALUE=c>Rum and Raisin
<OPTION VALUE=d>Peach and Orange
</SELECT>
```

Notes

See the description of the FORM element, which contains some examples of entire forms.

As an alternative to SELECT, you may wish to consider using a set of INPUT elements with TYPE=CHECKBOX or TYPE=RADIO, typically resulting in a rendering which allows the user see all alternatives at a glance.

You should always provide a default choice (an OPTION element with attribute `SELECTED` set) within a `SELECT` element. For reasons to this, see my document *Choices in HTML forms*.

SMALL - small font (Not in HTML 2.0!)

Purpose

To present text in a small font, e.g. in order to indicate it as less important.

Typical rendering

Smaller than normal font. See general notes on rendering markup.

Basic syntax

<SMALL>*text*</SMALL>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

A trivial example:

Example SMALL-1.html:

```
<P>
This is normal text.
</P>
<P>
<SMALL>
This text will be presented in a smaller font, if possible.
</SMALL>
</P>
```

An example which uses SMALL to simulate "small caps" font style.

Example SMALL-2.html:

```
J<SMALL>UKKA</SMALL> K<SMALL>ORPELA</SMALL> has written an HTML
primer G<SMALL>ETTING</SMALL> S<SMALL>TARTED WITH</SMALL> HTML.
```

Especially in a document which contains a lot of abbreviations or other expressions in all-caps, one might use SMALL for those abbrevs to make them look better:

Example SMALL-3.html:

```
<SMALL>HTML</SMALL>, <SMALL>HTTP</SMALL> and <SMALL>WWW</SMALL>
are widely used name-like initialisms. By the way, the reference
spelling of character names in Unicode uses upper case only; e.g. "A" is
officially <SMALL>CAPITAL LATIN LETTER A</SMALL>.
```

Notes

As mentioned in the discussion of phrase elements, there is no logical markup for de-emphasis. The SMALL element, despite being physical markup, might conceivably be used for the purpose.

The use of SMALL to simulate "small caps" as in example 2 above is not particularly effective. Some browsers simply ignore SMALL, leading to an all upper case presentation. In popular browsers, SMALL seems to cause presentation which is just marginally (if at all) smaller than normal font. It is better to use logical markup than to stick presentation conventions designed for traditional forms of publication. For example, use CITE for book titles and other citations. (A user who wants to see them in all caps style might consider using style sheets for the purpose.) Unfortunately there is no logical markup for people’s names in current HTML standard.

It is unspecified what happens if SMALL elements are nested; it might or might not result in using a font which is smaller than you get with a single SMALL.

The FONT element may provide more alternatives for specifying different font sizes.

Notice that people may set the normal text font in their browser to something which is just big enough for them to read. If you use SMALL, the result might be illegibly small.

See general notes on text markup, which provide additional examples.

STRIKE - strike-through text (Not in HTML 2.0!)

Purpose

To present strike-through text.

Typical rendering

Strike-through, i.e. with a horizontal line through the middle of the text. See general notes on rendering markup.

Basic syntax

<STRIKE>*text*</STRIKE>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Excerpt from a bill, where strikeout is used to indicate proposed deletion of text:

Example STRIKE-1.html:

```
"Private agency" means an accredited nonpublic school,
a nonprofit institution of higher education
<STRIKE>eligible for tuition grants</STRIKE>, or a hospital.
```

Notes

In HTML 4.0, the STRIKE element is deprecated .

STRIKE is defined as "font style element", i.e. physical markup. The HTML specification does not say what the *meaning* should be. Typically text is striked out to indicate that a text segment belongs to the original version of a text but has been *deleted* later.

If you use STRIKE in your document, it is advisable to include a note about its meaning. Even if you use it for the "normal" meaning, indicating deletion, you should tell this to your readers, since some of them might view the document with browsers which do not support STRIKE at all (and display text within STRIKE elements as normal text). You might even provide a way of getting different versions of the document, with STRIKE replaced by some other method of presenting deleted text.

See general notes on text markup, which provide additional examples.

The HTML 2.0 specification does not include STRIKE but mentions it as an element which has been "deployed to some extent".

The *HTML 3.2 Reference Specification* warns that ’STRIKE may be phased out in favor of the more concise "S" tag from HTML 3.0’.

STRONG - strong emphasis

Purpose

To emphasize strongly.

Typical rendering

In boldface. If this is impossible, a browser might use e.g. underlining (Lynx does so). See general notes on rendering markup.

Basic syntax

<STRONG>*text*</STRONG>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example STRONG-1.html:

```
For your own safety,
<STRONG>turn the power off before opening the device.</STRONG>
```

Notes

Avoid emphasizing too much; emphasizing everything is tantamount no not emphasizing anything.

The STRONG element involves stronger emphasis than the EM element.

See also general remarks on phrase elements.

STYLE - style sheets (Not in HTML 2.0!)

Purpose

To specify a style sheet to be used when rendering the document.

Typical rendering

Style sheets, if supported by a browser, can affect the rendering in a multitude of ways. On the other hand, the contents of a STYLE element consists of instructions for rendering and should not be displayed by the browser.

Basic syntax

<STYLE>*style info*</STYLE>

Possible attributes

None, according to the *HTML 3.2 Reference Specification*. Notice, however, that various style sheet specifications and proposal involve attributes to STYLE. And in HTML 4.0, the STYLE element is specified so that it has an obligatory TYPE attribute (and some optional attributes).

Allowed context

The head section.

Contents

Style information. The syntax and semantics is to be defined separately.

Technically, these elements are defined with CDATA as the content type. As a result they may contain only SGML characters. All markup characters or delimiters are ignored and passed as data to the application, except for the character pair </ followed immediately by a letter (a - z, A - Z), This means that the end tag of the element (or of an element in which it is nested) is recognized.

It is legal, and recommendable, to use the HTML comment delimiters **<!--** and **-->** around the contents of a STYLE element. The reason is that by doing so you ensure that old browsers (ignorant of STYLE) will not display the contents.

Examples

This example uses a very simple style sheet according to CSS1, to specify that some sans-serif font be used when rendering the document, except for U elements, which are to be rendered in a serif font (in addition to being underlined).

Example STYLE-1.html:

```
<HEAD>
<STYLE><!--
BODY { font-family: sans-serif }
U    { font-family: serif }
--></STYLE>
</HEAD>
<BODY>
Sample text 1.<BR>
<U>Sample text 2.</U>
</BODY>
```

Notes

According to the *HTML 3.2 Reference Specification*, the STYLE element is just a place holder for the introduction of style sheets in future versions of HTML.

SUB - subscript (Not in HTML 2.0!)

Purpose

To present subscripts, which are typically indexes attached to variables.

Typical rendering

Slightly below the normal text level, often so that it the text is vertically centered with respect to normal text baseline, and possibly in smaller font. See general notes on rendering markup.

As a side effect, subscripts often cause lines to be unevenly spaced.

Basic syntax

_{*text*}

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Mathematical usage:

Example SUB-1.html:

```
Let us form the sum of all x<SUB>i</SUB>'s, ie
x<SUB>1</SUB> + x<SUB>2</SUB> + ... + x<SUB>n</SUB>.
```

Usage in chemistry:

Example SUB-2.html:

```
SO<sub>3</sub> + H<sub>2</sub>O -> H<sub>2</sub>SO<sub>4</sub>
```

Using SUB and SUP to affect the presentation of fractions:

Example SUB-3.html:

```
Fractions &frac12; and  &frac14; and &frac34; have their own
symbols in ISO Latin 1. Other fractions like <SUP>2</SUP>/<SUB>3</SUB>
must be essentially presented in linearized notation, although you
can use SUB and SUP to affect the presentation.
```

Notes

There is also an element for superscripts, SUP, but HTML 3.2 provides no general support for mathematical formulas.

Since this element is new, support for it is not universal. Some browsers simply ignore it, displaying e.g. `a<SUB>1</SUB>` as `a1`. And naturally, text-only browsers *cannot* truly support SUB.

Subscripts can be nested. This may, however, result e.g. in rendering inner superscripts in a very small font. Internet Explorer ignores SUB tags after nesting level of two.

See also general notes on text markup.

SUP - superscript (Not in HTML 2.0!)

Purpose

To present superscripts. It its debatable whether this includes e.g. exponents in expressions.

Typical rendering

Slightly above the normal text level and possibly in smaller font. See general notes on rendering markup.

As a side effect, superscripts often cause lines to be unevenly spaced.

Basic syntax

^{*text*}

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Note: Most of the examples here are mathematical. It is debatable whether such use reflects the intentions behind the HTML specification.

Example SUP-1.html:

```
The notation A<SUP>T</SUP> denotes the transpose of A.
```

Example SUP-2.html:

```
Consider the equation
x<SUP>n</SUP> + y<SUP>n</SUP> = z<SUP>n</SUP>.
```

Example SUP-3.html:

```
The expression a<SUP>b<SUP>c</SUP></SUP>
means a<SUP>(b<SUP>c</SUP>)</SUP>.
```

Example SUP-4.html:

```
This example is a text paragraph which contains several
superscripted expressions such as m<SUP>2</SUP> and e<SUP>x</SUP>.
They may affect the visual appearance of the paragraph by
forcing the browser to use different line heights. This
applies in particular to expressions with large and nested
superscripts such as (f(a))<SUP>e<SUP>x<SUP>2y</SUP></SUP></SUP>.
```

Example SUP-5.html:

```
Non-mathematical examples:<BR>
The word "first" can be written as 1<SUP>st</SUP>.<BR>
Foo<SUP>(TM)</SUP> is a trademark of Bar, Inc.<BR>
In French, the word "mademoiselle" is abbreviated M<SUP>lle</SUP>.
```

Notes

Digit 1, 2, or 3 as a superscript is representable in another way, too, since the ISO Latin 1 character set contains characters for them. Example: m² or, using character escape, m².

There is also an element for subscripts, SUB, but HTML 3.2 provides no general support for mathematical formulas.

Since this element is new, support for it is not universal. Some browsers simply ignore it, displaying e.g. `a<SUP>T</SUP>` as `aT`. And naturally, text-only browsers *cannot* truly support SUP.

Superscripts can be nested, as the last example shows. This may, however, result e.g. in rendering inner superscripts in a very small font. Internet Explorer ignores SUP tags after nesting level of two.

See also general notes on text markup.

TABLE - tables (Not in HTML 2.0!)

Purpose

To present information which logically forms a table, i.e. a matrix-like structure.

Typical rendering

More or less tabular but by default with no surrounding border. When a border is requested (with the BORDER attribute), a common approach, introduced by Netscape, renders tables in bas-relief, raised up with the outer border as a bevel, and individual cells inset into this raised surface. Borders around individual cells are only drawn if the cell has explicit content. White space doesn't count for this purpose with the exception of . (Notice that there can be better ways to deal with empty cells than to use ` `.)

A table is generally sized automatically by a browser to fit the contents, but you can also set the table width using the WIDTH attribute.

Basic syntax

<TABLE>
rows of the table (TR elements)
</TABLE>

Possible attributes

attribute name	possible values	meaning	notes
ALIGN	LEFT, CENTER, RIGHT	horizontal alignment *of the entire table*	default is LEFT, but this can be overridden by an enclosing DIV or CENTER element; this attribute is deprecated in HTML 4.0
WIDTH	width specification	width of the entire table	by default, width is determined by a browser to fit the contents
BORDER	*integer*	width of the frame, in pixels	value of 0 (default) means no border; some browsers also accept plain BORDER with the same meaning as BORDER=1
CELLSPACING	*integer*	spacing *between* cells, in pixels	see note below
CELLPADDING	*integer*	spacing (padding), in pixels, between the contents of a cell and the border around a cell.	

Typically the BORDER attribute (with nonzero value) sets the default value of CELLSPACING to 1. This means that by setting a border for the entire table you also set borders of one pixel for the individual cells.

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

Allowed context

Block container.

Contents

One or more TR elements, optionally preceded by a CAPTION element.

Examples

A basic example:

Example TABLE-1.html:

```
<TABLE>
<CAPTION>Areas of the Nordic countries, in sq km</CAPTION>
<TR><TH>Country</TH> <TH>Total area</TH> <TH>Land area</TH>
<TR><TH>Denmark</TH> <TD ALIGN=RIGHT> 43,070 </TD><TD ALIGN=RIGHT> 42,370</TR>
<TR><TH>Finland</TH> <TD ALIGN=RIGHT>337,030 </TD><TD ALIGN=RIGHT>305,470</TR>
<TR><TH>Iceland</TH> <TD ALIGN=RIGHT>103,000 </TD><TD ALIGN=RIGHT>100,250</TR>
<TR><TH>Norway</TH>  <TD ALIGN=RIGHT>324,220 </TD><TD ALIGN=RIGHT>307,860</TR>
<TR><TH>Sweden</TH>  <TD ALIGN=RIGHT>449,964 </TD><TD ALIGN=RIGHT>410,928</TR>
</TABLE>
```

An example of control over presentation style:

Example TABLE-2.html:

```
<TABLE ALIGN=CENTER WIDTH="80%" BORDER=1 CELLSPACING=10 CELLPADDING=3>
<CAPTION>The Nordic countries</CAPTION>
<TR><TD>Denmark</TD> <TD>Finland </TD> <TD>Iceland </TD>
    <TD>Norway </TD> <TD>Sweden </TD> </TR>
</TABLE>
```

Notes

See the discussion of tables, which contains additional examples, too.

Tables can be *nested*. However, nested tables (and large tables in general) can be confusing, and there are implementation deficiencies involved. If you have a large collection of material which *might* be presented as a structure of nested tables, give some thought to the question whether it is *useful* (to your readers) that you do so. Often it pays off to present the material first as a compact overview table, then to accompany it with tables containing details about each part.

When there is normal text before or after a table, it is advisable to end the preceding paragraph with an explicit </P> tag and to begin the following paragraph with an explicit <P> tag. Otherwise the browser (e.g. Netscape) may not render the table with suitable empty vertical space around it.

Be careful. If numbers of cells in different rows do not match (taking COLSPAN attributes into account), the result is most probably a total mess.

The *default alignments* are often unsuitable, especially for numerical tables. Unfortunately there is no way for specifying the default alignment for table cells, except rowwise in the TR element; notice that the ALIGN attribute of a TABLE element specifies the alignment of the entire table and does not affect the default alignments for cells.

Several versions of Netscape do not obey an ALIGN=CENTER attribute in a TABLE element. The common solution is to enclose the entire TABLE element into a CENTER element, but this may cause some problems (see *WDG Web Authoring FAQ* entry *How do I center a table?*).

TD - table data (cell) (Not in HTML 2.0!)

Purpose

To present a data cell in a table.

Typical rendering

A data cell in a table, typically presented using the normal text font (although a browser might conceivably decide to use a smaller font). By default, the data is aligned to the left within the space allocated for the cell by the browser.

Basic syntax

<TD>*data*</TD>

In principle, the end tag </TD> can always be omitted. This is not recommendable, since some browsers (including Netscape) may act incorrectly when the end tag is omitted.

Possible attributes

attribute name	possible values	meaning	notes
NOWRAP	NOWRAP	suppress word wrap	equivalent to using non-breaking spaces, , instead of normal spaces within the contents of the cell; this attribute is deprecated in HTML 4.0
ROWSPAN	*integer*	number of rows spanned by the cell	default is 1
COLSPAN	*integer*	number of columns spanned by the cell	default is 1
ALIGN	LEFT, CENTER, RIGHT	horizontal alignment of data in the cell	default is LEFT or the ALIGN attribute in an enclosing TR element
VALIGN	TOP, MIDDLE, BOTTOM	vertical alignment of data in the cell	overrides a VALIGN attribute in an enclosing TR element
WIDTH	*integer*	suggested width of the cell, in **pixels**	the browser should use the value unless it conflicts with the width requirements for other cells in the same column; although many browsers also support percentage widths in this context, they do it rather inconsistently; and browsers implement even HTML 3.2 conformant WIDTH attributes differently in many cases, especially when you try to make some columns fixed width and other columns occupy the rest of the available total width; this attribute is deprecated in HTML 4.0
HEIGHT	*integer*	suggested height of the cell, in pixels	the browser should use the value unless it conflicts with the height requirements for other cells in the same row; this attribute is deprecated in HTML 4.0

Allowed context

TR element.

Contents

Headings, text elements, block elements, and ADDRESS elements.

Examples

<TD>3.1416</TD>

Notes

See the discussion of tables, which contains additional examples, too.

The TD and TH elements are very similar; in particular, they have the same attributes. The TD element is for data in a table whereas the TH element is for headings of columns or rows in a table. The visible differences are:

- usually TH elements are rendered more prominently than TD elements
- the default alignment is centering for TH, left alignment for TD

It is sometimes a matter of taste whether you use TD or TH especially as regards to the first column (i.e. first element of each row).

Normally you should let browsers select suitable height and width for table cells. If you really need to use WIDTH or HEIGHT attributes, it is best to specify the (same) WIDTH attribute for *all* elements in a column and the (same) HEIGHT attribute for all elements in a row. Some browsers might not honor the requirements otherwise; it is debatable whether this is a bug or a feature.

TEXTAREA - multi-line text input in a form

Purpose

To specify, within a form, an area for multi-line user input.

Typical rendering

An input area which appears as a separate box, possibly having a distinct background color, and usually with some kind of scroll bars for both vertical and horizontal direction.

The area is initialized with the contents of the TEXTAREA element, using monospaced font. The contents is displayed as it is written, similarly to PRE elements.

Basic syntax

<TEXTAREA NAME=*name* ROWS=*m COLS*=*n*>
initial text
</TEXTAREA>

Possible attributes

attribute name	possible values	meaning	notes
NAME	*string*	a property name that is used to identify the textarea field when the form is submitted to the server	obligatory
ROWS	*integer*	number of *visible* text lines	obligatory
COLS	*integer*	number of *visible* width of text, in average character widths	obligatory

A browser should not interpret the ROWS and COLS attributes as restricting the size of the actual input. On the contrary, the browser should provide some means to scroll through the contents of the textarea field when the contents extend the visible area.

A browser may wrap visible text lines to keep long input lines visible without need for scrolling.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. However, the text container must appear within a FORM element.

Contents

A string. Escape sequences are allowed, but no tags are recognized.

The contents is used to initialize the text that is shown in the input field when the document is first loaded.

Examples

```
<TEXTAREA NAME=address ROWS=4 COLS=40>
Your address here ...
</TEXTAREA>
```

Notes

See the description of the FORM element, which contains some examples of entire forms.

For single-line input fields you can use an INPUT element with TYPE=TEXT.

It is recommended in the specifications that browsers canonicalize line endings to CR, LF (ASCII decimal 13, 10) when submitting the contents of the field. However, authors should not rely on this, since not all browsers behave so. The character set for submitted data should be ISO Latin 1, unless the server has previously indicated that it can support alternative character sets.

The HTML specifications do not quite explicitly require that the contents of a TEXTAREA element (specifying the initial value) is to be rendered as it is written with respect to division into lines etc (similarly to PRE elements), but this is clearly the intention.

Typically browsers display the textarea content (both the initial content and the content entered by a user) in a fixed font. Although a TEXTAREA may occur within text level markup, it is thus usually not affected by it.

Browsers do not always honor the ROWS and COLS attributes exactly. Rather often the visible input area is somewhat larger than specified by them.

You cannot use ROWS and COLS attributes to *restrict the size of the actual input*, nor can you do that with other HTML constructs. The script that processes the form can be written so that it takes care of handling excessively large input if needed.

TH - table heading (cell) (Not in HTML 2.0!)

Purpose

To present, within a table, a cell which acts as a (row or column) heading.

Typical rendering

A cell in a table, typically presented using some more prominent font such as boldface. By default, the data is centered within the space allocated for the cell by the browser.

Basic syntax

<TH>*data*</TH>

In principle, the end tag </TH> can always be omitted. This is not recommendable, since some browsers (including Netscape) may act incorrectly when the end tag is omitted.

Possible attributes

attribute name	possible values	meaning	notes
NOWRAP	NOWRAP	suppress word wrap	equivalent to using non-breaking spaces, , instead of normal spaces within the contents of the cell; this attribute is deprecated in HTML 4.0
ROWSPAN	*integer*	number of rows spanned by the cell	default is 1
COLSPAN	*integer*	number of columns spanned by the cell	default is 1
ALIGN	LEFT, CENTER, RIGHT	horizontal alignment of data in the cell	default is CENTER or the ALIGN attribute in an enclosing TR element
VALIGN	TOP, MIDDLE, BOTTOM	vertical alignment of data in the cell	overrides a VALIGN attribute in an enclosing TR element
WIDTH	*integer*	suggested width of the cell, in pixels	the browser should use the value unless it conflicts with the width requirements for other cells in the same column; although many browsers also support percentage widths in this context, they do it rather inconsistently; and browsers implement even HTML 3.2 conformant WIDTH attributes differently in many cases, especially when you try to make some columns fixed width and other columns occupy the rest of the available total width; this attribute is deprecated in HTML 4.0
HEIGHT	*integer*	suggested height of the cell, in pixels	the browser should use the value unless it conflicts with the height requirements for other cells in the same row; this attribute is deprecated in HTML 4.0

Allowed context

TR element.

Contents

Headings, text elements, block elements, and ADDRESS elements.

Examples

<TH>Sum</TH>

Notes

See the discussion of tables, which contains additional examples, too.

The TD and TH elements are very similar; in particular, they have the same attributes. The TD element is for data in a table whereas the TH element is for headings of columns or rows in a table. The visible differences are:

- usually TH elements are rendered more prominently than TD elements
- the default alignment is centering for TH, left alignment for TD

It is sometimes a matter of taste whether you use TD or TH especially as regards to the first column (i.e. first element of each row).

TITLE - "external" title

Purpose

To define the (obligatory) "external" title for the document.

Typical rendering

The title is not displayed as part of the document itself but can stand for or be attached to the document in several contexts. The title can be displayed in a browser’s window caption, search result lists returned by search engines, hotlists defined by users, history lists etc.

Basic syntax

<TITLE>*character sequence*</TITLE>

Possible attributes

None.

Allowed context

The head element, in which exactly one TITLE element must appear.

Contents

Character sequence. Within it, character entities such as < (for <) and ä (for ä) are interpreted. No HTML tags are allowed in a title. Therefore, you cannot use different fonts or emphasis in it.

Example

```
<TITLE>A study of population dynamics</TITLE>
```

Notes

It is important to write a good title especially because search result lists returned by search engines may use the title. For the same reason the title should be descriptive (and appetizing!) even out of context, i.e. when it is the only information available about the document. Avoid titles like *Introduction*.

On the other hand, the title should be relatively short to fit into one line under all reasonable circumstances. The HTML 2.0 specification says that long titles may be truncated and that titles should be *at most 63 characters in length*.

See also general notes about the head section.

Use the H1 or some other heading element to specify the main heading to be displayed as part of the document. Using such a heading at the beginning of a document and using a TITLE element are not alternatives but serve different purposes; both are strongly recommended. The title text and the main heading text may well be identical, but of course they need not.

TR - table row (Not in HTML 2.0!)

Purpose

To present a row in a table.

Typical rendering

A single row in a table.

Basic syntax

<TR>*heading cells (TH elements) and data cells (TD elements)*</TR>

In principle, the end tag </TR> can always be omitted. This is not recommendable, since some browsers (including Netscape) may act incorrectly when the end tag is omitted.

Possible attributes

attribute name	possible values	meaning	notes
ALIGN	LEFT, CENTER, RIGHT	default horizontal alignment in cells	can be overridden by ALIGN attributes in TH and TD elements
VALIGN	TOP, MIDDLE, BOTTOM	default vertical alignment in cells	can be overridden by VALIGN attributes in TH and TD elements

Allowed context

TABLE element.

Contents

TH elements and TD elements.

Examples

```
<TR><TD>3.70 <TD>4.69 <TD>8.02 </TR>
```

Notes

See the discussion of tables, which contains additional examples, too.

TT - teletype (monospaced) text

Purpose

To present text in a monospaced font.

Typical rendering

Monospaced font. See general notes on rendering markup.

Basic syntax

<TT>text</TT>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example TT-1.html:

```
Compare <TT>monospaced font</TT> with normal font.
```

Notes

Avoid using TT; use logical markup instead, e.g. CODE or SAMP.

See general notes on text markup, which provide additional examples.

U - underline (Not in HTML 2.0!)

Purpose

To underline text.

Typical rendering

Underlined. However, e.g. several versions of Netscape still in use present U elements as normal text. See general notes on rendering markup.

Basic syntax

<U>text</U>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example U-1.html:

```
Compare <U>underlined text</U> with normal text.
```

Notes

Avoid using U; use logical markup instead. For example, to **emphasize** use EM or STRONG. In HTML 4.0, the U element is deprecated .

It is customary to use underlining in typewritten text for various other purposes than emphasis, too, but in HTML it is usually better to use e.g. the I element (to produce italics).

One particular reason for avoiding U is that typically Web browsers present links using underlining (instead of or in addition to other methods such as different color). Therefore, if you use U elements, the reader may have serious difficulties in distinguishing them from links.

The HTML 2.0 specification does not include U but mentions it as an element which has been "deployed to some extent".

See general notes on text markup, which provide additional examples.

UL - unnumbered list

Purpose

To present information in a list form (without numbering the items).

Typical rendering

A bulleted list. The list items are presented separately, although possibly with less space between them than there is e.g. between paragraphs. The presentation is often indented in a manner which causes nested lists to be indented according to their structure.

Basic syntax

<UL>
<LI> *list item 1*
<LI> *list item 2*
...
</UL>

Possible attributes

attribute name	possible values	meaning	notes
TYPE	DISC, SQUARE, CIRCLE	default bullet style for items	Not in HTML 2.0!
COMPACT	COMPACT	reduced interim spacing	often ignored by browsers

Both attributes are deprecated in HTML 4.0.

The default value of bullet type generally depends on the level of nesting (various) lists.

Allowed context

Block container.

Contents

LI elements (one or more).

Examples

A simple example:

Example UL-1.html:

```
Remember to buy
<UL>
<LI> milk
<LI> bread
<LI> apples.
</UL>
```

A contrived example to show what the bullets may look like. Notice that TYPE attribute in a LI element overrides that of an enclosing UL element.

Example UL-2.html:

```
<UL TYPE=DISC COMPACT>
<LI> disc
<LI TYPE=SQUARE> square
<LI TYPE=CIRCLE> circle
</UL>
```

See also *Examples of various list elements in HTML.*

Notes

See general notes about list elements for a discussion of selecting between them.

If your list items contain numeric or alphabetic labels like 1, 2, 3, ... or a, b, c, ..., you should use an ordered list, the OL element (and remove those labels, since they are generated by a Web browser when OL is used).

A UL element must contain at least one LI element. Some people and some HTML editors may generate UL elements with just text within, possibly even nesting UL elements just in the hope of getting different amounts of indentation. If you have to resort to such tricks, enclose the text into an LI element (although this will usually cause a bullet in the display) and this in turn into UL. (Style sheets will provide mechanisms for controlling indentation.)

VAR - variables

Purpose

To indicate that a piece of text (typically, a word) is a **variable**, a "placeholder", i.e. a generic notation to be replaced by different actual expressions.

Typical rendering

In italics. (Unfortunately, Internet Explorer (IE) 3.0 renders VAR using monospaced font.) See general notes on rendering markup.

Basic syntax

<VAR>*text*</VAR>

Possible attributes

None.

Allowed context

Text container, i.e. any element that may contain text elements. This includes most HTML elements. In particular, text elements can be nested.

Contents

Text elements. Notice that this disallows e.g. paragraph breaks.

Examples

Example VAR-1.html:

```
In the simplest case, the command for deleting a file in Unix is<BR>
<KBD>rm</KBD> <VAR>filename</VAR>
```

Notes

See notes on presenting interaction with computer and general remarks on phrase elements.