Cascading Style Sheets, level 2

W3C Working Draft *28-Jan-1998*

This version:
http://www.w3.org/TR/1998/WD-css2-19980128
Latest version:
http://www.w3.org/TR/WD-css2
Previous version:
http://www.w3.org/TR/WD-CSS2-971104
Editors:
Bert Bos <bbos@w3.org>
Håkon Wium Lie <howcome@w3.org>
Chris Lilley <chris@w3.org>
Ian Jacobs <ij@w3.org>

Abstract

This specification defines Cascading Style Sheet, level 2 (CSS2). CSS2 is a style sheet language that allows authors and users to attach style (e.g. fonts, spacing and aural cues) to structured documents (e.g. HTML and XML). The CSS2 language is human readable and writable, and expresses style in common desktop publishing terminology.

CSS2 builds on [CSS1] [p. 267] , specified in http://www.w3.org/TR/REC-CSS1-961217. All valid CSS1 style sheets are valid CSS2 style sheets.

Status of this document

This is a W3C Working Draft for review by W3C members and other interested parties. It is a draft document and may be updated, replaced or obsoleted by other documents at any time. It is inappropriate to use W3C Working Drafts as reference material or to cite them as other than "work in progress". This is work in progress and does not imply endorsement by, or the consensus of, either W3C or members of the CSS working group.

This document has been produced as part of the W3C Style Activity, and is intended as a draft of a proposed recommendation for CSS2.

If you did not get this document directly from the W3C website you may want to check whether you have the latest version of this document by looking at the list of W3C technical reports at <http://www.w3.org/TR/>.

Available formats

The CSS2 specification is also available in the following formats:

a plain text file:
http://www.w3.org/TR/1998/WD-css2-19980128/css20.txt,
HTML as a gzip'ed tar file:
http://www.w3.org/TR/1998/WD-css2-19980128/css20.tgz,
HTML as a zip file (this is a '.zip' file not an '.exe'):
http://www.w3.org/TR/1998/WD-css2-19980128/css20.zip,
as well as a postscript file:
http://www.w3.org/TR/1998/WD-css2-19980128/css20.ps,
and a PDF file:
http://www.w3.org/TR/1998/WD-css2-19980128/css20.pdf.

In case of a discrepancy between electronic and printed forms of the specification, the electronic version is considered the definitive version.

Available languages

The English version of this specification is the only normative version. However, for translations in other languages see
http://www.w3.org/TR/1998/WD-css2-19980128/translations.html.

Comments

Please send detailed comments on this document to the editors. We cannot guarantee a personal response but we will try when it is appropriate. Public discussion on CSS features takes place on www-style@w3.org and messages are archived at http://lists.w3.org/Archives/Public/www-style/.

Quick Table of Contents

1. About the CSS2 Specification [p. 13]
2. Introduction to CSS2 [p. 19]
3. Conformance: Requirements and Recommendations [p. 25]
4. CSS2 syntax and basic data types [p. 29]
5. Selectors [p. 43]
6. Assigning property values, Cascading, and Inheritance [p. 57]
7. Media types [p. 63]
8. Visual rendering model [p. 67]
9. Visual rendering model details [p. 99]
10. Visual effects [p. 123]
11. Generated content and automatic numbering [p. 129]
12. Paged media [p. 131]
13. Colors and Backgrounds [p. 141]
14. Fonts [p. 147]
15. Text [p. 185]
16. Lists [p. 193]
17. Tables [p. 201]
18. User interface [p. 227]
19. Aural style sheets [p. 231]

A. Appendix A: A sample style sheet for HTML 4.0 [p. 245]
B. Appendix B: Changes from CSS1 [p. 249]
C. Appendix C: Implementation and performance notes [p. 251]
D. Appendix D: The grammar of CSS2 [p. 263]

- References [p. 267]
- Property index [p. 271]
- Descriptor index [p. 277]
- Index [p. 279]

Full Table of Contents

1. About the CSS2 Specification [p. 13]
    1. Reading the specification [p. 13]
    2. How the specification is organized [p. 13]
    3. Conventions [p. 14]
        1. Document language [p. 25] elements and attributes
        2. CSS property definitions [p. 14]
            1. Shorthand properties [p. 16]
        3. Notes and examples [p. 16]
    4. Acknowledgments [p. 16]
    5. Copyright Notice [p. 17]
2. Introduction to CSS2 [p. 19]
    1. A brief CSS2 tutorial for HTML [p. 19]
    2. A brief CSS2 tutorial for XML [p. 20]
    3. The CSS2 processing model [p. 21]
        1. The canvas [p. 22]
        2. CSS2 addressing model [p. 22]
    4. CSS design principles [p. 22]
3. Conformance: Requirements and Recommendations [p. 25]
    1. Definitions [p. 25]
    2. Conformance [p. 27]
    3. Error conditions [p. 28]
4. CSS2 syntax and basic data types [p. 29]
    1. Syntax [p. 29]
        1. Tokenization [p. 29]
        2. Characters and case [p. 31]
        3. Statements [p. 32]
        4. At-rules [p. 32]
        5. Blocks [p. 33]
        6. Rule sets, declaration blocks, and selectors [p. 33]
        7. Declarations [p. 34] and properties
        8. Comments [p. 35]
    2. Rules for handling parsing errors [p. 35]
    3. Values [p. 36]
        1. Integers and real numbers [p. 36]
        2. Lengths [p. 36]
        3. Percentages [p. 38]
        4. URIs [p. 38]
        5. Colors [p. 39]
        6. Angles [p. 40]
        7. Times [p. 40]
        8. Frequencies [p. 40]
        9. Strings [p. 41]
    4. CSS embedded in HTML [p. 41]
    5. CSS as a stand-alone file [p. 41]
    6. Character escapes in CSS [p. 41]

5. Selectors [p. 43]
    1. Pattern matching [p. 43]
    2. Universal selector [p. 44]
    3. Type selectors [p. 45]
    4. Descendant selectors [p. 45]
    5. Child selectors [p. 46]
        1. :first-child [p. 46] pseudo-class
    6. Adjacent selectors [p. 47]
    7. Attribute selectors [p. 47]
        1. Matching attributes and attribute values [p. 47]
            1. Reusing the value of an attribute [p. 48]
        2. The "class" attribute in HTML [p. 48]
    8. ID selectors [p. 49]
    9. Grouping [p. 50]
    10. Pseudo-elements [p. 50] and pseudo-classes
        1. The :first-line [p. 51] pseudo-element
        2. The :first-letter pseudo-element [p. 52]
        3. The :before and :after pseudo-elements [p. 54]
        4. Pseudo-elements with descendant selectors [p. 54]
        5. The Anchor pseudo-classes [p. 54] : :link, :visited, :hover, and :active
        6. Combining pseudo-elements with attribute selectors [p. 55]
6. Assigning property values, Cascading, and Inheritance [p. 57]
    1. Specified, computed, and absolute values [p. 57]
        1. Specified values [p. 57]
        2. Computed values [p. 57]
        3. Actual values [p. 58]
    2. Inheritance [p. 58]
        1. The inherit [p. 59] value
    3. The cascade [p. 59]
        1. Cascading order [p. 60]
        2. 'Important' rules [p. 60]
        3. Cascading order in HTML [p. 61]
        4. Precedence of non-CSS presentational hints [p. 61]
7. Media types [p. 63]
    1. Introduction to media types [p. 63]
    2. Specifying media-dependent style sheets [p. 63]
        1. The @media rule [p. 64]
        2. The media-dependent @import rule [p. 64]
    3. Recognized media types [p. 64]
        1. Media groups [p. 65]
8. Visual rendering model [p. 67]
    1. Introduction to the visual rendering model [p. 67]
        1. The viewport [p. 68]
    2. The box model [p. 68]
        1. Controlling box generation: [p. 68] the 'display' property
            1. Compact and run-in boxes [p. 70]

    2. Box dimensions [p. 71]
    3. Example of margins, padding, and borders [p. 73]
3. Positioning schemes [p. 76]
    1. Choosing a positioning scheme: [p. 76] ’position’ property
    2. Box offsets [p. 77] : ’top’, ’right’, ’bottom’, ’left’
4. Normal flow [p. 79]
    1. Anonymous boxes [p. 79]
    2. Block formatting context [p. 79]
    3. Inline formatting context [p. 79]
    4. Direction of inline flow [p. 81] : the ’direction’ property
    5. Relative positioning [p. 82]
5. Floats [p. 82] : the ’float’ and ’clear’ properties
    1. Controlling flow next to floats [p. 84]
6. Absolute positioning [p. 85]
    1. Fixed positioning [p. 85]
7. Relationships between ’display’, ’position’, and ’float’ [p. 87]
8. Comparison of normal, relative, floating, absolute positioning [p. 87]
    1. Normal flow [p. 88]
    2. Relative positioning [p. 88]
    3. Floating a box [p. 89]
    4. Absolute positioning [p. 92]
9. Z-order [p. 96] : Layered presentation
    1. Specifying the stack level [p. 96] : the ’z-index’ property

9. Visual rendering model details [p. 99]
    1. Margin, border, and padding properties [p. 99]
        1. Margin properties [p. 99] : ’margin-top’, ’margin-right’, ’margin-bottom’, ’margin-left’, and ’margin’
        2. Padding properties [p. 102] : ’padding-top’, ’padding-right’, ’padding-bottom’, ’padding-left’, and ’padding’
        3. Border properties [p. 104]
            1. Border width [p. 104] : ’border-top-width’, ’border-right-width’, ’border-bottom-width’, ’border-left-width’, and ’border-width’
            2. Border color [p. 106] : ’border-top-color’, ’border-right-color’, ’border-bottom-color’, ’border-left-color’, and ’border-color’
            3. Border style [p. 107] : ’border-top-style’, ’border-right-style’, ’border-bottom-style’, ’border-left-style’, and ’border-style’
            4. Border shorthand properties [p. 109] : ’border-top’, ’border-bottom’, ’border-right’, ’border-left’, and ’border’
    2. Containing blocks [p. 111]
    3. Box width calculations [p. 112]
        1. Content width [p. 112] : the ’width’ property
        2. Widths of boxes in the normal flow and floated boxes [p. 113]
            1. Determining the content width [p. 113]
            2. Computing margin widths [p. 114]
        3. Width of absolutely positioned elements [p. 114]
        4. Minimum and maximum widths [p. 114] : ’min-width’ and ’max-width’

    4. Box height calculations [p. 115]
        1. Content height [p. 115] : the ’height’ property
        2. Determining the content height [p. 116]
        3. Height of absolutely positioned elements [p. 116]
        4. Minimum and maximum heights [p. 116] : ’min-height’ and ’max-height’
    5. Collapsing margins [p. 117]
    6. Line height calculations [p. 118] : the ’line-height’ and ’vertical-align’ properties
        1. Leading and half-leading [p. 118]
    7. Floating constraints [p. 121]
10. Visual effects [p. 123]
    1. Overflow and clipping [p. 123]
        1. Overflow [p. 123] : the ’overflow’ property
        2. Clipping [p. 125] : the ’clip’ property
    2. Visibility [p. 127] : the ’visibility’ property
11. Generated content and automatic numbering [p. 129]
    1. The [p. 129] :before and :after pseudo elements and the ’content’ property
    2. Automatic counters and numbering [p. 130]
12. Paged media [p. 131]
    1. Introduction to paged media [p. 131]
    2. Page boxes [p. 132] : the @page rule
        1. Page margins [p. 132]
        2. Page size [p. 133] : the ’size’ property
            1. Rendering page boxes that do not fit a target sheet [p. 134]
            2. Positioning the page box on the sheet [p. 135]
        3. Crop marks [p. 135] : the ’marks’ property
        4. Left, right, and first pages [p. 135]
        5. Content outside the page box [p. 136]
    3. Page breaks [p. 137]
        1. Page break properties [p. 137] : ’page-break-before’, ’page-break-after’, ’orphans’, and ’widows’
        2. Allowed page breaks [p. 138]
        3. Forced page breaks [p. 139]
        4. "Best" page breaks [p. 139]
    4. Cascading in the page context [p. 140]
13. Colors and Backgrounds [p. 141]
    1. Foreground color [p. 141] : the ’color’ property
    2. The background [p. 141]
        1. Background properties [p. 142] : ’background-color’, ’background-image’, ’background-repeat’, ’background-attachment’, ’background-position’, and ’background’
14. Fonts [p. 147]
    1. Introduction [p. 148]
    2. Font specification [p. 149]
        1. Font specification properties [p. 149]

2. Font family [p. 150] : the ’font-family’
3. Font style [p. 151] : the ’font-style’, ’font-variant’, and ’font-weight’ properties
4. Font size [p. 154] : the ’font-size’ and ’font-size-adjust’ properties
5. Shorthand font property [p. 158] : the ’font’ property
6. Generic font families [p. 159]
    1. serif [p. 160]
    2. sans-serif [p. 160]
    3. cursive [p. 161]
    4. fantasy [p. 161]
    5. monospace [p. 161]

3. Font selection [p. 162]
    1. Font Descriptions and @font-face [p. 163]
    2. Descriptors for Selecting a Font [p. 165] : ’font-family’, ’font-style’, ’font-variant’, ’font-weight’, and ’font-size’
    3. Descriptors for Font Data Qualification [p. 166] : ’unicode-range’
    4. Descriptor for Numeric Values [p. 168] : ’units-per-em’
    5. Descriptor for Referencing [p. 168] : ’src’
    6. Descriptors for Matching [p. 169] : ’panose-1’, ’stemv’, ’stemh’, ’slope’, ’cap-height’, ’x-height’, ’ascent’, and ’descent’
    7. Descriptors for Synthesis [p. 171] : ’widths’ and ’definition-src’
    8. Descriptors for Alignment [p. 172] : ’baseline’, ’centerline’, ’mathline’, and ’topline’
4. Font Characteristics [p. 173]
    1. Introducing Font Characteristics [p. 173]
    2. Adorned font name [p. 174]
    3. Central Baseline [p. 175]
    4. Co-ordinate units on the em square [p. 175]
    5. Font encoding tables [p. 176]
    6. Font family name [p. 176]
    7. Glyph Representation widths [p. 176]
    8. Horizontal stem width [p. 176]
    9. Height of capital glyph representations [p. 176]
    10. Height of lowercase glyph representations [p. 177]
    11. Lower Baseline [p. 177]
    12. Mathematical Baseline [p. 178]
    13. Maximal bounding box [p. 178]
    14. Maximum unaccented height [p. 178]
    15. Maximum unaccented depth [p. 178]
    16. Panose-1 number [p. 178]
    17. Range of ISO10646characters [p. 179]
    18. Top Baseline [p. 180]
    19. Vertical stem width [p. 180]
    20. Vertical stroke angle [p. 180]
5. Font matching algorithm [p. 180]
    1. Examples of font matching [p. 182]

15. Text [p. 185]

1. Indentation [p. 185] : the ’text-indent’ property
2. Alignment [p. 186] the ’text-align’ property
3. Decoration [p. 186]
    1. Underlining, over lining, striking, and blinking [p. 186] : the ’text-decoration’ property
    2. Text shadows [p. 187] : the ’text-shadow’ property
4. Letter and word spacing [p. 189] : the ’letter-spacing’ and ’word-spacing’ properties
5. Case [p. 190]
    1. Capitalization [p. 190] : the ’text-transform’ property
    2. Special first letter/first line [p. 191]
6. White space [p. 191] : the ’white-space’ property
7. Text in HTML [p. 192]
    1. Forcing a line break [p. 192]

16. Lists [p. 193]
    1. Visual formatting of lists [p. 193]
        1. List properties [p. 195] : ’list-style-type’, ’list-style-image’, ’list-style-position’, and ’list-style’

17. Tables [p. 201]
    1. Building visual tables [p. 202]
        1. The ’display’ property [p. 202]
        2. Cell spanning properties: [p. 204] ’column-span’, and ’row-span’
    2. Layout model of elements inside tables [p. 204]
        1. Margins on rows, columns and cells [p. 205]
        2. Interactions between rows and columns [p. 205]
            1. Labeled diagram: [p. 206]
        3. The ’border-collapse’ property [p. 207]
        4. The border styles [p. 209]
    3. Computing widths and heights [p. 210]
        1. The ’table-layout’ property [p. 210]
        2. The ’collapse’ value for the ’visibility’ property [p. 211]
    4. ’Vertical-align’ on table cells [p. 211]
    5. Table elements in selectors [p. 212]
        1. Columns and cell selectors [p. 212]
        2. Row, column and cell pseudo-classes [p. 213]
            1. Row pseudo-classes: [p. 213]
            2. Column pseudo-classes: [p. 213]
    6. HTML 4.0 default table stylesheet [p. 213]
    7. UNDER CONSTRUCTION! [p. 214] *-material yet to be integrated, substituted, or discarded*
        1. Table layout [p. 214]
        2. Computing widths and heights [p. 216]
        3. Placement of the borders: [p. 216] ’cell-spacing’
        4. Conflict resolution for borders [p. 218]
        5. Properties for columns and rows [p. 220]
        6. Vertical alignment of cells in a row [p. 220]
        7. Horizontal alignment of cells in a column [p. 222]

8. Table captions [p. 222] : the ’caption-side’ property
9. Generating speech [p. 222] : the ’speak-header’ property
10. Table implementation notes [p. 225]

18. User interface [p. 227]
    1. Cursors: [p. 227] the ’cursor’ property
    2. User preferences for colors [p. 228]
    3. User preferences for fonts [p. 230]
    4. Other rendering issues that depend on user agents [p. 230]
        1. Magnification [p. 230]
19. Aural style sheets [p. 231]
    1. Introduction to aural style sheets [p. 231]
    2. Volume properties [p. 231] : ’volume’
    3. Speaking properties [p. 232] : ’speak’
    4. Pause properties [p. 233] : ’pause-before’, ’pause-after’, and ’pause’
    5. Cue properties [p. 234] : ’cue-before’, ’cue-after’, and ’cue’
    6. Mixing properties [p. 236] : ’play-during’
    7. Spatial properties [p. 237] : ’azimuth’ and ’elevation’
    8. Voice characteristic properties [p. 239] : ’speech-rate’, ’voice-family’, ’pitch’, ’pitch-range’, ’stress’, and ’richness’
    9. Speech properties [p. 242] : ’speak-punctuation’, ’speak-date’, ’speak-numeral’, and ’speak-time’

A. Appendix A: A sample style sheet for HTML 4.0 [p. 245]
B. Appendix B: Changes from CSS1 [p. 249]
    1. New functionality [p. 249]
    2. Updated descriptions [p. 249]
    3. Changes [p. 249]
C. Appendix C: Implementation and performance notes [p. 251]
    1. Order of property value calculation [p. 251]
    2. Colors [p. 251]
        1. Gamma Correction [p. 251]
    3. Fonts [p. 252]
        1. Glossary of font terms [p. 252]
        2. Font retrieval [p. 256]
        3. Meaning of the Panose Digits [p. 256]
        4. Deducing Unicode Ranges for TrueType [p. 259]
D. Appendix D: The grammar of CSS2 [p. 263]
    1. Grammar [p. 263]
    2. Lexical scanner [p. 265]

- References [p. 267]
    - Normative references [p. 267]
    - Informative references [p. 268]
- Property index [p. 271]
- Descriptor index [p. 277]
- Index [p. 279]

About the CSS2 Specification

1.1 Reading the specification

This specification has been written with two types of readers in mind: CSS authors and CSS implementors. We hope the specification will provide authors with the tools they need to write efficient, attractive, and accessible documents, without overexposing them to CSS's implementation details. Implementors, however, should find all they need to build conforming user agents [p. 27] . The specification begins with a general presentation of CSS and becomes more and more technical and specific towards the end. For quick access to information, a general table of contents, specific tables of contents at the beginning of each section, and an index provide easy navigation, in both the electronic and printed versions.

The specification has been written with two modes of presentation in mind: electronic and printed. Although the two presentations will no doubt be similar, readers will find some differences. For example, links will not work in the printed version (obviously), and page numbers will not appear in the electronic version. In case of a discrepancy, the electronic version is considered the authoritative version of the document.

1.2 How the specification is organized

The specification is organized into the following sections:

Section 2: An introduction to CSS2
: The introduction includes a brief tutorial on CSS2 and a discussion of design principles behind CSS2.

Sections 3 - 19: CSS2 reference manual.
: The bulk of the reference manual consists of the CSS2 language reference. This reference defines what may go into a CSS2 style sheet (syntax, properties, property values) and how user agents must interpret these style sheets in order to claim conformance [p. 27] .

Appendixes:
: Five appendixes contain information about a sample style sheet for HTML 4.0 [p. 245] , changes from CSS1 [p. 249] , implementation and performance notes [p. 251] , and the grammar of CSS2 [p. 263] .

References:
: A list of normative and informative references.

General index:
: The general index [p. 279] contains links to key concepts, property and value definitions, and other useful information.

1.3 Conventions

1.3.1 Document language elements and attributes

- CSS property, descriptor, and pseudo-class names are delimited by single quotes.
- CSS values are delimited by single quotes.
- Document language element names are in upper case letters.
- Document language attribute names are in lower case letters and delimited by double quotes.

1.3.2 CSS property definitions

Each CSS property definition begins with a summary of key information that resembles the following:

'Property-name'

Value:	Possible constant values or value types
Initial:	The initial value
Applies to:	Elements this property applies to
Inherited:	Whether the property is inherited
Percentage values:	How percentage values should be interpreted
Media groups:	Which media groups this property applies to

The categories have the following meanings:

Value

This part of the property definition specifies the set of valid values for the property. Value types may be designated in several ways:

1. constant values (e.g., 'auto', 'disc', etc.)
2. basic data types, which appear between "<" and ">" (e.g., <length>, <percentage>, etc.). In the electronic version of the document, each instance of a basic data type links to its definition.
3. non-terminals that have the same range of values as a property bearing the same name (e.g., <'border-width'> <'background-attachment'>, etc.). In this case, the non-terminal name is the property name (complete with quotes) between "<" and ">" (e.g., <'border-width'>). In the electronic version of the document, each instance of this type of non-terminal links to the corresponding property definition.
4. non-terminals that do not share the same name as a property. In this case, the non-terminal name appears between "<" and ">" (e.g., <border-width>) and its definition is located near its first appearance in the specification. In the electronic version of the document, each instance of this type of non-terminal links to the corresponding value definition.

Other words in these definitions are keywords that must appear literally, without quotes (e.g., red). The slash (/) and the comma (,) must also appear literally.

Values may be arranged as follows:

- Several juxtaposed words mean that all of them must occur, in the given order.
- A bar (|) separates alternatives: one of them must occur.
- A double bar (A || B) means that either A or B or both must occur, in any order.
- Brackets ([]) are for grouping.

Juxtaposition is stronger than the double bar, and the double bar is stronger than the bar. Thus, the following lines are equivalent:

```
  a b   |   c || d e
[ a b ] | [ c || [ d e ]]
```

Every type, keyword, or bracketed group may be followed by one of the following modifiers:

- An asterisk (*) indicates that the preceding type, word or group is repeated zero or more times.
- A plus (+) indicates that the preceding type, word or group is repeated one or more times.
- A question mark (?) indicates that the preceding type, word or group is optional.
- A pair of numbers in curly braces ({A,B}) indicates that the preceding type, word or group is repeated at least A and at most B times.

The following examples illustrate different value types:

Value: N | NW | NE
Value: [<length> | thick | thin]{1,4}
Value: [<family-name> ,]* <family-name>
Value: <uri>? <color> [/ <color>]?
Value: <uri> || <color>

Initial

The property's initial value. If the property is inherited, this is the value that is given to the root element of the document tree [p. 26] . Please consult the section on the cascade [p. 57] for information about the interaction between style sheet-specified, inherited, and initial values.

Applies to

Lists the elements to which the property applies. All elements are considered to have all properties, but some properties have no rendering effect on some types of elements. For example, 'white-space' has no effect on inline elements.

Inherited

Indicates whether the value of the property is inherited from a ancestor element. Please consult the section on the cascade [p. 57] for information about the interaction between style sheet-specified, inherited, and initial values.

Percentage values

Indicates how percentages should be interpreted, if they occur in the value of the property. If "N/A" appears here, it means that the property does not accept percentages as values.

Media groups

Indicates the media groups [p. 65] to which the property applies. The conformance [p. 25] conditions state that user agents must support this property if they support rendering to the media types [p. 64] included in these media groups [p. 65] .

1.3.2.1 Shorthand properties

Some properties are *shorthand* rules that allow authors to specify the values of several properties with a single property.

For instance, the ’font’ property is a shorthand property for setting ’font-style’, ’font-variant’, ’font-weight’, ’font-size’, ’line-height’, and ’font-family’ all at once.

When values are omitted from a shorthand form, each "missing" property either inherits its value or is assigned its initial value (see the section on the cascade [p. 57]).

The multiple style rules of the previous example:

```
H1 {
  font-weight: bold;
  font-size: 12pt;
  line-height: 14pt;
  font-family: Helvetica;
  font-variant: normal;
  font-style: normal;
}
```

may be rewritten with a single shorthand property:

```
H1 { font: bold 12pt/14pt Helvetica }
```

In this example, ’font-variant’ and ’font-style’ have been omitted from the shorthand form, so they must be assigned either an inherited or initial value.

1.3.3 Notes and examples

All examples illustrating deprecated usage are marked as "DEPRECATED EXAMPLE". Deprecated examples also include recommended alternate solutions. All examples that illustrate illegal usage are clearly marked as "ILLEGAL EXAMPLE".

All [HTML40] [p. 268] examples conform to the strict DTD unless otherwise indicated by a document type declaration.

Examples and notes are marked within the source HTML for the specification and some user agents may render them specially.

1.4 Acknowledgments

This specification is the product of the W3C Working Group on Cascading Style Sheets and Formatting Properties. In addition to the editors of this specification, the members of the Working Group are: Brad Chase (Bitstream), Chris Wilson (Microsoft), Daniel Glazman (Electricité de France), Dave Raggett (W3C/HP), Ed Tecot (Microsoft), Jared Sorensen (Novell), Lauren Wood (SoftQuad), Laurie Anna Kaplan (Microsoft), Mike Wexler (Adobe), Murray Maloney (Grif), Powell Smith (IBM), Robert Stevahn (HP), Steve Byrne (JavaSoft), Steven Pemberton (CWI), and Thom Phillabaum (Netscape). We thank them for their continued

efforts.

A number of invited experts to the Working Group have contributed: George Kersher, Glenn Rippel (Bitstream), Jeff Veen (HotWired), Markku T. Hakkinen (The Productivity Works), Martin Dürst (Universität Zürich), Roy Platon (RAL), Todd Fahrner (Verso) and Vincent Quint (W3C).

The section on Web Fonts was strongly shaped by Brad Chase (Bitstream) David Meltzer (Microsoft Typography) and Steve Zilles (Adobe). The following people have also contributed in various ways to the section pertaining to fonts: Alex Beamon (Apple), Ashok Saxena (Adobe), Ben Bauermeister (HP), Dave Raggett (W3C/HP), David Opstad (Apple), David Goldsmith (Apple), Ed Tecot (Microsoft), Erik van Blokland (LettError), François Yergeau (Alis), Gavin Nicol (Inso), Herbert van Zijl (Elsevier), Liam Quin, Misha Wolf (Reuters), Paul Haeberli (SGI), and the late Phil Karlton (Netscape).

The section on Paged Media was in large parts authored by Robert Stevahn (HP) and Stephen Waters (Microsoft).

Robert Stevahn (HP), Scott Furman (Netscape), and Scott Isaacs (Microsoft) were key contributors to CSS Positioning.

Mike Wexler (Adobe) was the editor of the interim Working Draft which described many of the new features of CSS2.

T.V.Raman (Adobe) made pivotal contributions towards Aural Cascading Style Sheets and the concepts of aural presentation.

Todd Fahrner (Verso) researched contemporary and historical browsers to develop the sample style sheet in the appendix.

Thanks to Jan Kärrman, author of html2ps for helping so much in creating the Postscript version of the specification.

Through electronic and physical encounters, the following people have contributed to the development of CSS2: James Clark, Dan Connolly, Douglas Rand, Sho Kuwamoto, Donna Converse, Scott Isaacs, Lou Montulli, Henrik Frystyk Nielsen, Jacob Nielsen, Vincent Mallet, Philippe Le Hégaret, William Perry, David Siegel, Al Gilman, Jason White, Daniel Dardailler, Eva von Pepel, Robert Cailliau.

The discussions on www-style@w3.org have been influential in many key issues for CSS. Especially, we would like to thank Bjorn Backlund, Todd Fahrner, MegaZone, Eric Meyer, Brian Wilson, Lars Marius Garshol, David Perrell, Liam Quinn, Neil St. Laurent, Andrew Marshall, Douglas Rand and Chris Wilson for their participation.

Special thanks to Arnaud Le Hors, whose engineering contributions made this document work.

Lastly, thanks to Tim Berners-Lee without whom none of this would have been possible.

1.5 Copyright Notice

2 Introduction to CSS2

Contents

1. A brief CSS2 tutorial for HTML [p. 19]
2. A brief CSS2 tutorial for XML [p. 20]
3. The CSS2 processing model [p. 21]
    1. The canvas [p. 22]
    2. CSS2 addressing model [p. 22]
4. CSS design principles [p. 22]

2.1 A brief CSS2 tutorial for HTML

In this tutorial, we show how easy it can be to design simple style sheets. For this tutorial, you will need to know a little [HTML40] [p. 268] and some basic desktop publishing terminology.

We begin with a small HTML document:

```
<HTML>
  <TITLE>Bach’s home page</TITLE>
  <BODY>
    <H1>Bach’s home page</H1>
    <P>Johann Sebastian Bach was a prolific composer.
  </BODY>
</HTML>
```

To set the text color of the H1 elements to blue, you can write the following CSS rule:

```
  H1 { color: blue }
```

A CSS rule consists of two main parts: selector [p. 43] (’H1’) and declaration (’color: blue’). The declaration has two parts: property (’color’) and value (’blue’). While the example above tries to influence only one of the properties needed for rendering an HTML document, it qualifies as a style sheet on its own. Combined with other style sheets (one fundamental feature of CSS is that style sheets are combined) it will determine the final presentation of the document.

The [HTML40] [p. 268] specification defines how style sheet rules may be specified for HTML documents: either within the HTML document, or via an external style sheet. To put the style sheet into the document, use the STYLE element:

```
<HTML>
  <TITLE>Bach’s home page</TITLE>
  <STYLE type="text/css">
    H1 { color: blue }
  </STYLE>
  <BODY>
    <H1>Bach’s home page</H1>
    <P>Johann Sebastian Bach was a prolific composer.
  </BODY>
</HTML>
```

For maximum flexibility, we recommend that authors specify external style sheets; they may be changed without modifying the source HTML document, and they may be shared among several documents. To link to an external style sheet, you can use the LINK element:

```
<HTML>
  <TITLE>Bach’s home page</TITLE>
  <LINK rel="stylesheet" href="bach.css" type="text/css">
  <BODY>
    <H1>Bach’s home page</H1>
    <P>Johann Sebastian Bach was a prolific composer.
  </BODY>
</HTML>
```

The LINK element specifies:

- the type of link: to a "stylesheet".
- the location of the style sheet via the href attribute.
- the type of style sheet being linked: "text/css".

2.2 A brief CSS2 tutorial for XML

CSS can be used with any structured document format, for example [XML]. In fact, XML depends more on style sheets than HTML since authors can make up their own elements which user agents don’t know how to display.

Here is a simple XML fragment:

```
<ARTICLE>
  <HEADLINE>Fredrick the Great meets Bach</HEADLINE>
  <AUTHOR>Johann Nikolaus Forkel</AUTHOR>
  <LOCATION>Potsdam</LOCATION>
  <PARA>
    One evening, just as he was getting his
    <INSTRUMENT>flute</INSTRUMENT> ready and his
    musicians were assembled, an officer brought him a list of
    the strangers who had arrived.
  </PARA>
</ARTICLE>
```

To display this fragment in a document-like fashion, we must first declare which elements are inline (i.e., do not cause line breaks) and which are block-level (i.e., cause line breaks).

```
INSTRUMENT { display: inline }
ARTICLE, HEADLINE, AUTHOR, LOCATION, PARA { display: block }
```

The first rule declares INSTRUMENT to be inline, and the second rule, with its comma-separated list of selectors, declares all the other elements to be block-level.

It’s not yet clear how style sheets will be linked to XML documents, but assuming the above CSS fragment is combined with the XML fragment, a visual user agent could format the result as:

```
Fredrick the Great meets Bach
Johann Nikolaus Forkel
Potsdam
One evening, just as he was getting his flute ready
and his musicians were assembled, an officer brought
him a list of the strangers who had arrived.
```

Notice that the word "flute" remains within the paragraph since it is the content of the inline element INSTRUMENT.

Still, the text isn't formatted the way you would expect. For example, the headline font size should be larger than then rest of the text, and you may want to display the author's name in italic:

```
INSTRUMENT { display: inline }
ARTICLE, HEADLINE, AUTHOR, LOCATION, PARA { display: block }
HEADLINE { font-size: 1.5em }
AUTHOR { font-style: italic }
```

[add image, description]

Adding more rules to the style sheet will allow you to further improve the presentation of the document.

2.3 The CSS2 processing model

This section presents one possible model of how user agents that support CSS work. This is only a conceptual model; real implementations may vary.

In this model, a user agent processes a source by going through the following steps:

1. Parse the source document and create a document tree [p. 26] from the source document.
2. Identify the target media type [p. 63] .
3. Retrieve all style sheets associated with the document that are specified for the target media type [p. 63] .
4. Annotate every node of the document tree by assigning a single value to every property [p. 34] that is applicable to the target media type [p. 63] . Properties are assigned values according to the mechanisms described in the section on cascading and inheritance [p. 57] .

   Part of the calculation of values depends on the formatting algorithm appropriate for the target media type [p. 63] . For example, if the target medium is the screen, user agents apply the visual rendering model [p. 67] . If the destination medium is the printed page, user agents apply the page model [p. 131] . If the destination medium is an aural rendering device (e.g., speech synthesizer), user agents apply the aural rendering model [p. 231] .
5. From the annotated document tree, generate a *rendering structure*. The rendering structure may differ significantly from the document tree. First , the rendering structure need not be "tree-shaped" at all -- the nature of the structure depends on the implementation. Second the rendering structure may contain more or less information than the document tree. For instance, if an element in the document tree has a value of 'none' for the 'display' property, that element will generate nothing in the rendering structure. A list element, on the other hand, may generate more information in the rendering structure: the list element's content and list style information (e.g., a bullet

image).

6. Transfer the rendering structure to the target medium (e.g., print the results, display them on the screen, render text as speech, etc.).

Step 1 lies outside the scope of this specification (see, for example, [DOM] [p. 268]).

Steps 2-5 are addressed by the bulk of this specification.

Step 6 lies outside the scope of this specification.

2.3.1 The canvas

For all media, the term *canvas* describes "the space where the rendering structure is rendered." The canvas is infinite for each dimension of the space, but rendering generally occurs within a finite region of the canvas, established by the user agent according to the target medium. For instance, user agents rendering to a screen generally impose a minimum width and choose an initial width based on the dimensions of the viewport [p. 68] . User agents rendering to a page generally impose width and height constraints. Aural user agents may impose limits in audio space, but not in time.

2.3.2 CSS2 addressing model

CSS2 selectors [p. 43] and properties allow authors to refer to the following "entities" from within a style sheet:

- Elements in the document tree and certain relationships between them (see the section on selectors [p. 43]).
- Attributes of elements in the document tree, and values of those attributes (see the section on attribute selectors [p. 47]).
- Some parts of element content (see the :first-line [p. 52] and :first-letter [p. 52] pseudo-elements.
- Elements of the document tree when they are in a certain state (see the section on pseudo-classes [p. 50]).
- Some aspects of the canvas [p. 22] where the document will be rendered.
- Some system information (see the section on user interface [p. 227]).

2.4 CSS design principles

CSS2, as CSS1 before it, is based on set of design principles:

- **Backward compatibility**. User agents supporting CSS2 will be able to understand CSS1 style sheets, while CSS1 user agents are able to read CSS2 style sheets and discarding parts they don’t understand. Also, user agents with no CSS support will be able to view style-enhances documents. Of course, the stylistic enhancements made possible by CSS will not be rendered, but all content will be presented.
- **Complementary to structured documents**. Style sheets complement structured documents (e.g. HTML and XML), providing stylistic information for the marked-up text. It should be easy to change the style sheet with little or no impact on the markup.

- **Vendor, platform and device independence**. Style sheets enable documents to be remain vendor, platform and device independent. Style sheets themselves are also vendor and platform independent, but CSS2 allows you to target a style sheet for a group of devices (e.g. printers).
- **Maintainability**. By pointing to style sheets from documents, Webmasters can simplify site maintenance and retain consistent look and feel throughout the site. For example, if organization's background color changes, only one file needs to be changed.
- **Simplicity**. CSS2 is more complex than CSS1, but it remains a simple style language which is human read- and writable. The CSS properties are kept independent of each other to the largest extent possible and there generally only one way to achieve a certain effect.
- **Network performance**. CSS provides for compact encodings of how to present content. Compared to images or audio files, which are often used by authors to achieve certain rendering effects, using style sheets will decrease the size of the content. Also, fewer network connections have to be opened which further increases network performance.
- **Flexibility**. CSS can be applied to content in several ways. The key feature is the ability to cascade style information specified in: the default UA style sheet, user style sheets, linked style sheets, the document head, and in attributes for the elements forming the document body.
- **Richness**. Providing authors with a rich set of rendering effects increases the richness of the Web as a medium of expression. Designers have been longing for functionality commonly found e.g. in desktop publishing and slide-show applications. Some of the requested rendering effects conflict with device independence, but CSS2 goes a long way of granting designers their requests.
- **Alternate language bindings**. The set of CSS properties described in this specification form a consistent formatting model for visual and aural presentations. This formatting model can be accessed through the CSS language, but bindings to other languages are also possible. For example, a JavaScript program may dynamically change the value a certain element's 'color''color' property.
- **Accessibility**. Last, but not least, using CSS will increase accessibility to Web documents. By retaining textual information in text form, both robots indexing Web pages and human users will have more options for digesting the content. Users can provide their personal style sheets if author-suggested style sheets hinders accessibility. The cascading mechanism negotiates between, and combines, different style sheets.

3 Conformance: Requirements and Recommendations

Contents

1. Definitions [p. 25]
2. Conformance [p. 27]
3. Error conditions [p. 28]

3.1 Definitions

In this section, we begin the formal specification of CSS2, starting with the contract between authors, documents, users, and user agents.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119] [p. 268] . However, for readability, these words do not appear in all upper case letters in this specification.

At times, the authors of this specification recommend good practice for authors and user agents. These recommendations are not normative and conformance with this specification does not depend on their realization. These recommendations contain the expression "We recommend ...", "This specification recommends ...", or some similar wording.

Style sheet

A set of statements that specify presentation of a document.

Style sheets may have three different origins: author [p. 26] , user [p. 27] , and user agent [p. 27] . The interaction of these sources is described in the section on cascading and inheritance [p. 57] .

Valid style sheet

The validity of a style sheet depends on the level of CSS used for the style sheet.

All valid level N-1 style sheets are valid level N style sheets. In particular, all valid CSS1 style sheets are valid CSS2 style sheets.

A valid CSS2 style sheet must respect the grammar of CSS2 [p. 263] and the selector syntax [p. 43] . Furthermore, it must only contain at-rules, property names, and property values defined in this specification.

Source document

The document to which one or more style sheets refer.

Document language

The computer language of the source document (e.g., HTML, XML, etc.).

The primary syntactic constructs of the document language are called *elements,* (an SGML term, see [ISO8879] [p. 267]). Most CSS style sheet rules refer to these elements and specify rendering information for them. Examples of elements in HTML include "P" (for structuring paragraphs), "TABLE" (for creating tables), "OL" (for creating ordered lists), etc.

The *content* of an element is the content of that element in the source document; not all elements have content. The *rendered content* of an element is the content actually rendered. An element’s content is generally

its rendered content. The rendered content of a *replaced element* comes from outside the source document. Rendered content may also be *alternate text* for an element (e.g., the value of the HTML "alt" attribute).

Document tree

User agents transform a document written in the document language into a *document tree* where every element except one has exactly one parent element. (See the SGML ([ISO8879] [p. 267]) and XML ([XML] [p. 268]) specifications for the definition of parent.) The one exception is the *root* element, which has no parent.

An element A is called an *ancestor* of an element B, if either (1) A is the parent B, or (2) A is the parent of some element C that is an ancestor of B.

An element A is called a *descendant* of an element B, if and only if B is an ancestor of A. An element A is called a child of an element B, if and only if B is the parent of A.

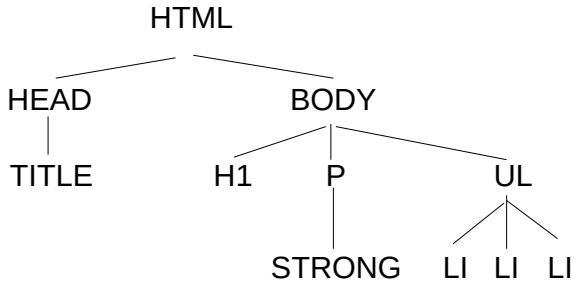
An element A is called a *sibling* of an element B, if and only if B and A share the same parent element. Element A is a preceding sibling if it comes before B in the document tree. Element B is a following sibling if it comes after B in the document tree.

An element A is called a *preceding element* of an element B, if and only if (1) A is an ancestor of B or (2) A is a preceding sibling of B. An element A is called a *following element* of an element B, if and only if (1) A is an descendant of B or (2) A is a following sibling of B.

For example, the following HTML document:

```
<HTML>
  <TITLE>My home page</TITLE>
  <BODY>
    <H1>My home page</H1>
    <P>Welcome to my home page! Let me tell you about my favorite
                composers:
    <UL>
      <LI> Elvis Costello
      <LI> Johannes Brahms
      <LI> Georges Brassens
    </UL>
  </BODY>
</HTML>
```

results in the following tree:



According to the definition of HTML, HEAD elements will be inferred during parsing and become part of the document tree even if the HEAD tags are not in the document source.

Author

An author is a person or program that writes or generates style sheets.

User

A user is a person who interacts with a user agent to view, hear, or otherwise use a document and its associated style sheets.

User agent

A user agent is any program that interprets a document written in the document language and applies associated style sheets according to the terms of this specification. A user agent may display a document, read it aloud, cause it to be printed, convert it to another format, etc.

3.2 Conformance

This section defines conformance with the CSS2 specification only. There may be other levels of CSS in the future that may require a user agent to implement a different set of features in order to conform.

In general, the following points must be observed by user agents claiming conformance to this specification:

1. It must identify the CSS2 media types [p. 63] it supports.
2. For each source document, it must retrieve all associated style sheets that are appropriate for the supported media types. If a user agent cannot retrieve a specified style sheet, it should make a best effort to display the document.
3. It must parse the style sheets according to this specification. In particular, it must recognize all at-rules, blocks, declarations, and selectors (see the grammar of CSS2 [p. 263]). If a user agent encounters a property that applies for a supported media type, the user agent must parse the value according to the property definition. This means that the user agent must accept all legal values and must skip other values. User agents must skip rules that apply to unsupported media types [p. 63] .
4. Given a document tree [p. 26] , it must assign a value for every supported property according to the rules of cascading and inheritance [p. 57] .

Not every user agent must observe every point, however:

- A user agent that *inputs* style sheets must respect points 1 - 3.
- A user agent that *outputs* style sheets is only required to output valid style sheets [p. 25]
- A user agent that *renders* a document with associated style sheets must respect points 1 - 4 and render the document according to the media-specific requirements set forth in this specification.

The inability of a user agent to implement part of this specification due to the limitations of a particular device (e.g., a user agent cannot render colors on a monochrome monitor or page) does not imply non-conformance.

This specification also recommends that a user agent offer the following functionality to the user (these do not refer to any specific user interface):

- Allow the user to specify user style sheets
- Allow the user to turn on or off specific style sheets .
- Approximate style sheet values even if it can’t implement them in strict

accordance with this specification.

3.3 Error conditions

In general, this document does not specify error handling behavior for user agents.

However, user agents must observe the rules for handling parsing errors [p. 35] .

Since user agents may vary in how they handle error conditions, authors and users must not rely on specific error recovery behavior.

4 CSS2 syntax and basic data types

Contents

1. Syntax [p. 29]
    1. Tokenization [p. 29]
    2. Characters and case [p. 31]
    3. Statements [p. 32]
    4. At-rules [p. 32]
    5. Blocks [p. 33]
    6. Rule sets, declaration blocks, and selectors [p. 33]
    7. Declarations [p. 34] and properties [p. 34]
    8. Comments [p. 35]
2. Rules for handling parsing errors [p. 35]
3. Values [p. 36]
    1. Integers and real numbers [p. 36]
    2. Lengths [p. 36]
    3. Percentages [p. 38]
    4. URIs [p. 38]
    5. Colors [p. 39]
    6. Angles [p. 40]
    7. Times [p. 40]
    8. Frequencies [p. 40]
    9. Strings [p. 41]
4. CSS embedded in HTML [p. 41]
5. CSS as a stand-alone file [p. 41]
6. Character escapes in CSS [p. 41]

4.1 Syntax

This section describes a grammar common to any version of CSS (including CSS2). Future versions of CSS will adhere to this core syntax, although they may add additional syntactic constraints.

These descriptions are normative. They are also complemented by the normative grammar rules presented in Appendix B [p. 263] .

4.1.1 Tokenization

All levels of CSS, level 1, level 2, but also any future levels, use the same core syntax. This allows UAs to parse (though not, of course, completely understand) style sheets written in levels of CSS that didn't exist at the time the UAs were created. Designers can use this feature to create style sheets that work with downlevel UA, while also exercising the possibilities of the latest levels of CSS.

CSS style sheets consist of a sequence of tokens. The list of tokens for CSS2 is as follows. The definitions use Lex-style regular expressions. Octal codes refer to [ISO10646] [p. 267] As in Lex, in case of multiple matches, the longest match determines the token.

Token	Definition
IDENT	*{ident}*
ATKEYWORD	*@{ident}*
STRING	*{string}*
HASH	*#{name}*
NUMBER	*{num}*
PERCENTAGE	*{num}*%
DIMENSION	*{num}{ident}*
URI	url\(*{w}{string}{w}*\) |url\(*{w}*([!#$%&*-~]|{nonascii}|{escape})**{w}*\)
UNICODE-RANGE	U\+[0-9A-F?]{1,6}(-[0-9A-F]{1,6})?
CDO	\<!--
CDC	-->
;	;
{	\{
}	\}
(	\(
)	\)
[	\[
]	\]
S	[\t\r\n\f]+
COMMENT	\/*[^*]**+([^/][^*]**+)*\/
FUNCTION	{ident}\(
INCLUDES	~=
DELIM	*any other character*

The macros in curly braces ({}) above are defined as follows:

Macro	Definition
ident	*{nmstart}{nmchar}**
nmstart	[a-zA-Z]|*{nonascii}*|*{escape}*
nonascii	[^\0-\4177777]
unicode	\\[0-9a-f]{1,6}
escape	{unicode}|\\[-~\200-\4177777]
nmchar	[a-z0-9-]|*{nonascii}*|*{escape}*
num	[0-9]+|[0-9]*\.[0-9]+
string	*{string1}*|*{string2}*
string1	\"([\t !#$%&(-~]|\\\n|\'|*{nonascii}*|*{escape}*)*\"
string2	\'([\t !#$%&(-~]|\\\n|\"|*{nonascii}*|*{escape}*)*\'

Below is the core syntax for CSS. The sections that follow describe how to use it. Appendix B [p. 263] describes a more restrictive grammar that is closer to the CSS level 2 language.

```
stylesheet  : [ CDO | CDC | S | statement ]*;
statement   : ruleset | at-rule;
at-rule     : ATKEYWORD S* any* [ block | ';' S* ];
block       : '{' S* [ any | block | ATKEYWORD S* | ';' ]* '}' S*;
ruleset     : selector '{' S* declaration? [ ';' S* declaration ]* '}' S*;
selector    : any+;
declaration : property ':' S* value;
property    : IDENT S*;
value       : [ any | block | ATKEYWORD S* ]+;
any         : [ IDENT | NUMBER | PERCENTAGE | DIMENSION | STRING
              | DELIM | URI | HASH | UNICODE-RANGE | INCLUDES
              | '(' any* ')' | '[' any* ']' ] S*;
```

COMMENT tokens do not occur in the grammar (to keep it readable), but any number of these tokens may appear anywhere.

In some cases, user agents must "skip" part of an illegal style sheet. This specification defines *skip* to mean that the user agent parses the illegal string (from beginning to end), but then skips [p. 31] the string.

An *identifier* consists of letters, digits, hyphens, non-ASCII, and escaped characters. [p. 32]

4.1.2 Characters and case

The following rules always hold:

- All CSS style sheets are case-insensitive, except for parts that are not under the control of CSS. For example, the case-sensitivity of value of the HTML attributes 'id' and 'class', of font names, and of URIs lies outside the scope of this specification. Note in particular that element names are case-insensitive in HTML, but case-sensitive in XML.

- In CSS2, selectors [p. 43] (element names, classes and IDs) can contain only the characters [A-Za-z0-9] and [ISO10646] [p. 267] characters 161 and higher, plus the hyphen (-); they cannot start with a hyphen or a digit. They can also contain escaped characters and any [ISO10646] [p. 267] character as a numeric code (see next item).

  Note that [UNICODE] [p. 268] is code-by-code equivalent to [ISO10646] [p. 267] .
- The backslash (\) followed by at most six hexadecimal digits (0..9A..F) stands for the [ISO10646] [p. 267] character with that number.
- Any character except a hexadecimal digit can be escaped to remove its special meaning, by putting a backslash (\) in front, For example, `"\""` is a string consisting of one double quote.
- The two preceding items define *backslash-escapes*. Backslash-escapes are always considered to be part of an identifier [p. 31] or a string (i.e., "\7B" is not punctuation, even though "{" is, and "\32" is allowed at the start of a class name, even though "2" is not).

4.1.3 Statements

A CSS style sheet, for any version of CSS, consists of a list of *statements* (see the grammar above). There are two kinds of statements: *at-rules* and *rule sets.* There may be whitespace [p. 31] around the statements.

In this specification, the expressions "immediately before" or "immediate after" mean with no intervening white space or comments.

4.1.4 At-rules

At-rules start with an *at-keyword*, which is an identifier [p. 31] beginning with '@' (for example, '@import', '@page', etc.).

An at-rule consists of everything up to and including the next semicolon (;) or the next block, [p. 33] whichever comes first. A CSS UA that encounters an unrecognized at-rule must skip [p. 31] the whole of the @-rule and continue parsing after it.

CSS2 user agents have some additional constraints, e.g., they must also skip [p. 31] any '@import' rule that occurs inside a block [p. 33] or that doesn't precede all rule sets.

Here is an example. Assume a CSS2 parser encounters this style sheet:

```
@import "subs.css";
H1 { color: blue }
@import "list.css";
```

The second '@import' is illegal according to CSS2. The CSS2 parser skips [p. 31] the whole at-rule, effectively reducing the style sheet to:

```
@import "subs.css";
H1 { color: blue }
```

In the following example, the second '@import' rule is invalid, since it occurs inside a '@media' block [p. 33] .

```
@import "subs.css";
@media print {
  @import "print-main.css";
  BODY { font-size: 10pt }
}
H1 {color: blue }
```

4.1.5 Blocks

A *block* starts with a left curly brace ({) and ends with the matching right curly brace (}). In between there may be any characters, except that parentheses (()), brackets ([]) and braces ({}) must always occur in matching pairs and may be nested. Single (') and double quotes (") must also occur in matching pairs, and characters between them are parsed as a string. See Tokenization [p. 29] above for the definition of a string.

Here is an example of a block. Note that the right brace between the double quotes does not match the opening brace of the block, and that the second single quote is an escaped character [p. 32] , and thus doesn't match the first single quote:

```
{ causta: "}" + ({7} * '\'') }
```

Note that the above rule is not legal CSS2, but it is still a block as defined above.

4.1.6 Rule sets, declaration blocks, and selectors

A rule set consists of a selector followed by a declaration block.

A *declaration-block* (also called a {}-block in the following text) starts with a left curly brace ({) and ends with the matching right curly brace (}). In between there must be a list of one or more semicolon-separated (;) declarations.

The *selector* (see also the section on selectors [p. 43]) consists of everything up to (but not including) the first left curly brace ({). A selector always goes together with a {}-block. When a UA can't parse the selector (i.e., it is not valid CSS2), it should skip [p. 31] the {}-block as well.

Note. *CSS2 gives a special meaning to the comma (,) in selectors. However, since it is not known if the comma may acquire other meanings in future versions of CSS, the whole statement should be skipped [p. 31] if there is an error anywhere in the selector, even though the rest of the selector may look reasonable in CSS2.*

For example, since the "&" is not a legal token in a CSS2 selector, a CSS2 UA must skip [p. 31] the whole second line, and not set the color of H3 to red:

```
H1, H2 {color: green }
H3, H4 & H5 {color: red }
H6 {color: black }
```

Here is a more complex example. The first two pairs of curly braces are inside a string, and do not mark the end of the selector. This is a legal CSS2 statement.

```
    P[example="public class foo
{
    private int x;

    foo(int x) {
        this.x = x;
    }

}"] { color: red }
```

4.1.7 Declarations and properties

A *declaration* is either empty or consists of a property, followed by a colon (:), followed by a value. Around each of these there may be whitespace [p. 31] .

Multiple declarations for the same selector may be organized into semicolon (;) separated groups.

Thus, the following rules:

```
H1 { font-weight: bold }
H1 { font-size: 12pt }
H1 { line-height: 14pt }
H1 { font-family: Helvetica }
H1 { font-variant: normal }
H1 { font-style: normal }
```

are equivalent to:

```
H1 {
  font-weight: bold;
  font-size: 12pt;
  line-height: 14pt;
  font-family: Helvetica;
  font-variant: normal;
  font-style: normal;
}
```

A property is an identifier [p. 31] . Any character may occur in the value, but parentheses (()), brackets ([]), braces ({}), single quotes (') and double quotes (") must come in matching pairs. Parentheses, brackets, and braces may be nested. Inside the quotes, characters are parsed as a string.

Values are specified separately for each property, but in any case are built from identifiers, strings, numbers, lengths, percentages, URIs, colors, angles, times, and frequencies.

To ensure that new properties and new values for existing properties can be added in the future, a UA must skip [p. 31] a declaration with an invalid property name or an invalid value. Every CSS2 property has its own syntactic and semantic restrictions on the values it accepts.

For example, assume a CSS2 parser encounters this style sheet:

```
H1 { color: red; font-style: 12pt }  /* Invalid value: 12pt */
P { color: blue;  font-vendor: any;  /* Invalid prop.: font-vendor */
    font-variant: small-caps }
EM EM { font-style: normal }
```

The second declaration on the first line has an invalid value ’12pt’. The second declaration on the second line contains an undefined property ’font-vendor’. The CSS2 parser will skip [p. 31] these declarations, effectively reducing the style

sheet to:

```
H1 { color: red; }
P { color: blue;  font-variant: small-caps }
EM EM { font-style: normal }
```

4.1.8 Comments

Comments begin with the characters "/*" and end with the characters "*/". They may occur anywhere where whitespace [p. 31] can occur and their contents have no influence on the rendering. Comments may not be nested.

CSS also allows the SGML comment delimiters ("<!--" and "-->") in certain places, but they do not delimit CSS comments. They are permitted so that style rules appearing in an HTML source document (in the STYLE element) may be hidden from pre-HTML3.2 user agents. See [HTML40] [p. 268] for more information.

4.2 Rules for handling parsing errors

User agents are required to obey the following rules when it encounters these parsing errors:

- **Unknown properties.** User agents must skip [p. 31] a declaration [p. 34] with an unknown property. For example, if the style sheet is:

  ```
  H1 { color: red; rotation: 70minutes }
  ```

  the UA will treat this as if the style sheet had been

  ```
  H1 { color: red }
  ```

- **Illegal values.** User agents must treat illegal values, *or values with illegal parts*, as if the entire declaration weren't there at all:

  ```
  IMG { float: left }       /* CSS2 */
  IMG { float: left here }  /* "here" is not a value of 'float' */
  IMG { background: "red" } /* keywords cannot be quoted in CSS2 */
  IMG { border-width: 3 }   /* a unit must be specified for length values */
  ```

  In the above example, a CSS2 parser would honor the first rule and skip [p. 31] the rest, as if the style sheet had been:

  ```
  IMG { float: left }
  IMG { }
  IMG { }
  IMG { }
  ```

  A UA conforming to a future CSS specification may accept one or more of the other rules as well.
- User agents must skip [p. 31] an invalid at-keyword together with everything following it, up to and including the next semicolon (;) or brace pair ({...}), whichever comes first. For example, assume the style sheet reads:

```
@three-dee {
  @background-lighting {
    azimuth: 30deg;
    elevation: 190deg;
  }
  H1 { color: red }
}
H1 { color: blue }
```

The ’@three-dee’ at-rule is not part of CSS2. Therefore, the whole at-rule (up to, and including, the third right curly brace) is skipped. [p. 31] A CSS2 UA skips [p. 31] it, effectively reducing the style sheet to:

```
H1 { color: blue }
```

4.3 Values

4.3.1 Integers and real numbers

Some value types may have integer values (denoted by <integer>) or real number values (denoted by <number>). Real numbers and integers are specified in decimal notation only. An <integer> consists of one or more digits "0" to "9". A <number> can either be an <integer>, or it can be zero of more digits followed by a dot (.) followed by one or more digits. Both integers and real numbers may be preceded by a "-" or "+" to indicate the sign.

Note that many properties that allow an integer or real number as a value actually restrict the value to some range, often to a non-negative value.

4.3.2 Lengths

The format of a length value (denoted by <length> in this specification) is an optional sign character (’+’ or ’-’, with ’+’ being the default) immediately followed by a <number> (with or without a decimal point) immediately followed by a unit identifier (e.g., px, deg, etc.). After the number ’0’, the unit identifier is optional.

Some properties allow negative length units, but this may complicate the formatting model and there may be implementation-specific limits. If a negative length value cannot be supported, it should be converted to the nearest value that can be supported.

There are two types of length units: relative and absolute. *Relative length* units specify a length relative to another length property. Style sheets that use relative units will more easily scale from one medium to another (e.g., from a computer display to a laser printer).

Relative units are: em, ex, and px.

```
H1 { margin: 0.5em }      /* em: the height of the element’s font */
H1 { margin: 1ex }        /* ex: the height of the letter ’x’ */
P  { font-size: 12px }    /* px: pixels, relative to viewing device */
```

The ’em’ unit, as used in CSS, is equal to the font size used when rendering an element’s text. It may be used for vertical or horizontal measurement. The ’ex’ unit is equal to the font’s *x-height* (the height of the letter ’x’) of the element’s font. A font need not contain the letter "M" to have an ’em’ size or the letter "x" to have an x-height; the font should still define the two units.

Both ’em’ and ’ex’ refer to the font size of an element except when used in the ’font-size’ property, where they are relative to the font size inherited from the parent element.

The rule:

```
H1 { line-height: 1.2em }
```

means that the line height of H1 elements will be 20% greater than the font size of the H1 elements. On the other hand:

```
H1 { font-size: 1.2em }
```

means that the font-size of H1 elements will be 20% greater than the font size inherited by H1 elements.

When specified for the root of the document tree [p. 26] (e.g., HTML or BODY in HTML), ’em’ and ’ex’ refer to the property’s initial value [p. 57] .

Please consult the section on line height calculations [p. 118] for more information about line heights in the visual flow model [p. 67] .

Pixel units are relative to the resolution of the viewing device, i.e., most often a computer display. If the pixel density of the output device is very different from that of a typical computer display, the UA should rescale pixel values. The suggested *reference pixel* is the visual angle of one pixel on a device with a pixel density of 90dpi and a distance from the reader of an arm’s length. For a nominal arm’s length of 28 inches, the visual angle is about 0.0227 degrees.

Child elements do not inherit the relative values specified for their parent; they inherit the computed values [p. 57] . For example:

```
BODY {
  font-size: 12pt;
  text-indent: 3em;  /* i.e. 36pt */
}
H1 { font-size: 15pt }
```

In these rules, the ’text-indent’ value of H1 elements will be 36pt, not 45pt, if H1 is a child of the BODY element.

Absolute length units are only useful when the physical properties of the output medium are known. The absolute units are: in (inches), cm (centimeters), mm (millimeters), pt (points), and pc (picas).

For example:

```
H1 { margin: 0.5in }      /* inches, 1in = 2.54cm */
H2 { line-height: 3cm }   /* centimeters */
H3 { word-spacing: 4mm }  /* millimeters */
H4 { font-size: 12pt }    /* points, 1pt = 1/72 in */
H4 { font-size: 1pc }     /* picas, 1pc = 12pt */
```

In cases where the specified length cannot be supported, UAs should try to approximate. For all CSS2 properties, further computations and inheritance should be based on the approximated value.

4.3.3 Percentages

The format of a percentage value (denoted by <percentage> in this specification) is an optional sign character (’+’ or ’-’, with ’+’ being the default) immediately followed by a number immediately followed by ’%’.

Percentage values are always relative to another value, for example a length. Each property that allows percentages also defines to which value the percentage refers. When a percentage value is set for a property of the root of the document tree [p. 26] and the percentage is defined as referring to the inherited value of some property X, the resultant value is the percentage times the initial value [p. 57] of property X.

Since child elements inherit the computed values [p. 57] of their parent, in the following example, the children of the P element will inherit a value of 12pt for ’line-height’ (i.e., 12pt), not the percentage value (120%):

```
P { font-size: 10pt }
P { line-height: 120% }  /* relative to ’font-size’, i.e. 12pt */
```

4.3.4 URIs

This specification uses the term Uniform Resource Identifier (URI) as defined in [URI] [p. 268] (see also [RFC1630] [p. 269]).

Note that URIs include URLs (as defined in [RFC1738] [p. 267] and [RFC1808] [p. 267]).

Relative URIs are resolved to full URIs using a base URI. [RFC1808] [p. 267] , section 3, defines the normative algorithm for this process.

URI values in this specification are denoted by <uri>.

For historical reasons, the functional notation used to designate URI values is "url()".

For example:

```
BODY { background: url(http://www.bg.com/pinkish.gif) }
```

The format of a URI value is ’url(’ followed by optional whitespace [p. 31] followed by an optional single quote (’) or double quote (") character followed by the URI itself, followed by an optional single quote (’) or double quote (") character followed by optional whitespace followed by ’)’. Quote characters that are not part of the URI itself must be balanced.

Parentheses, commas, whitespace characters, single quotes (’) and double quotes (") appearing in a URI must be escaped with a backslash: ’\(’, ’\)’, ’\,’.

In order to create modular style sheets that are not dependent on the absolute location of a resource, authors may specify the location of background images [p. 142] with partial URIs. Partial URIs (as defined in [RFC1808] [p. 267]) are interpreted relative to the base URI of the style sheet, not relative to the base URI of the source document.

For example, suppose the following rule is located in a style sheet designated by the URI `http://www.myorg.org/style/basic.css`:

```
BODY { background: url(yellow) }
```

The background of the source document’s BODY will be tiled with whatever image is described by the resource designated by the URI `http://www.myorg.org/style/yellow`.

User agents may vary in how they handle URIs that designate unavailable or inapplicable resources.

4.3.5 Colors

A <color> is either a keyword or a numerical RGB specification.

The suggested list of keyword color names is: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white, and yellow. These 16 colors are taken from the Windows VGA palette, and their RGB values are not defined in this specification.

```
BODY {color: black; background: white }
H1 { color: maroon }
H2 { color: olive }
```

The RGB color model is used in numerical color specifications. These examples all specify the same color:

```
EM { color: #f00 }              /* #rgb */
EM { color: #ff0000 }           /* #rrggbb */
EM { color: rgb(255,0,0) }      /* integer range 0 - 255 */
EM { color: rgb(100%, 0%, 0%) } /* float range 0.0% - 100.0% */
```

In addition to these color keywords, users may specify keywords that correspond to the colors used by certain objects in the user's environment. Please consult the section on system colors [p. 228] for more information.

The format of an RGB value in hexadecimal notation is a '#' immediately followed by either three or six hexadecimal characters. The three-digit RGB notation (#rgb) is converted into six-digit form (#rrggbb) by replicating pairs of digits, not by adding zeros. For example, #fb0 expands to #ffbb00. This makes sure that white (#ffffff) can be specified with the short notation (#fff) and removes any dependencies on the color depth of the display.

The format of an RGB value in the functional notation is 'rgb(' followed by a comma-separated list of three numerical values (either three integer values in the range of 0-255, or three percentage values, typically in the range of 0.0% to 100.0%) followed by ')'. Whitespace [p. 31] characters are allowed around the numerical values.

Values outside the device gamut should be clipped. For a device whose gamut is sRGB, the three rules below are equivalent:

```
EM { color: rgb(255,0,0) }       /* integer range 0 - 255 */
EM { color: rgb(300,0,0) }       /* clipped to 255 */
EM { color: rgb(110%, 0%, 0%) }  /* clipped to 100% */
```

All RGB colors are specified in the sRGB color space (see [SRGB] [p. 268]). UAs may vary in the fidelity with which they represent these colors, but using sRGB provides an unambiguous and objectively measurable definition of what the color should be, which can be related to international standards (see [COLORIMETRY] [p. 267]).

Conforming UAs may limit their color-displaying efforts to performing a gamma-correction on them. sRGB specifies a display gamma of 2.2 under specified viewing conditions. UAs should adjust the colors given in CSS such that, in combination with an output device's "natural" display gamma, an effective display gamma of 2.2 is produced. See the section on gamma correction [p. 251]

for further details. Note that only colors specified in CSS are affected; e.g., images are expected to carry their own color information.

4.3.6 Angles

Angle values (denoted by <angle> in the text) are used with aural cascading style sheets [p. 231] .

Their format is an optional sign character ('+' or '-', with '+' being the default) immediately followed by a <number> immediately followed by an angle unit identifier. After a '0' number, the unit identifier is optional.

These following are legal angle unit identifiers:

- **deg**: degrees.
- **grad**: gradient
- **rad**: radians

Angle values may be negative. They should be normalized to the range 0-360deg by the UA. For example, -10deg and 350deg are equivalent. The angle value must be followed immediately by the angle unit.

4.3.7 Times

Time values (denoted by <time> in the text) are used with aural cascading style sheets [p. 231] .

Their format is a <number> immediately followed by a time unit identifier. After a '0' number, the unit identifier is optional.

The following are legal time unit identifiers:

- **ms**: milliseconds
- **s**: seconds

Time values may not be negative. The time value must be followed immediately by the time unit.

4.3.8 Frequencies

Frequency values (denoted by <frequency> in the text) are used with aural cascading style sheets [p. 231] .

Their format is a <number> immediately followed by a frequency unit identifier. After a '0' number, the unit identifier is optional.

There are two legal frequency unit identifiers:

- **Hz**: Hertz
- **kHz**: kilo Hertz

For example, 200Hz (or 200hz) is a bass sound, and 6kHz (or 6khz) is a treble sound.

The frequency value must be followed immediately by the frequency unit.

4.3.9 Strings

Strings can either be written with double quotes or with single quotes. Double quotes cannot occur inside double quotes, unless escaped (as '\"' or as '\22'). Analogously for single quotes ("\'" or "\27"). Examples:

```
"this is a 'string'"
"this is a \"string\""
'this is a "string"'
'this is a \'string''
```

A string cannot directly contain a newline. To include a newline in a string, use the escape "\A" (hexadecimal A is the line feed character in Unicode, but represents the generic notion of "newline" in CSS). Sometimes it is safer to write "\00000A", since that will avoid the situation where the character following the "A" can be interpreted as a hexadecimal digit. For example, in the string

```
"A. one\AB. two"
```

the UA will see an escape sequence "\AB" («) instead of \A.

It is possible to break strings over several lines, for aesthetic or other reasons, but in such a case the newline itself has to be escaped with a "\". For instance, the following two selectors are exactly the same:

```
A[TITLE="a not s\
o very long title"] {border: double}
A[TITLE="a not so very long title"] {border: double}
```

4.4 CSS embedded in HTML

CSS style sheets may be embedded in HTML documents, and to be able to hide style sheets from older UAs, it is convenient put the style sheets inside HTML comments. Please consult [HTML40] [p. 268] for more information.

When CSS is embedded in HTML, it shares the `charset` parameter used to transmit the enclosing HTML document. As with HTML, the value of the `charset` parameter is used to convert from the transfer encoding to the document character set, which is specified by [ISO10646] [p. 267] .

4.5 CSS as a stand-alone file

CSS style sheets may exist in files by themselves, being linked from the document. In this case, the CSS files are served with the media type `text/css`. As with all text media types, a `charset` parameter may be added which is used to convert from the transfer encoding to [ISO10646] [p. 267] .

4.6 Character escapes in CSS

CSS may need to use characters that are outside the encoding used to transmit the document. For example, the "class" attribute of HTML allows more characters in a class name than the set allowed for selectors above. In CSS2, such characters can be escaped [p. 32] or written as [ISO10646] [p. 267] numbers.

For instance, "B&W?" may be written as "B\&W\?" or "B\26W\3F". For example, a document transmitted as ISO-8859-1 (Latin-1) cannot contain Greek letters directly: "κουρος" (Greek: "kouros") has to be written as "\3BA\3BF\3C5\3C1\3BF\3C2". These escapes are thus the CSS equivalent of numeric character references in HTML or XML documents.

5 Selectors

Contents

1. Pattern matching [p. 43]
2. Universal selector [p. 44]
3. Type selectors [p. 45]
4. Descendant selectors [p. 45]
5. Child selectors [p. 46]
    1. :first-child [p. 46] pseudo-class
6. Adjacent selectors [p. 47]
7. Attribute selectors [p. 47]
    1. Matching attributes and attribute values [p. 47]
        1. Reusing the value of an attribute [p. 48]
    2. The "class" attribute in HTML [p. 48]
8. ID selectors [p. 49]
9. Grouping [p. 50]
10. Pseudo-elements [p. 50] and pseudo-classes [p. 50]
    1. The :first-line [p. 51] pseudo-element
    2. The :first-letter pseudo-element [p. 52]
    3. The :before and :after pseudo-elements [p. 54]
    4. Pseudo-elements with descendant selectors [p. 54]
    5. The Anchor pseudo-classes [p. 54] : :link, :visited, :hover, and :active
    6. Combining pseudo-elements with attribute selectors [p. 55]

5.1 Pattern matching

In CSS, pattern matching rules determine which style rules apply to elements in the document tree [p. 26] . These patterns, called *selectors,* may range from simple element names to rich contextual patterns. If all conditions in the pattern are true for a certain element, the selector *matches* the element.

The *subject* of the pattern is the rightmost part of the pattern (generally an element name). The style information in the declaration block following the pattern applies to the subject of the pattern.

The case-sensitivity of document language element names in selectors depends on the document language. For example, in HTML, element names are case-insensitive, but in XML they are case-sensitive.

The following table summarizes CSS2 selector syntax:

Pattern	Meaning	Described in section
*	Matches any element.	Universal selector [p. 44]
E	Matches any E element.	Type selectors [p. 45]
A B	Matches any B element that is a descendant of an A element.	Descendant selectors [p. 45]
A > B	Matches any B element that is a child of an element A.	Child selectors [p. 46]
A:first-child	Matches element A when A is the first child of some other element	The :first-child pseudo-class [p. 46]
E + F	Matches any F element immediately preceded by an element E.	Adjacent selectors [p. 47]
E[foo]	Matches any E element with the "foo" attribute set (whatever the value).	Attribute selectors [p. 47]
E[foo="warning"]	Matches any E element whose "foo" attribute value is exactly equal to "warning".	Attribute selectors [p. 47]
E[foo~="warning"]	Matches any E element whose "foo" attribute value is a list of space-separated of values, one of which is exactly equal to "warning".	Attribute selectors [p. 47]
DIV.warning	*HTML only*. The same as DIV[class~="warning"].	The "class" attribute in HTML [p. 48]
E#myid	Matches any E element with the "id" attribute equal to "myid".	ID selectors [p. 49]

5.2 Universal selector

The universal selector -- written "*" -- matches the name of any element type. It only matches a single node in the document tree. [p. 26]

In attribute selectors [p. 47] and id selectors [p. 49] , the absence of an explicit element name implies the universal selector. However, we recommend that authors always include the "*" for clarity. For example:

- *[LANG=fr] and [LANG=fr] are equivalent.
- *#myid and #myid are equivalent.

5.3 Type selectors

A *type selector* matches the name of a document language element type. A type selector matches every instance of the element type in the document tree.

The following type selector matches all H1 elements in the document tree:

```
H1 { font-family: Helvetica }
```

Type selectors may be grouped [p. 50] .

5.4 Descendant selectors

At times, authors may want selectors to match an element that is the descendant of another element in the document tree (e.g., "Match those EM elements that are contained by an H1 element"). Descendant selectors express such a relationship in a pattern. A descendant selector matches when an element B is an arbitrary descendant of some ancestor [p. 26] element A. A descendant selector is made up of two or more selectors separated by whitespace [p. 31] .

For example, consider the following rules:

```
H1 { color: red }
EM { color: red }
```

Although the intention of these rules is to add emphasis to text by changing its color, the effect will be lost in a case such as:

```
 <H1>This headline is <EM>very</EM> important</H1>
```

We address this case by supplementing the previous rules with a rule that sets the text color to blue whenever an EM occurs anywhere within an H1:

```
H1 { color: red }
EM { color: red }
H1 EM { color: blue }
```

The third rule will match the EM in the following fragment:

```
<H1>This
   <SPAN class="myclass">headline is <EM>very</EM>
   important</SPAN></H1>
```

The following selector:

```
 DIV * P
```

matches a P element that is a grandchild or later descendant of a DIV element.

Descendant selectors may be grouped [p. 50] . A descendant selector may also contain attribute selectors [p. 47] .

For example, the following matches any element with an "href" attribute inside a P with class "myclass" inside any DIV:

```
DIV P.myclass *[href]
```

5.5 Child selectors

A *child selector* matches when an element is the child [p. 26] of some element. A child selector is made up of two or more selectors separated by ">".

The following rule sets the style of all P elements that are children of BODY:

```
BODY > P { line-height: 1.3 }
```

Child selectors may be grouped [p. 50] . A child selector may also contain attribute selectors [p. 47] .

Descendant selectors and child selectors may be combined. For instance:

```
DIV OL > LI P
```

matches a P element that is a descendant of an LI; the LI element must be the child of an OL element; the OL element must be a descendant of a DIV.

5.5.1 :first-child pseudo-class

The first-child pseudo-class [p. 50] matches an element that is the first child of some other element.

In the following example, the selector matches any P element that is the first child of a DIV element. The rule suppresses indentation for the first paragraph of a DIV:

```
DIV > P:first-child { text-indent: 0 }
```

This selector would match the P inside the DIV of the following fragment:

```
<P> The last P before the note.
<DIV class="note">
   <P> The first P inside the note.
</DIV>
```

but would not match the second P in the following fragment:

```
<P> The last P before the note.
<DIV class="note">
   <H2>Note</H2>
   <P> The first P inside the note.
</DIV>
```

The following rule sets the font weight to "bold" for any EM element that is some descendant of a P element that is a first child:

```
P:first-child EM { font-weight : bold }
```

Note. *Anonymous text boxes [p. 79] are not counted as an element when calculating the first child.*

For example, the EM in:

```
  <P>abc <EM>default</EM>
```

is the first child of the P.

5.6 Adjacent selectors

Often, special formatting rules apply when two types of elements appear next to each other in a document. For example, when block-level elements are laid out, the vertical space between them collapses. In this case, the special formatting is handled by the rules for collapsing margins, [p. 117] but in other cases of adjacent selectors, authors may want to specify their own special formatting rules.

Adjacent selectors have the following syntax: E1 + E2, where E2 is the subject of the selector. The selector matches if E1 and E2 share the same parent in the document tree and E1 immediately precedes E2.

Thus, the following rule states that when a P element immediately follows a MATH element, it should not be indented:

```
MATH + P { text-indent: 0 }
```

The next example reduces the vertical space separating an H1 and an H2 that immediately follows it:

```
H1 + H2 { margin-top: -5mm }
```

Adjacent selectors may be grouped [p. 50] .

Adjacent selectors may also contain attribute selectors [p. 47] .

Adjacent selectors may be combined with other types of selectors.

Thus, for example, the following rule is similar to the one in the previous example, except that the special formatting only occurs when H1 has `class="opener"`:

```
H1.opener + H2 { margin-top: -5mm }
```

5.7 Attribute selectors

CSS2 allows authors to specify rules that match according to attributes defined in the document language.

5.7.1 Matching attributes and attribute values

Attribute selectors may match in three ways:

`[att]`
: Match when the element sets the "att" attribute, whatever the value of the attribute.

`[att=val]`
: Match when the element’s "att" attribute value is exactly "val".

`[att~=val]`
: Match when the element’s "att" attribute value is a space-separated list of "words", one of which is exactly "val". If this selector is used, the words in the value must not contain spaces (since they are separated by spaces).

Attribute values must be quoted or escaped if they are not identifiers. [p. 31]

For example, the following attribute selector matches all H1 elements that specify the "title" attribute, whatever its value:

```
H1[title] { color: blue; }
```

In the following example, the selector matches all SPAN elements whose "class" attribute has exactly the value "example":

```
SPAN[class=example] { color: blue; }
```

Attribute selectors may refer to several attributes, in which case the attribute parts must follow one another, in any order.

Here, the selector matches all SPAN elements whose "hello" attribute has exactly the value "Cleveland" and whose "goodbye" attribute has exactly the value "Columbus":

```
SPAN[hello="Cleveland"][goodbye="Columbus"] { color: blue; }
```

The following rules illustrate the differences between "=" and "~=":

```
A[rel~="copyright"] {} /* matches, e.g., <A rel="copyright copyleft ..." */
TD[colspan="2"] {}     /* matches only <TD colspan="2"> ... */
```

The following rule hides all elements for which the value of the "lang" attribute is "fr" (i.e., the language is French).

```
*[LANG=fr] { display : none }
```

Reusing the value of an attribute

Authors may use the *attr()* function to refer to the value of an attribute in a declaration. The function call is replaced by the value of the attribute.

Note. *The "attr()" function has not be fully specified as of this draft. One stumbling block concerns the type of the value returned by the function call. For instance, some HTML attributes such as width/height may take several types of values: integers (pixel widths), percentages (percentage widths), and the "*i" values (proportional widths). How are these values to be interpreted (generically) in CSS? The editors welcome suggestions on possible solutions to this problem. For instance, one might have several attr() functions for different, well-defined types: attr-integer(), attr-url(), attr-color(), etc.*

5.7.2 The "class" attribute in HTML

For style sheets used with HTML, authors may use the dot (.) notation as an alternative to the "~=" notation. Thus, in HTML, "DIV.value" and "DIV[class~=value]" have the same meaning. The attribute value must immediately follow the ".".

For example, we can assign style information to all elements with `class~="pastoral"` as follows:

```
*.pastoral { color: green }  /* all elements with class=pastoral */
```

or just to H1 elements with `class="pastoral"`:

```
H1.pastoral { color: green }  /* H1 elements with class=pastoral */
```

Given these rules, the first H1 instance below would not have green text, while the second would:

```
<H1>Not green</H1>
<H1 class="pastoral">Very green</H1>
```

To match a subset of "class" values, each value must be preceded by a ".", in any order.

For example, the following rule matches any P element whose "class" attribute has been assigned a list of space-separated values that includes "pastoral" and "marine":

```
P.pastoral.marine { color: green }
```

This rule matches when `class="pastoral blue aqua marine"` but does not match for `class="pastoral blue"`.

Similarly, the following aural style sheet rules allow a script to be read aloud in different voices for each role:

```
P.role.romeo  { voice-family: romeo, male }
P.role.juliet { voice-family: juliet, female }
```

Note. *CSS gives so much power to the "class" attribute, that authors could conceivably design their own "document language" based on elements with almost no associated presentation (such as DIV and SPAN in HTML) and assigning style information through the "class" attribute. Authors should avoid this practice since the structural elements of a document language have recognized and accepted meanings and author-defined classes may not.*

5.8 ID selectors

The ID attribute of a document language allows authors to assign an identifier to a specific element instance in the document tree. This identifier must be unique in the document tree. CSS ID selectors match an element instance based on its identifier.

Each document language may only contain one ID attribute. In HTML 4.0, the ID attribute is called "id", but in an XML application it may be called something else. The name of the ID attribute is immaterial for CSS.

A CSS "id" selector contains a "#" immediately followed by the "id" value.

The following ID selector matches the H1 element whose "id" attribute has the value "chapter1":

```
H1#chapter1 { text-align: center }
```

In the following example, the style rule matches any element that has the "id" value "z98y". The rule will thus match for the P element:

```
<HEAD>
<TITLE>Match P</TITLE>
<STYLE type="text/css">
   *#z98y { letter-spacing: 0.3em }
</STYLE>
</HEAD>
<BODY>
   <P id=z98y>Wide text</P>
</BODY>
```

In the next example, however, the style rule will only match an H1 element that has an "id" value of "z98y". The rule will not match the P element in this example:

```
<HEAD>
<TITLE>Match H1 only</TITLE>
<STYLE type="text/css">
   H1#z98y { letter-spacing: 0.5em }
</STYLE>
</HEAD>
<BODY>
   <P id=z98y>Wide text</P>
</BODY>
```

ID selectors have a higher precedence than attribute selectors. For example, in HTML, the selector `#p123` is more specific than `[ID=p123]` in terms of the cascade [p. 57] .

5.9 Grouping

When several selectors share the same declarations, they may be grouped into a comma-separated list.

In this example, we condense three rules with identical declarations into one. Thus,

```
H1 { font-family: Helvetica }
H2 { font-family: Helvetica }
H3 { font-family: Helvetica }
```

is equivalent to:

```
H1, H2, H3 { font-family: Helvetica }
```

CSS offers other "shorthand" mechanisms as well, including multiple declarations [p. 34] and shorthand properties [p. 16] .

5.10 Pseudo-elements and pseudo-classes

In CSS2, style is normally attached to an element based on its position in the document tree [p. 26] . This simple model is sufficient for many cases, but some common publishing scenarios (such as changing the font size of the first letter of a paragraph) may be independent of the document tree [p. 26] . For instance, in [HTML40] [p. 268] , no element refers to the first line of a paragraph, and therefore no simple CSS selector may refer to it.

CSS introduces the concepts of *pseudo-elements* and *pseudo-classes* to permit formatting based on information that lies outside the document tree.

- Pseudo-elements refer to sub-parts of an element's content (e.g., the first letter or first line of a paragraph, etc.).
- Pseudo-classes refer to elements that are grouped dynamically (e.g., all links that have been visited, all left-hand pages, etc.)

Pseudo-classes are allowed anywhere in selectors while pseudo-elements may only appear as the last segment of a selector.

Although pseudo-elements and pseudo-classes do not exist in the document tree, their behavior is defined as if they did. Each pseudo-element and pseudo-class may be modeled by a *fictional tag sequence,* a fragment of document source that includes imaginary elements from the document language. The fictional tag sequence is only a model used to describe the rendering effects of pseudo-elements and pseudo-classes and does not indicate how these should be implemented.

Pseudo-elements and pseudo-class names are case-insensitive.

Several pseudo-element rules may have an impact on the same content. These are called overlapping pseudo-elements. An example [p. 53] is provided below.

Conforming HTML user agents [p. 27] may skip [p. 31] all rules with :first-line or :first-letter in the selector, or, alternatively, may only support a subset of the properties on these pseudo-elements.

Note. *In CSS2, only one pseudo-element can be specified per selector. This may change in future versions of CSS.*

5.10.1 The :first-line pseudo-element

The :first-line pseudo-element applies special styles to the first formatted line of a paragraph. For instance:

```
P:first-line { font-variant: small-caps }
```

The above rule means "change the font variant of the first line of every paragraph to small-caps". However, the selector "P:first-line" does not match any real HTML element. It does match a pseudo-element that conforming user agents [p. 27] will insert at the beginning of every paragraph.

Note that the length of the first line depends on a number of factors, including the width of the page, the font size, etc. Suppose for this example that the paragraph is broken into the lines indicated in the example. Thus, an ordinary HTML paragraph such as:

```
<P>This is a somewhat long HTML paragraph that will
be broken into several lines. The first line will be
identified by a fictional tag sequence. The other lines will
be treated as ordinary lines in the paragraph.</P>
```

might be "rewritten" by user agents to include the fictional tag sequence for :first-line.

```
<P>
<P:first-line>This is a somewhat long HTML paragraph that will</P:first-line>
be broken into several lines. The first line will be
identified by a fictional tag sequence. The other lines will
be treated as ordinary lines in the paragraph.</P>
```

If a pseudo-element breaks up a real element, the necessary extra tags must be regenerated in the fictional tag sequence. Thus, if we mark up the previous paragraph with a SPAN element:

```
<P><SPAN class="test">This is a somewhat long HTML paragraph that will
be broken into several lines.</SPAN> The first line will be
identified by a fictional tag sequence. The other lines will
be treated as ordinary lines in the paragraph.</P>
```

The user agent should generate the appropriate start and end tags for SPAN when inserting the fictional tag sequence for :first-line.

```
<P><P:first-line><SPAN class="test">This is a somewhat long HTML paragraph that will</SPAN></P:first-line>
<SPAN>be broken into several lines.</SPAN> The first line will be
identified by a fictional tag sequence. The other lines will
be treated as ordinary lines in the paragraph.</P>
```

The :first-line pseudo-element can only be attached to a block-level element.

The :first-line pseudo-element is similar to an inline element, but with certain restrictions. Only the following properties apply to a :first-line element: font properties, [p. 149] color properties, [p. 141] background properties, [p. 142] 'word-spacing', 'letter-spacing', 'text-decoration', 'vertical-align', 'text-transform', 'line-height', and 'clear'.

5.10.2 The :first-letter pseudo-element

[Define better alignment of drop caps? BB]

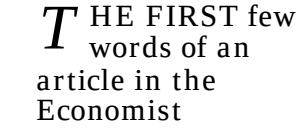
The :first-letter pseudo-element may be used for "initial caps" and "drop caps", which are common typographical effects. This type of initial letter is similar to an inline element if its 'float' property is 'none', otherwise it is similar to a floating element.

These are the properties that apply to :first-letter pseudo-elements: font properties, [p. 149] color properties, [p. 141] background properties, [p. 142] 'text-decoration', 'vertical-align' (only if 'float' is 'none'), 'text-transform', 'line-height', margin properties, [p. 99] padding properties, [p. 102] border properties, [p. 104] 'float', and 'clear'.

The following CSS2 will make a dropcap initial letter span two lines:

```
<HTML>
 <HEAD>
  <TITLE>Dropcap initial letter</TITLE>
  <STYLE type="text/css">
   P              { font-size: 12pt; line-height: 12pt }
   P:first-letter { font-size: 200%; font-style: italic; font-weight: bold; float: left }
   SPAN           { text-transform: uppercase }
  </STYLE>
 </HEAD>
 <BODY>
  <P><SPAN>The first</SPAN> few words of an article in The Economist.</P>
 </BODY>
</HTML>
```

This example might be formatted as follows:

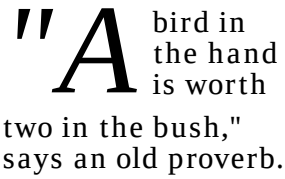


The fictional tag sequence is:

```
<P>
<SPAN>
<P:first-letter>
T
</P:first-letter>he first
</SPAN>
few words of an article in the Economist.
</P>
```

Note that the :first-letter pseudo-element tags abut the content (i.e., the initial character), while the :first-line pseudo-element start tag is inserted right after the start tag of the element to which it is attached.

The UA defines what characters are inside the :first-letter element. Quotes that precede the first letter should be included, as in:



When the paragraph starts with other punctuation (e.g., parenthesis and ellipsis points) or other characters that are normally not considered letters (e.g., digits and mathematical symbols), :first-letter pseudo-elements are generally skipped [p. 31]

The :first-letter pseudo-element can only be attached to a block-level element.

Note. *Some languages may have specific rules about how to treat certain letter combinations. In Dutch, for example, if the letter combination "ij" appears at the beginning of a word, they should both be considered within the :first-letter pseudo-element.*

The following example illustrates how overlapping pseudo-elements may interact. The first letter of each P element will be green with a font size of ’24pt’. The rest of the first formatted line will be ’blue’ while the rest of the paragraph will be ’red’.

```
P { color: red; font-size: 12pt }
P:first-letter { color: green; font-size: 200% }
P:first-line { color: blue }

<P>Some text that ends up on two lines</P>
```

Assuming that a line break will occur before the word "ends", the fictional tag sequence for this fragment might be:

```
<P>
<P:first-line>
<P:first-letter>
S
</P:first-letter>ome text that
</P:first-line>
ends up on two lines
</P>
```

Note that the :first-letter element is inside the :first-line element. Properties set on :first-line are inherited by :first-letter, but are overridden if the same property is set on :first-letter.

When the :first-letter and :first-line pseudo-elements are combined with :before and :after, they apply to the first letter or line of the element including the inserted text.

5.10.3 The :before and :after pseudo-elements

The ':before' and ':after' pseudo-elements can be used to insert fixed text before or after an element. They are explained in the section on generated text. [p. 129]

5.10.4 Pseudo-elements with descendant selectors

In a descendant selector, pseudo-elements are only allowed at the end of the selector.

The following example illustrates this with the :first-letter pseudo-element.

```
BODY P:first-letter { color: purple }
```

Pseudo-classes, however, may be used anywhere in a descendant selector.

The following example sets the border color to blue of all images that descend from A elements that have not yet been visited:

```
A:link IMG { border: solid blue }
```

5.10.5 The Anchor pseudo-classes: :link, :visited, :hover, and :active

User agents commonly display unvisited links differently from previously visited ones. CSS2 allows authors to specify the rendering of a link in one of several states:

- The :link pseudo-class applies for links that have not yet been visited.
- The :visited pseudo-class applies once the link has been visited by the user. **Note.** After a certain amount of time, user agents may choose to return a visited link to the (unvisited) 'link' state.
- The :hover pseudo-class applies when the user designates, but does not activate, a link. For example, a visual user agent could apply this pseudo-class when the cursor hovers over an element. [or, in our terminology, do cursors only hover over boxes? -howcome]
- The :active pseudo-class applies while the link is being activated by the user.

The four states are mutually exclusive. If a link qualifies for several states, the order of preference is: active, hover, visited, link.

User agents are not required to reflow a currently displayed document due to anchor pseudo-class transitions. For instance, a style sheet may legally specify that the 'font-size' of an :active link should be larger that a :visited link, but the UA is not required to dynamically reflow the document when the reader selects the :visited link.

```
A:link { color: red }       /* unvisited links */
A:visited { color: blue }   /* visited links   */
A:hover { color: yellow }   /* user hovers     */
A:active { color: lime }    /* active links    */
```

In HTML, the following two CSS2 declarations are equivalent and select the same elements:

```
A:link { color: red }
:link { color: red }
```

5.10.6 Combining pseudo-elements with attribute selectors

Pseudo-classes can be combined with attribute selectors [p. 47] . In this case, the class name must precede the pseudo-class name in the selector.

If the following link:

```
<P>
<A class="external" href="http://out.side/">external link</A>
```

has been visited, this rule:

```
A.external:visited { color: blue }
```

will cause it to be blue.

Pseudo-elements can also be combined with attribute selectors.

Thus, the following rule:

```
P.initial:first-letter { color: red }
```

would make the first letter of all P elements with `class="initial"` such as the following, the color ’red’, as in:

```
<P class="initial">First paragraph</P>
```

Pseudo-elements must be specified at the end of a selector.

6 Assigning property values, Cascading, and Inheritance

Contents

1. Specified, computed, and absolute values [p. 57]
    1. Specified values [p. 57]
    2. Computed values [p. 57]
    3. Actual values [p. 58]
2. Inheritance [p. 58]
    1. The inherit [p. 59] value
3. The cascade [p. 59]
    1. Cascading order [p. 60]
    2. ’Important’ rules [p. 60]
    3. Cascading order in HTML [p. 61]
    4. Precedence of non-CSS presentational hints [p. 61]

6.1 Specified, computed, and absolute values

Once a user agent has parsed a document and constructed a document tree [p. 26] , it must assign, for every node in the tree, a value to every property that applies to the target media type [p. 63] .

The final value of a property is the result of a three-step calculation: the value is determined through specification (the "specified value"), then resolved into an absolute value if necessary (the "computed value"), and finally transformed according to the limitations of the local environment (the "actual value") if necessary.

6.1.1 Specified values

User agents must first assign a specified value to a property based on the following mechanisms (in order of precedence):

1. If the cascade [p. 59] results in a value, use it.
2. Otherwise, if the value is inherited [p. 58] , use it.
3. Otherwise use the property’s initial value. The initial value of each property is indicated in the property’s definition.

6.1.2 Computed values

Specified values may be "absolute" (e.g., the color value ’red’ or the constant value ’hidden’) or "relative" (e.g., the variable value ’auto’, the font-related value ’em’, pixel values, percentage values, etc.). Each value must be transformed into a computed value according to algorithms described in this specification.

When the root of the document tree has a property whose specified value is inherited and has relative units, the computed value is the percentage times the property’s initial value.

For example, with an HTML document and the following style sheet:

```
HTML {font-size: 120%}
```

the computed value for ’font-size’ will be 120% of the initial value of the ’font-size’ property. The initial value of ’font-size’ is defined to be ’medium’, so the actual value is 20% larger than ’medium’. The actual value that this results in depends on the current environment.

6.1.3 Actual values

A computed value has an absolute meaning but a user agent may not be able to respect this meaning in a given environment. For example, a user agent may not have a specified font size available, in which case the user agent must approximate the computed value. Computed values that are transformed to match the current environment are called actual values.

Only actual values are inherited.

6.2 Inheritance

Some actual values [p. 58] are inherited by the descendants of a node in the document tree [p. 26] . Each property definition specifies whether its value may be inherited.

Suppose there is an H1 element with an emphasized element inside:

```
<H1>The headline <EM>is</EM> important!</H1>
```

If no color has been assigned to the EM element, the emphasized "is" will inherit the color of the parent element, so if H1 has the color blue, the EM element will likewise be in blue.

The root of the document tree cannot inherit values.

To set a "default" style property for a document, authors may set the property on the root of the document tree. In HTML, for example, the HTML or BODY elements can serve this function. Note that this will work even if the author omits the BODY tag in the HTML source since the HTML parser will infer the missing tag.

For example, these rules cause the ’color’ property on the BODY element to be inherited by all descendants of the BODY element:

```
BODY {
  color: black;
  background: url(texture.gif) white;
}
```

In this example, all descendants of the BODY element inherit the ’color’ property.

Not all style properties are inherited. For example, the ’background’ property is not inherited. (However, due to the initial value of ’transparent’ on the ’background’ property, the parent’s background shines through.)

The following example illustrates that specified [p. 57] percentage values are not inherited; only actual values [p. 58] are inherited. Consider the style sheet:

```
BODY {font-size: 10pt}
H1 {font-size: 120%}
```

and the document fragment:

```
<BODY>
<H1>A <EM>large</EM> heading</H1>
</BODY>
```

The computed value [p. 57] of the ’font-size’ property for the H1 element is 12pt (120% times 10pt). If the user agent has the appropriate 12pt font available, 12pt will also be the property’s actual value [p. 58] and the EM will inherit that value for the ’font-size’ property. However, if the user agent does not have the 12pt font available, it may assign an actual value of, for example, 11pt to the ’font-size’ property of the H1 element. In that case, the EM will inherit a value of 11pt for the same property.

6.2.1 The inherit value

Each property may also take the value ’inherit’, which means that, for a given element, the property takes the same computed value [p. 57] as the property for the element’s parent.

6.3 The cascade

Style sheets may have three different origins: author, user, and user agent.

- **Author**. The author specifies style sheets for a source document according to the conventions of the document language. For instance, in HTML, style sheets may be included in the document or linked externally.
- **User**: The user may be able to specify style information for a document. The user agent may provide an interface that "generates" a user style sheet (or must behave as if it did).
- **User agent**: Conforming user agents [p. 27] must apply a *default style sheet* (or behave as if they did) prior to all other style sheets for a document. A user agent’s default style sheet should present the elements of the document language in ways that satisfy general presentation expectations for the document language (e.g., for visual browsers, the EM element in HTML is presented using an italic font). See "A sample style sheet for HTML 4.0" [p. 245] for a recommended default style sheet for HTML 4.0 documents.

  Note that default style sheet may change if system settings are modified by the user (e.g., system colors). However, due to limitations in a user agents’ internal implementation, it may be impossible to change the values in the default style sheet.

Style sheets from these three origins will overlap in scope, and they interact according to the cascade.

The CSS cascade assigns a weight to each style rule. When several rules apply, the one with the greatest weight takes precedence.

By default, rules in a user's personal style sheets have less weight than rules in the author's style sheets. Thus, if there are conflicts between the style sheets of an incoming document and the reader's personal sheets, the author's rules will be used. Both reader and author rules override the UA's default style sheet.

Imported style sheets also cascade and their weight depends on their import order. Rules specified in a given style sheet override rules imported from other style sheets. Imported style sheets can themselves import and override other style sheets, recursively, and the same precedence rules apply.

6.3.1 Cascading order

To find the value for an element/property combination, user agents must apply the following algorithm:

1. Find all declarations that apply to the element/property in question. Declarations apply if the associated selector matches [p. 43] the element in question. If no declarations apply, terminate the algorithm.
2. Sort the declarations by explicit weight: declarations marked '!important' carry more weight than unmarked (normal) declarations. See the section on 'important' [p. 60] rules for more information.
3. Sort by origin: the author's style sheets override the user's style sheets which override the default style sheet. An imported style sheet has the same origin as the style sheet from which it is imported.
4. Sort by specificity of selector: more specific selectors will override more general ones. The definition and calculation of specificity is object-language dependent. Pseudo-elements and pseudo-classes are counted as normal elements and classes, respectively.
5. Sort by order specified: if two rules have the same weight, the latter specified wins. Rules in imported style sheets are considered to be before any rules in the style sheet itself.

The search for the property value must be terminated when any of the above steps yields a rule that has a higher weight than the other rules that apply to the same element/property combination.

This strategy gives author's style sheets considerably higher weight than those of the reader. It is therefore important that the User agent gives the user the ability to turn off the influence of a certain style sheet, e.g., through a pull-down menu.

6.3.2 'Important' rules

Style sheet designers can increase the weights of their declarations by declaring them 'important'.

```
H1 { color: black ! important; background: white ! important }
P  { font-size: 12pt ! important; font-style: italic }
```

In the example above, the first three declarations have increased weight, while the last declaration has normal weight.

A reader rule with an important declaration will override an author rule with a normal declaration. An author rule with an important declaration will override a reader rule with an important declaration.

Declaring a shorthand property (e.g., 'background') to be important is equivalent to declaring all of its sub-properties important.

6.3.3 Cascading order in HTML

In HTML, a selector's specificity is calculated as follows:

- (a) count the number of "id" attributes in the selector
- (b) count the number of other attributes in the selector (including class attributes)
- (c) count the number of element names in the selector

Concatenating the three numbers (in a number system with a large base) gives the specificity.

Some examples:

```
LI            {}  /* a=0 b=0 c=1 -> specificity =   1 */
UL LI         {}  /* a=0 b=0 c=2 -> specificity =   2 */
UL OL+LI      {}  /* a=0 b=0 c=3 -> specificity =   3 */
/H1 [REL=up]/ {}  /* a=0 b=1 c=1 -> specificity =  11 */
UL OL LI.red  {}  /* a=0 b=1 c=3 -> specificity =  13 */
LI.red.level  {}  /* a=0 b=2 c=1 -> specificity =  21 */
#x34y         {}  /* a=1 b=0 c=0 -> specificity = 100 */
```

A declaration in the "style" attribute of an element has the same weight as a declaration with an "id"-based selector that is specified at the end of the style sheet:

```
<HEAD>
<STYLE type="text/css">
  #x97z { color: blue }
</STYLE>
</HEAD>
<BODY>
<P ID=x97z style="color: red">
</BODY>
```

In the above example, the color of the P element would be red. Although the specificity is the same for both declarations, the declaration in the "style" attribute will override the one in the STYLE element because of cascading rule number 5.

6.3.4 Precedence of non-CSS presentational hints

The UA may choose to honor presentational hints from other sources than style sheets, for example the FONT element or the "align" attribute in HTML. If so, the non-CSS presentational hints must be translated to the corresponding CSS rules with specificity equal to 1. The rules are assumed to be at the start of the author style sheet and may be overridden by subsequent style sheet rules.

Note. *In a transition phase, this policy will make it easier for stylistic attributes to coexist with style sheets.*

7 Media types

Contents

1. Introduction to media types [p. 63]
2. Specifying media-dependent style sheets [p. 63]
    1. The @media rule [p. 64]
    2. The media-dependent @import rule [p. 64]
3. Recognized media types [p. 64]
    1. Media groups [p. 65]

7.1 Introduction to media types

One of the most important features of style sheets is that they allow authors to specify how a document is to be presented on different media: on the screen, on paper, with a speech synthesizer, with a braille device, etc.

Certain CSS properties are only designed for certain media (e.g., the 'cue-before' property for aural style sheets). On occasion, however, style sheets for different media types may share a property, but require different values for that property. For example, the 'font-size' property is useful both for screen and print media. However, the two media are different enough to require different values for the common property; a document will typically need a larger font on a computer screen than on paper. Experience also shows that sans serif fonts are easier to read on screen, while fonts with serifs are easier to read on paper. For these reasons, it is necessary to express that a style sheet -- or a section of a style sheet -- applies to certain media types.

Below we describe how authors may specify different style sheets for different media (all of which participate in the cascade [p. 57]).

7.2 Specifying media-dependent style sheets

There are currently two ways to specify media dependencies for style sheets:

- Specify the target medium from a style sheet with the @media or @import at-rules.

```
@import url(loudvoice.css) speech;
@media print {
  /* style sheet for print goes here */
}
```

- Specify the target medium within the document language. For example, in [HTML40] [p. 268] , the "media" attribute on the LINK element specifies the target medium of an external style sheet:

```
<HTML>
<HEAD>
<TITLE>Link to a target medium</TITLE>
<LINK rel="stylesheet" type="text/css"
         media="print" href="foo.css">
</HEAD>
<BODY>
<P>The body...
</BODY>
</HTML>
```

Since these two examples specify the same media type [p. 64] , they are semantically equivalent.

7.2.1 The @media rule

An @media rule lists the media types [p. 64] (separated by commas) affected by a set of rules delimited by curly braces.

The @media construct allows style sheet rules for various media in the same style sheet:

```
@media print {
  BODY { font-size: 10pt }
}
@media screen {
  BODY { font-size: 12pt }
}
@media screen, print {
  BODY { line-height: 1.2 }
}
```

7.2.2 The media-dependent @import rule

So that user agents can avoid retrieving resources for unsupported media types, authors may specify media-dependent @import [p. 57] rules. These conditional imports specify comma-separated media types after the URI.

The following rules have the same effect as if the imported style sheet were wrapped in an @media rule for the same media, but it may save the UA a fruitless download.

```
@import url(fineprint.css) print;
@import url(bluish.css) projection, tv;
```

In the absence of any media types, the import is unconditional. Specifying ’all’ for the medium has the same effect.

7.3 Recognized media types

Due to rapidly changing technologies, CSS2 does not specify a definitive list of media types that may be values for @media. However, user agents that elect to support the devices in the following list must recognize the associated media type:

all
: Suitable for all devices.

aural
: Intended for speech synthesizers. See the section on aural style sheets [p. 231] for details.

braille
: Intended for braille tactile feedback devices.

embossed
: Intended for paged braille printers.

handheld
: Intended for handheld devices (small screen, monochrome, limited bandwidth).

print
: Intended for paged, opaque material and for documents viewed on screen in print preview mode. Please consult the section on paged media [p. 131] for information about formatting issues that are specific to paged media.

projection
: Intended for projected presentations, for example projectors or print to transparencies. Please consult the section on paged media [p. 131] for information about formatting issues that are specific to paged media.

screen
: Intended primarily for color computer screens. See the section on rendering to continuous media [p. 65] for more information.

tty
: Intended for media using a fixed-pitch character grid, such as teletypes, terminals, or portable devices with limited display capabilities. Authors should not use pixel units [p. 37] with the "tty" media type.

tv
: Intended for television-type devices (low resolution, color, limited scrollability).

Media type names are case-insensitive.

7.3.1 Media groups

Each CSS property definition specifies the media types for which the property must be implemented by a conforming user agent [p. 27] . Since properties generally apply to several media, the "Applies to media" section of each property definition indicates a media group rather than a list of media types. A property applies to all media types that belong to a given media group.

CSS2 defines the following media groups:

- **continuous** or **paged**. When "continuous" or "paged" are not specified explicitly, the property in question applies to both media groups.
- **visual**, **aural**, or **tactile**. When "visual", "aural", or "tactile" are not specified explicitly, the property in question applies to all three media groups.
- **grid** (for character grid devices), or **bitmap**. When "grid" or "bitmap" are not specified explicitly, the property in question applies to both media groups.
- **all** (includes all media types)

The following table shows the relationships between media groups and media types:

Media Groups

Media Groups / Media Types	continuous/paged	visual/aural/tactile	character grid
all			
aural	continuous	aural	N/A
braille	continuous	tactile	grid
emboss	paged	tactile	grid
handheld		visual	
print	paged	visual	
projection	paged	visual	
screen	continuous	visual	
tty	continuous	visual	grid
tv		visual, aural	

8 Visual rendering model

Contents

1. Introduction to the visual rendering model [p. 67]
    1. The viewport [p. 68]
2. The box model [p. 68]
    1. Controlling box generation: [p. 68] the ’display’ property
        1. Compact and run-in boxes [p. 70]
    2. Box dimensions [p. 71]
    3. Example of margins, padding, and borders [p. 73]
3. Positioning schemes [p. 76]
    1. Choosing a positioning scheme: [p. 76] ’position’ property
    2. Box offsets [p. 77] : ’top’, ’right’, ’bottom’, ’left’
4. Normal flow [p. 79]
    1. Anonymous boxes [p. 79]
    2. Block formatting context [p. 79]
    3. Inline formatting context [p. 79]
    4. Direction of inline flow [p. 81] : the ’direction’ property
    5. Relative positioning [p. 82]
5. Floats [p. 82] : the ’float’ and ’clear’ properties
    1. Controlling flow next to floats [p. 84]
6. Absolute positioning [p. 85]
    1. Fixed positioning [p. 85]
7. Relationships between ’display’, ’position’, and ’float’ [p. 87]
8. Comparison of normal, relative, floating, absolute positioning [p. 87]
    1. Normal flow [p. 88]
    2. Relative positioning [p. 88]
    3. Floating a box [p. 89]
    4. Absolute positioning [p. 92]
9. Z-order [p. 96] : Layered presentation
    1. Specifying the stack level [p. 96] : the ’z-index’ property

8.1 Introduction to the visual rendering model

This chapter describes the visual rendering model, how user agents process the document tree [p. 26] for visual media [p. 63] .

In the visual rendering model, each element in the document tree generates zero or more rectangular boxes [p. 68] that are then rendered. Some boxes belong to the "normal flow" of boxes while others are "outside the flow". A box in the normal flow has a *preceding box* in the normal flow (unless it is the first box) and a *following box* in the normal flow (unless it is the last box).

Most boxes establish a containing block [p. 111] whose edges serve as references for the layout of descendant boxes (see the next chapter [p. 99] for details). in the CSS visual rendering model, a box establishes reference edges for its descendants and is itself positioned with respect to its containing block. A box is not confined by its containing block -- it is positioned with respect to its edges and may even overflow [p. 123] those edges. When a box is floated [p. 82]

inside a containing block, layout of boxes in the containing block is also affected by the edges of those floating boxes.

The box model [p. 68] describes the generation of boxes. The layout of these boxes is governed by:

- box dimensions [p. 71] and type [p. 68] .
- positioning scheme [p. 76] (normal, float, and absolute positioning models).
- relationships between elements in the document tree. [p. 26]
- external information (e.g., viewport size, intrinsic dimensions of images, etc.).

The properties defined in this chapter apply to both continuous media [p. 65] and paged media [p. 65] . However, the meanings of the margin properties [p. 99] vary when applied to paged media (see the page model [p. 131] for details).

The next chapter [p. 99] supplies the details of the visual rendering model. However, the model does not specify all aspects of formatting (e.g., it does not specify a letter-spacing algorithm). Conforming user agents [p. 27] may behave differently for those formatting issues not covered by this specification.

8.1.1 The viewport

User agents for continuous media [p. 65] generally offer users a *viewport* (a window or other viewing area on the screen) through which users consult a document. User agents may change the document's layout when the viewport is resized (see the initial containing block [p. 111]). When the viewport is smaller than the document's containing block, the user agent should offer a scrolling mechanism. There is at most one viewport per canvas [p. 22] , but user agents may offer users several views of a document.

8.2 The box model

The CSS box model describes how rectangular boxes are generated for elements in the document tree [p. 26] . The page box [p. 131] is a special kind of box that is described in detail on the section on paged media [p. 131] .

8.2.1 Controlling box generation: the 'display' property

The 'display' property determines whether an element generates a box, and if so, what type of box it generates.

'display'

Property name:	'display'
Value:	inline | block | list-item | none | run-in | compact | table | inline-table| table-row-group | table-column-group | table-header-group | table-footer-group | table-row | table-cell | table-caption | inherit
Initial:	inline
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

Block-level elements are those elements of the document language that, by default, are formatted visually as blocks (e.g., paragraphs). Several values of the 'display' property make an element block-level: 'block', 'list-item', 'compact' and 'run-in' (part of the time; see compact and run-in boxes [p. 70]), and 'table'.

Most block-level elements generate block-level boxes, but some elements also produce inline anonymous [p. 79] boxes.

Inline elements are those elements of the document language that do not cause paragraph breaks (e.g., pieces of text, inline images, etc.). Several values of the 'display' property make an element inline: 'inline', 'inline-table', 'compact' and 'run-in' (part of the time; see compact and run-in boxes [p. 70]). Inline elements generate inline boxes.

A block-level box acts as a containing block [p. 67] for its descendant boxes, which are either block-level boxes or inline boxes (but not both). Sibling block-level boxes participate in a block formatting context [p. 79] .

An inline [p. 69] box participates in an inline formatting context [p. 79] with its siblings and children.

The values of this property have the following meanings:

'block'

This value causes an element to generate a block-level box.

'inline'

This value causes an element to generate an inline box.

'list-item'

This value causes an element to generate a block-level [p. 69] box that also has a list-item marker box. For example, in HTML, the LI element will typically have this 'display' value. For information about lists and examples of list formatting, please consult the section on lists [p. 193] .

'none'

This value causes an element to generate **no** boxes in the rendering structure [p. 21] (i.e., the element has no effect on layout). Descendant elements do not generate any boxes either; this behavior **cannot** be overridden by setting the 'display' property on the descendants.

Please note that a display of 'none' does not create an invisible box; it creates no box at all. CSS includes mechanisms that enable an element to generate boxes in the rendering structure that affect formatting but are not visible themselves. Please consult the section on visibility [p. 127] for details.

’run-in’ and **’compact’**
: These values create a box that is block-level or inline, depending on context. These values are described below.

’table’, <inline-table>, **’table-row-group’**, **’table-column-group’**, ’table-header-group’, ’table-footer-group’, **’table-row’**, **’table-cell’**, and **’table-caption’**
: These values cause an element to behave like a table element (subject to restrictions described in the chapter on tables [p. 201]).

Note that although the initial value [p. 57] of ’display’ is ’inline’, the user agent’s default style sheet [p. 59] may override [p. 57] this value for each element of the document language [p. 25] . See the sample style sheet [p. 245] for HTML 4.0 in the appendix.

Here are some examples of the ’display’ property:

```
P   { display: block }
EM  { display: inline }
LI  { display: list-item }
IMG { display: none }      /* Don’t display images */
```

Conforming HTML user agents [p. 27] may skip [p. 31] the ’display’ property when specified in author and user style sheets but must specify a value for it in the default style sheet [p. 59] .

Compact and run-in boxes

A *compact box* behaves as follows:

- If a block-level [p. 69] or inline [p. 69] box (that does not float and is not absolutely positioned) follows [p. 67] the compact box, the compact box is formatted like a one-line inline box. If the resulting box width [p. 73] is less than or equal to the left margin of the block-level box, the inline box is positioned in the margin as described immediately below. If the writing direction of the following block-level box is right-to-left (the ’direction’ property value is ’rtl’), the inline box is placed in the right margin.
- Otherwise, the compact box behaves like a block-level box.

The compact box is positioned in the margin as follows: it is outside (to the left or right) of the first line box [p. 79] of the block, but it affects the calculation of that line box’s height [p. 115] . The ’vertical-align’ property of the compact box determines the vertical position of the compact box relative to that line box. The horizontal position of the compact box is always in the margin of the block-level box, as far to the outside as possible.

The following example illustrates a compact box.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>A compact box example</TITLE>
<STYLE type="text/css">
  DT {display: compact}
  DD {margin-left: 4em}
</STYLE>
</HEAD>
<BODY>
```

```
<DL>
  <DT>Short
    <DD><P>Description goes here.
  <DT>too long for the margin
     <DD><P>Description goes here.
</DL>
</BODY>
</HTML>
```

This example might be rendered as:

```
short    Description goes here

too long for the margin
         Description goes here
```

A *run-in box* behaves as follows:

- If a block-level [p. 69] or inline [p. 69] box (that does not float and is not absolutely positioned) follows [p. 67] the run-in box, the run-in box behaves like an inline child of the block-level box.
- Otherwise, the run-in box behaves like a block-level box.

A ’run-in’ box, on the other hand, is useful for run-in headers, as in this example:

```
<HTML>
<HEAD>
<TITLE>A run-in box example</TITLE>
<STYLE type="text/css">
  H3 {display: run-in}
  H3:after {content: ". "}
</STYLE>
</HEAD>
<BODY>
<H3>A run-in heading</H3>
<P>And a paragraph of text that
follows it.
</BODY>
</HTML>
```

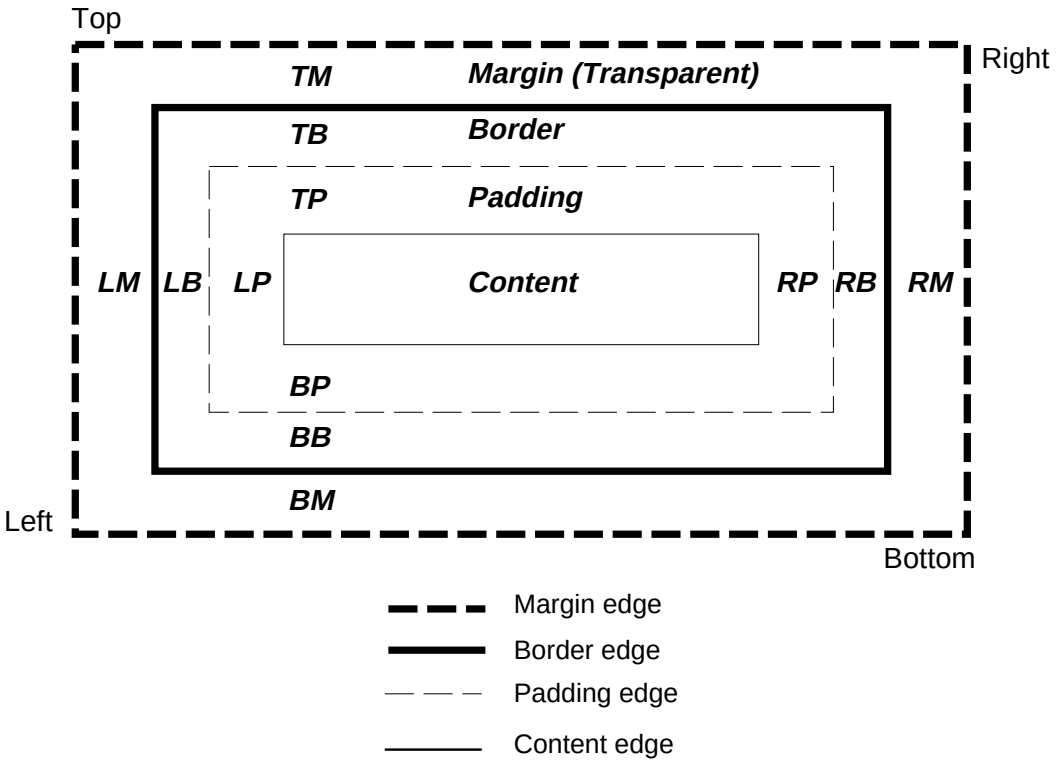
This example might be rendered as:

```
A run-in heading. And a
paragraph of text that
follows it.
```

Properties apply to run-in and compact boxes based on their final status (inline or block-level). For example, the ’white-space’ property only applies if the box behaves like a block-level box.

8.2.2 Box dimensions

Each box has a *content area* (e.g., text, an image, etc.) and optional surrounding *padding*, *border*, and *margin* areas; the size of each area is specified by properties defined in the next chapter [p. 99] . The following diagram shows how these areas relate and the terminology used to refer to pieces of margin, border, and padding:



The padding, border, and padding can be broken down into left, right, top, and bottom segments (e.g., in the diagram, "LM" for left margin, "RM" for right margin, "TM" for top margin, "BM" for bottom margin, etc.).

The perimeter of each of the four areas (content, padding, border, and margin) is called an "edge", so each box has four edges:

content edge or **inner edge**
: The content edge surrounds the element's rendered content [p. 25] .

padding edge
: The padding edge surrounds the box padding. If the padding has 0 width, the padding edge is the same as the content edge. The padding edge of a box defines the edges of the containing block [p. 67] established by the box.

border edge
: The border edge surrounds the box border. If the padding has 0 width, the border edge is the same as the padding edge.

margin edge or **outer edge**
: The margin edge surrounds the box margin. If the margin has 0 width, the margin edge is the same as the border edge.

Each edge may be broken down into a left, right, top, and bottom edge.

The dimensions of the content area of a box -- the *content width* and *content height* -- may be established in one of several ways:

Width and height properties set explicitly
: The 'width' and 'height' properties specify a dimension explicitly. Except for table [p. 201] cells, specified values other than 'auto' for 'width' and 'height' cannot be overridden for a generated box.

Block-level box widths are calculated top-down

The width of a block-level box is given by the width of its containing block and the box's margins, borders, and padding. Please consult the sections on box width calculations [p. 112] for details.

Inline box widths are calculated bottom-up

The width of an inline box is given by its rendered content [p. 25] . Please consult the sections on box width calculations [p. 112] for details.

Block-level box heights are calculated bottom-up

Block level boxes grow to the size of the boxes they contain. Please consult the section on box height calculations [p. 115] for details.

Intrinsic dimension of replaced content

The rendered content [p. 25] of a replaced element [p. 26] may have "intrinsic dimensions" that user agents use as the computed [p. 57] content width and height (e.g., the unscaled width and height of an included image). If the intrinsic dimensions are overridden, the replaced content is scaled by the user agent. When scaling an image, the aspect ratio of the image is preserved if values for the 'width' and 'height' properties are set to 'auto'.

Hybrid calculations

The dimension of a table cell is determined by both the cell's contents and the surrounding available space.

The *box width* (resp., *box height*) is given by the sum of the content width (resp., content height), the padding, the border, and the margin.

If an inline element generates boxes over several lines (i.e., for more than one line box [p. 79]), the margin [p. 99] , border [p. 104] , and padding [p. 102] properties do not affect line height calculations (see the section on line height calculations [p. 118] for details). However, margin, border, and padding areas remain visible.

Note that there are no properties to set the color of margins and padding; margins are always transparent and padding areas always use the background of the element itself.

8.2.3 Example of margins, padding, and borders

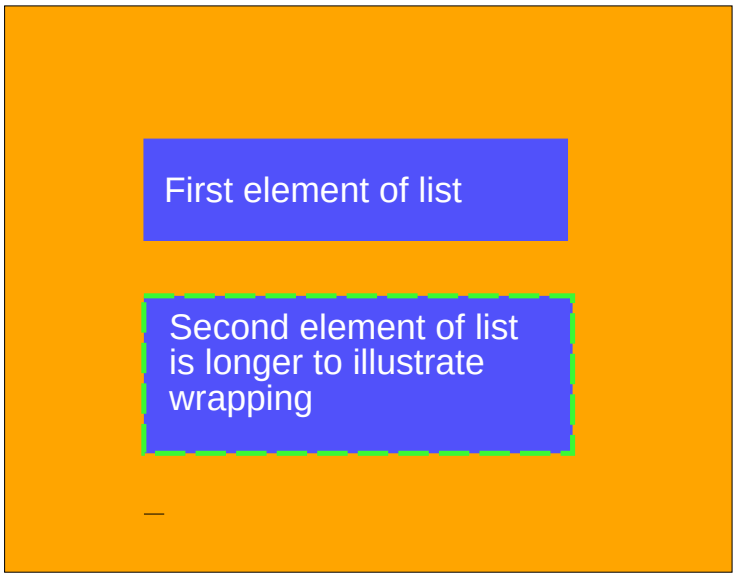
This example illustrates how margins, padding, and borders interact. The example HTML document:

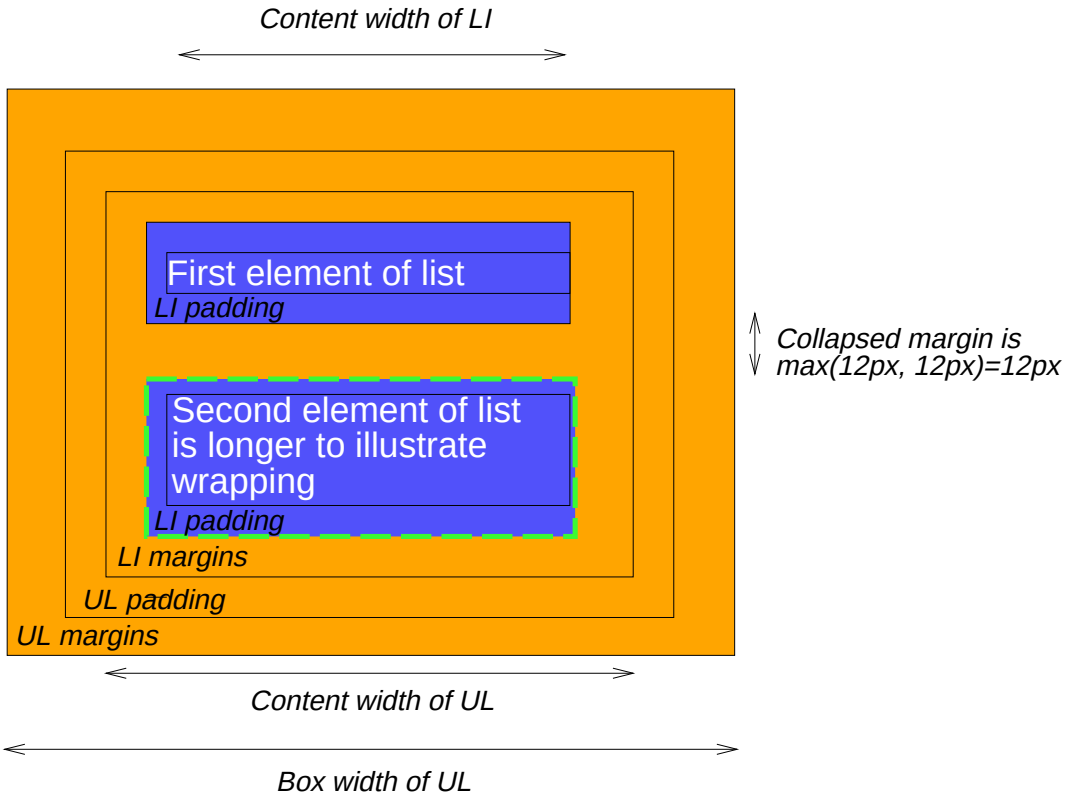
```
<HTML>
<HEAD>
<STYLE type="text/css">
  UL {
    background: orange;
    margin: 12px 12px 12px 12px;
    padding: 3px 3px 3px 3px;
                                  /* No borders set */
  }
  LI {
    color: white;                 /* text color is white */
    background: blue;             /* Content, padding will be blue */
    margin: 12px 12px 12px 12px;
    padding: 12px 0px 12px 12px; /* Note 0px padding right */
    list-style: none              /* no glyphs before a list item */
                                  /* No borders set */
  }
```

```
  LI.withborder {
    border-style: dashed;
    border-width: medium;        /* sets border width on all sides */
    border-color: green;
  }
</STYLE>
</HEAD>
<BODY>
<UL>
  <LI>First element of list
  <LI class="withborder">Second element of list is longer
       to illustrate wrapping.
</UL>
</BODY>
</HTML>
```

results in a document tree [p. 26] with (among other relationships) a UL element that has two LI children.

The first of the following diagrams illustrates what this example would produce. The second illustrates the relationship between the margins, padding, and borders of the UL elements and those of its children LI elements.





Note that:

- The content width [p. 72] for each LI box is calculated top-down; the containing block [p. 67] for the each LI box is the box generated by the UL element.
- The height of each LI box is given by its content height [p. 72] , plus padding, borders, and margins. Note that vertical margins between the LI boxes collapse. [p. 117]

- The right padding of the LI boxes has been set to zero width (the ’padding’ property). The effect is apparent in the second illustration.
- The foreground color of the LI boxes has been set to white for legibility against a blue background (the ’color’ property).
- The margins of the LI boxes are transparent -- margins are always transparent -- so the background color of the UL boxes (orange) shines through them. However, the (blue) background of the LI boxes is also used for the LI padding.
- The second LI element specifies a dashed border (the ’border-style’ property).

8.3 Positioning schemes

In CSS2, a box may be laid out according to three *positioning schemes*

1. Normal flow [p. 79] . The normal flow includes block formatting [p. 79] of block level [p. 69] elements, inline formatting [p. 79] of inline [p. 69] elements, relative positioning [p. 82] of block-level or inline elements, and positioning of compact and run-in [p. 70] boxes.
2. Floats [p. 82] . The floating model translates a box’s position to the left or right of where it would normally appear in the flow. For instance, authors may float paragraph boxes in order to place them side-by-side.
3. Absolute positioning [p. 85] . Authors may specify the absolute position of a box (with respect to a containing block).

The primary difference between a floating box and one that is absolutely positioned is that absolute positioning has no impact on the flow of later siblings; later siblings are laid out as though their absolutely positioned sister did not exist at all. Later siblings of floating boxes flow with respect to the final position of the floating box.

8.3.1 Choosing a positioning scheme: ’position’ property

The ’position’, and ’float’ properties determine which CSS2 positioning algorithms are used to calculate the coordinates of a box.

’position’

Property name:	’position’
Value:	normal | relative | absolute | fixed | inherit
Initial:	normal
Applies to:	elements that generate absolutely positioned and floated boxes
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

The values of this property have the following meanings:

’normal’
The box coordinates are calculated according to the normal flow [p. 79] .

’relative’
The box coordinates are calculated according to the normal flow [p. 79] , then the box is offset relative [p. 82] to its normal position. Note that the position of the following box [p. 67] is established independently of the offset.

’absolute’
The box coordinates (and possibly size) are calculated in absolute [p. 85] terms with respect to the box’s containing block [p. 67] .

’fixed’
The box coordinates are calculated according to the ’absolute’ model, but in addition, the box is fixed [p. 85] with respect to some reference. In the case of continuous media [p. 65] , the box is fixed with respect to the viewport [p. 68] (and doesn’t move when scrolled). In the case of paged media [p. 65] , the box is fixed with respect to the page. Fixed boxes are fixed with respect to pages that are seen through a viewport [p. 68] . **Note.** Authors may wish to specify ’fixed’ in a media-dependent way (e.g., information should appear on every screen but not on every page).

Note. *The value ’normal’ causes some user agents to skip [p. 31] the ’left’ and ’top’ properties. To ensure that values of ’left’ and ’top’ are taken into account, authors should explicitly set the value of the ’position’ property to ’relative’.*

8.3.2 Box offsets: ’top’, ’right’, ’bottom’, ’left’

The position of an relatively [p. 82] or absolutely [p. 85] (including fixed [p. 85]) positioned boxes is established by four properties:

’top’

Property name:	’top’
Value:	<length> | <percentage> | auto | inherit
Initial:	auto
Applies to:	all elements
Inherited:	no
Percentage values:	refer to height of containing block
Media groups:	visual [p. 65]

This property specifies how far a box’s top content edge is offset below the top edge of the box’s containing block [p. 67] .

’right’

Property name:	’right’
Value:	<length> | <percentage> | auto | inherit
Initial:	auto
Applies to:	all elements
Inherited:	no
Percentage values:	refer to width of containing block
Media groups:	visual [p. 65]

This property specifies how far a box's right content edge is offset to the left of the right edge of the box's containing block [p. 67] .

'bottom'

Property name:	'bottom'
Value:	<length> | <percentage> | auto | inherit
Initial:	auto
Applies to:	all elements
Inherited:	no
Percentage values:	refer to height of containing block
Media groups:	visual [p. 65]

This property specifies how far a box's bottom content edge is offset above the bottom of the box's containing block [p. 67] .

'left'

Property name:	'left'
Value:	<length> | <percentage> | auto | inherit
Initial:	auto
Applies to:	all elements
Inherited:	no
Percentage values:	refer to width of containing block
Media groups:	visual [p. 65]

This property specifies how far a box's left content edge is offset to the right of the left edge of the box's containing block [p. 67] .

The values for the four properties have the following meanings:

<length>
The offset is a fixed distance from the reference edge.

<percentage>
The offset is a percentage of the containing block's width (for 'left' or 'right') or height (for 'top' and 'bottom').

auto
The offset is automatically calculated based on the width and height of the box.

For absolutely positioned boxes, the offsets are with respect to the box's containing block [p. 67] . For relatively positioned boxes, the offsets are with respect to the outer edges of the box itself before the offset is applied.

For absolutely positioned boxes, the values of the 'left', 'right', 'top', and 'bottom' properties replace the roles of the corresponding margin properties [p. 99] (i.e., absolutely positioned boxes do not have margins but do have padding and borders).

For more information about the width and height of boxes, please consult the sections on box width calculations [p. 112] and box height calculations [p. 115] respectively.

8.4 Normal flow

Boxes in the normal flow belong to a formatting context, which may be block or inline, but not both simultaneously.

Block-level [p. 69] boxes participate in an block formatting [p. 79] context.

Inline [p. 69] boxes participate in an inline formatting [p. 79] context.

8.4.1 Anonymous boxes

Block-level elements whose rendered content [p. 25] contains text that is not the content of an inline element have *anonymous* children in the document tree [p. 26] . These anonymous elements inherit property values (colors, fonts, etc.). They generate boxes that contain chunks of text as content. By default, anonymous boxes act like inline boxes. However, they may act like block-level boxes if context demands (e.g., an inline run-in [p. 71] box that behaves like a block-level box). Decisions about the construction of anonymous inline boxes depend on many factors (language, hyphenation, etc.) and lie outside the scope of this specification.

8.4.2 Block formatting context

In a block formatting context, boxes are laid out one after the other, vertically. The vertical distance between two sibling boxes is determined by the 'margin' properties. Vertical margins between adjacent block-level boxes collapse [p. 117] .

To lay out boxes horizontally in CSS2, authors may declare them to be inline [p. 69] , or position them (floating [p. 82] or absolute [p. 85] positioning).

For information about page breaks in paged media, please consult the section on allowed page breaks [p. 138] .

8.4.3 Inline formatting context

In an inline formatting context, boxes are laid out horizontally, one after the other. Horizontal margins, borders, and padding are respected between these boxes. They may be aligned vertically in different ways: their bottoms or tops may be aligned, or the baselines of text within them may be aligned. The rectangular area that contains the boxes that form a line is called a *line box*.

The width of a line box is determined by a containing block [p. 67] . The height of a line box is determined by the rules given in the section on line height calculations [p. 118] . A line box is always tall enough for all of the boxes it contains. However, it may be taller than the tallest box it contains (if, for example, boxes are aligned so that baselines line up). When the height of a box B is less than the height of the line box containing it, the vertical alignment of B within the line box is determined by the 'vertical-align' property.

When several inline boxes cannot fit within a single line box, they are distributed among two or more vertically-stacked line boxes. Thus, a paragraph is a vertical stack of line boxes. Line boxes are stacked with no vertical separation and they never overlap.

Line boxes in the same inline formatting context generally have the same width (that of the containing block) but may vary in width if available horizontal space is reduced due to floats [p. 82] . Line boxes in the same inline formatting context generally vary in height (e.g., one line might contain an image while the others contain only text). When a box is less wide than the width of the line box containing it, its horizontal alignment within the line box is determined by the 'text-align' property. When a box is wider than a line box, it may be split into several boxes and these boxes distributed across several line boxes. When a box is split, margins, borders, and padding have no visual effect at the end of the first line box or at the beginning of the next line box.

For example, the following paragraph (created by the HTML block-level element P) contains anonymous text interspersed with the elements EM and STRONG:

```
<P>Several <EM>emphasized words</EM> appear
<STRONG>in this</STRONG> sentence, dear.</P>
```

In terms of the document tree, P has five children:

- Anonymous: "Several"
- EM: "emphasized words"
- Anonymous: "appear"
- STRONG: "in this"
- Anonymous: "sentence, dear."

To format the paragraph, the user agent creates a box for each child and flows the boxes into line boxes. Since the parent box in normal flow acts as the containing block for an inline box, the width of the P box determines the width of these line boxes. If the width of P is sufficient, all the inline boxes will fit into a single line box:

```
Several emphasized words appear in this sentence, dear.
```

If the boxes do not fit within a single line box, they will be split up and distributed across several line boxes. The previous paragraph might be split as follows:

```
Several emphasized words appear
in this sentence, dear.
```

or like this:

```
Several emphasized
words appear in this
sentence, dear.
```

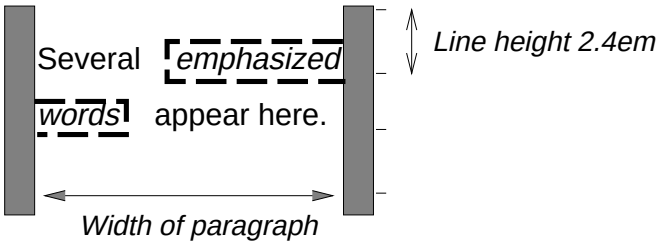
In the previous example, the EM box was split into two EM boxes (call them "split1" and "split2"). Therefore, margins, borders, padding, or text decorations have no visible effect after split1 or before split2.

Consider the following example:

```
<HTML>
<HEAD>
<TITLE>Example of inline flow on several lines</TITLE>
<STYLE type="text/css">
EM {
```

```
  padding: 2px ;
  margin: 1em ;
  border-width: medium;
  border-style: dashed;
  line-height: 2.4em;
}
</STYLE>
</HEAD>
<BODY>
<P>Several <EM>emphasized words</EM> appear here.</P>
</BODY>
</HTML>
```

Depending on the width of the P, the boxes may be distributed as follows:



- The margin is inserted before "emphasized" and after "words".
- The padding is inserted before, above, and below "emphasized" and after, above, and below "words". A dashed border is rendered on three sides in each case.

Note that with a small line height, the padding and borders around text in different lines may overlap.

8.4.4 Direction of inline flow: the ’direction’ property

’direction’

Property name:	’direction’
Value:	ltr | rtl | ltr-override | rtl-override | inherit
Initial:	ltr
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

This property determines whether the direction of flow in an inline formatting context [p. 79] is left-to-right or right-to-left. It also specifies the direction of table layout [p. 201] .

Values have the following meanings:

ltr
: Left-to-right flow.

rtl
: Right-to-left flow.

ltr-override
Left-to-right flow, overriding the [UNICODE] [p. 268] bidirectional algorithm.
rtl-override
Right-to-left flow, overriding the [UNICODE] [p. 268] bidirectional algorithm.

For a left-to-right inline formatting context [p. 79] , the horizontal distance between the right side of a box and the left side of the following box [p. 67] (or right side of the parent box if no following box exists) is determined by the source element's 'margin' properties. For right-to-left flow, the horizontal distance between the left side of a box and the right side of the preceding box [p. 67] (or left side of the parent box is no preceding box exists) is similarly determined by the source element's 'margin' properties.

8.4.5 Relative positioning

Once a box has been assigned a position according to the normal flow [p. 79] , it may be shifted relative to this position. This is called *relative positioning*. Offsetting a box in this way has no effect on the following box: it is positioned as if the preceding box were not offset and it is not repositioned after the offset is applied. This implies that relative positioning may cause boxes to overlap.

Relatively positioned boxes keep their normal flow size, including line breaks and the space originally reserved for them. A relatively positioned box establishes a new containing block [p. 67] for descendant boxes.

A relatively positioned box is generated when the 'position' property for an element has the value 'relative'. The offset is specified by the 'top', 'bottom', 'left', and 'right' properties.

Dynamic movement of relatively positioned boxes can produce animation effects in scripting environments (see the 'visibility' property). Relative positioning may also be used as a general form of superscripting and subscripting except that line height is not automatically adjusted to take the positioning into consideration. See the description of line height calculations [p. 118] for more information.

Examples of relative positioning are provided in the section comparing normal, relative, floating, and absolute positioning [p. 87] .

8.5 Floats: the 'float' and 'clear' properties

A floating box has two interesting qualities:

1. It may be positioned at the right or left edge of a containing block (the 'float' property).
2. It may or may not allow content to flow next to it (the 'clear' property).

'float'

Property name: 'float'
Value: left | right | none | inherit
Initial: none
Applies to: elements that are not positioned absolutely
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

This property specifies whether a box should float to the left, right, or not at all. It may be set for elements that generate boxes that are not positioned absolutely (including fixed boxes). A floating box is given a position and height according to the normal flow [p. 79] (the computed value [p. 57] of the 'width' is '0' unless assigned explicitly), then taken out of the flow and shifted to the left or right until its outer edge [p. 72] is flush with the current left or right edge (which may be established by a containing block [p. 67] or another floated box).

User agents take the outer edge of a floating box into account when positioning subsequent boxes; the outer edge of a float becomes the current edge for flowed or floated boxes to the left or right side. The margins of floating boxes never collapse [p. 117] with margins of adjacent boxes.

The values of this property have the following meanings:

left
Makes the generated box a block-level [p. 69] box (i.e., 'display' is set to 'block') in the normal flow [p. 79] , then shifts the box to the left. Text wraps on the right side of the box.

right
Makes the generated box a block-level [p. 69] box (i.e., 'display' is set to 'block') in the normal flow [p. 79] , then shifts the box to the right. Text wraps on the left side of the box.

none
Has no effect on the generated box.

The following rule positions all IMG elements with `class="icon"` along the left side of the parent element:

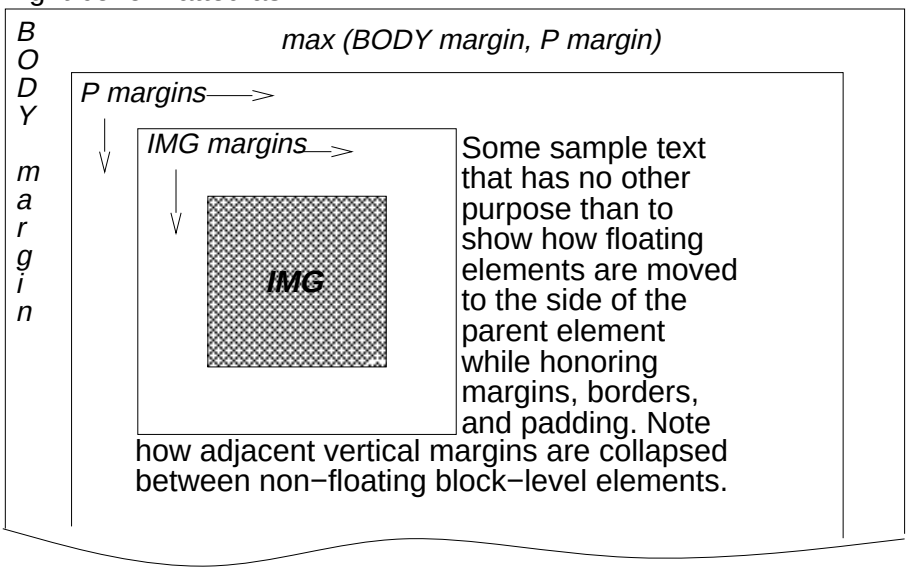
```
IMG.icon {
  float: left;
  margin-left: 0;
}
```

The following HTML source:

```
<HTML>
<HEAD>
<STYLE type="text/css">
  IMG { float: left }
  BODY, P, IMG { margin: 2em }
</STYLE>
</HEAD>
<BODY>
  <P>
```

```
    <IMG src=img.gif alt="This image will illustrate floats">
    Some sample text that has no other...
</BODY>
</HTML>
```

might be formatted as:



Note that the margin of the P element encloses the floating IMG element and that the vertical margins do not collapse.

8.5.1 Controlling flow next to floats

'clear'

Property name:	'clear'
Value:	none | left | right | both | inherit
Initial:	none
Applies to:	block-level elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

When set for an element generating a box B, this property indicates which sides of B may **not** be adjacent to a floating box. This property may only be specified for block-level [p. 69] elements. For compact and run-in boxes [p. 70] , this property applies to the final block-level box to which the compact or run-in box belongs.

Values have the following meanings:

left
The generated box is moved below any floating box to its left.

right
The generated box is moved below any floating box to its right.

both
The generated box is moved below any floating box to its left or right.

none
The box may be placed next to floating boxes to the left or right.

The following style rule means that no H1 element may have a floating element to its left; this means that H1 elements will be positioned below any floating box.

```
H1 { clear: left }
```

A floating box B is subject to several constraints:

- Once floated, the margin of B is increased enough so that its border top is positioned at or below the margin bottom of the bottom-most float it clears.
- The floats cleared by B must be generated by elements that precede the element in the document tree [p. 26] that generated B.

Please consult the section on floating constraints [p. 121] for additional constraints.

8.6 Absolute positioning

Like other boxes, an absolutely positioned element is positioned with respect to a containing block [p. 67] . It also establishes a new containing block for descendant boxes. However, the contents of an absolutely positioned element do not flow around any other boxes. They may or may not obscure the contents of another box, depending on the z-order [p. 96] of the overlapping boxes.

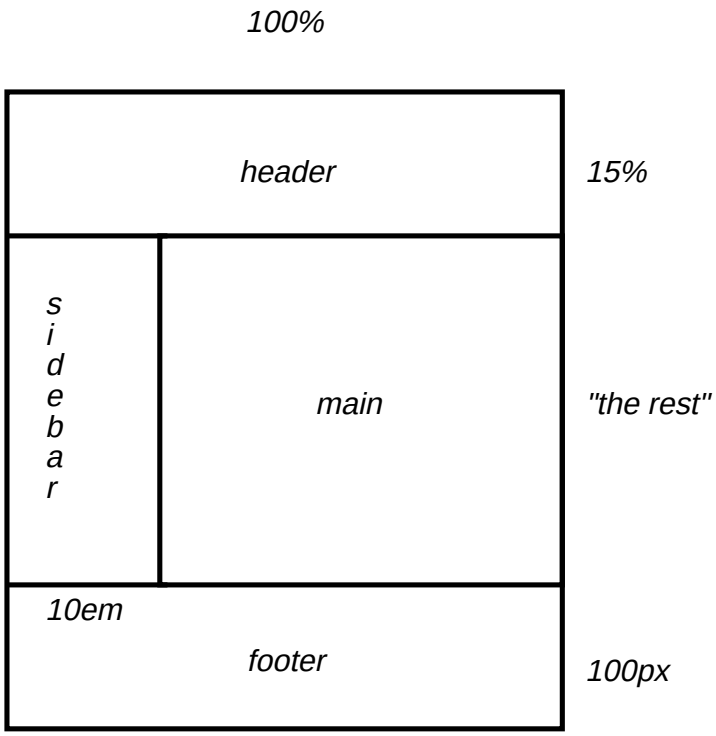
8.6.1 Fixed positioning

Fixed positioning is a variant of absolute positioning. The only difference is that for a fixed positioned box, the containing block is established by the viewport [p. 68] .

For continuous media [p. 65] , fixed boxes do not move when the document is scrolled. In this respect, they are similar to fixed background images [p. 142] .

For paged media [p. 131] , boxes with fixed positions are repeated on every page. This is useful for placing, for instance, a signature at the bottom of each page.

Authors may use fixed positioning to create frame-like presentations. Consider the following frame layout:



This might be achieved with the following HTML document and style rules:

```
<HTML>
<HEAD>
<TITLE>A frame document with CSS2</TITLE>
<STYLE type="text/css">
  #header {
     position: fixed;
     width: 100%;
     height: 15%;
     top: 0;
     right: 0;
     bottom: auto;
     left: 0;
  }
  #sidebar {
     position: fixed;
     width: 10em;
     height: auto;
     top: 15%;
     right: auto;
     bottom: 100px;
     left: 0;
  }
  #main {
     position: fixed;
     width: auto;
     height: auto;
     top: 15%;
     right: 0;
     bottom: 100px;
     left: 10em;
  }
  #footer {
     position: fixed;
     width: 100%;
```

```
      height: 100px;
      top: auto;
      right: 0;
      bottom: 0;
      left: 0;
    }
  </STYLE>
  </HEAD>
  <BODY>
    <DIV id="header"> ...  </DIV>
    <DIV id="sidebar"> ...  </DIV>
    <DIV id="main"> ...  </DIV>
    <DIV id="footer"> ...  </DIV>
  </BODY>
  </HTML>
```

8.7 Relationships between ’display’, ’position’, and ’float’

When specified for the same elements, the three properties that affect box generation and layout -- ’display’, ’position’, and ’float’ -- interact according to the following precedences (highest to lowest):

1. If ’display’ has the value ’none’, user agents must skip [p. 31] ’position’ and ’float’. In this case, the element generates no box.
2. Otherwise, if ’position’ has the value ’absolute’ or ’fixed’, the generated box will be non-floating and block-level. The position of the box will be determined by the ’top’, ’right’, ’bottom’ and ’left’ properties and the containing block.
3. Otherwise, if ’float’ has a value other than ’none’, the generated box is a block-level box and is floated.
4. Otherwise, the remaining ’display’ properties apply as specified.

Note on scripting and layout. *CSS2 does not specify layout behavior when values for these properties are changed by scripts. For example, what happens when an element having ’width: auto’ is repositioned? Do the contents reflow, or do they maintain their original formatting? The answer is outside the scope of this document, and such behavior is likely to differ in initial implementations of CSS2.*

8.8 Comparison of normal, relative, floating, absolute positioning

To illustrate the relationship between normal flow, relative positioning, floats, and absolute positioning, we provide a series of examples based on the following HTML fragment:

```
<HTML>
<HEAD>
<TITLE>Comparison of positioning schemes</TITLE>
</HEAD>
<BODY>
  <P>Beginning of body contents.
  <SPAN id="outer"> Start of outer contents.
```

```
  <SPAN id="inner"> Inner contents.</SPAN>
  End of outer contents.</SPAN>
  End of body contents.
  </P>
</BODY>
</HTML>
```

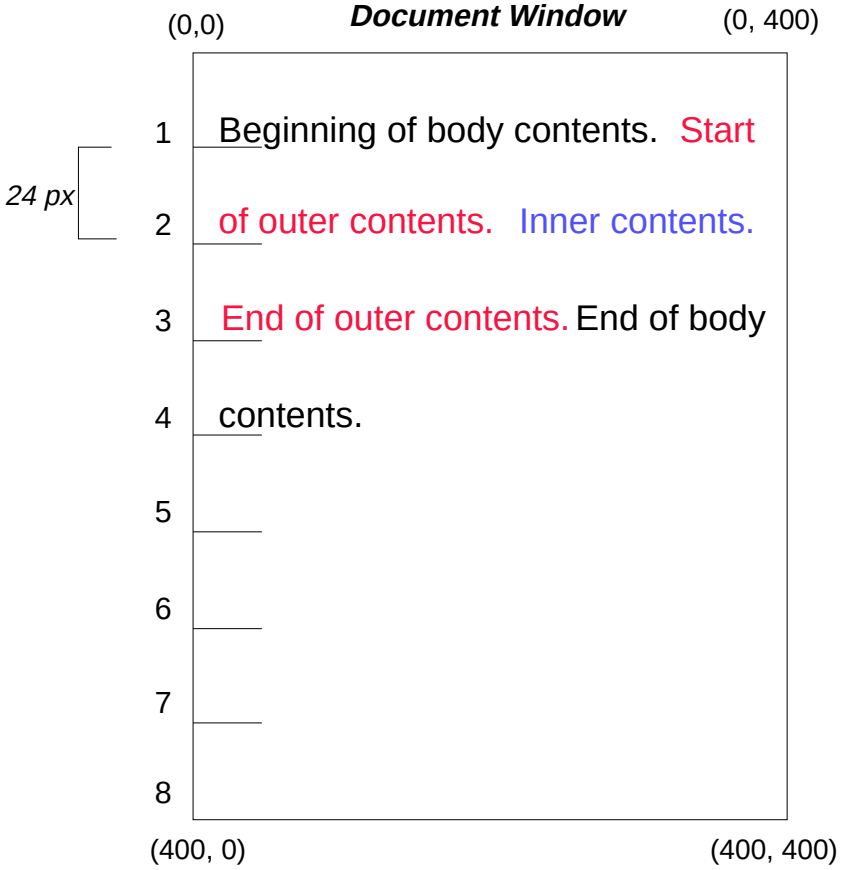
The final positions of boxes generated by the *outer* and *inner* elements vary in each example. In each illustration, the numbers to the left of the illustration indicate the normal flow [p. 79] position of the double-spaced (for clarity) lines.

8.8.1 Normal flow

Consider the following CSS declarations for *outer* and *inner* that don't alter the normal flow [p. 79] of boxes:

```
#outer { color: red }
#inner { color: blue }
```

This results in something like the following:



8.8.2 Relative positioning

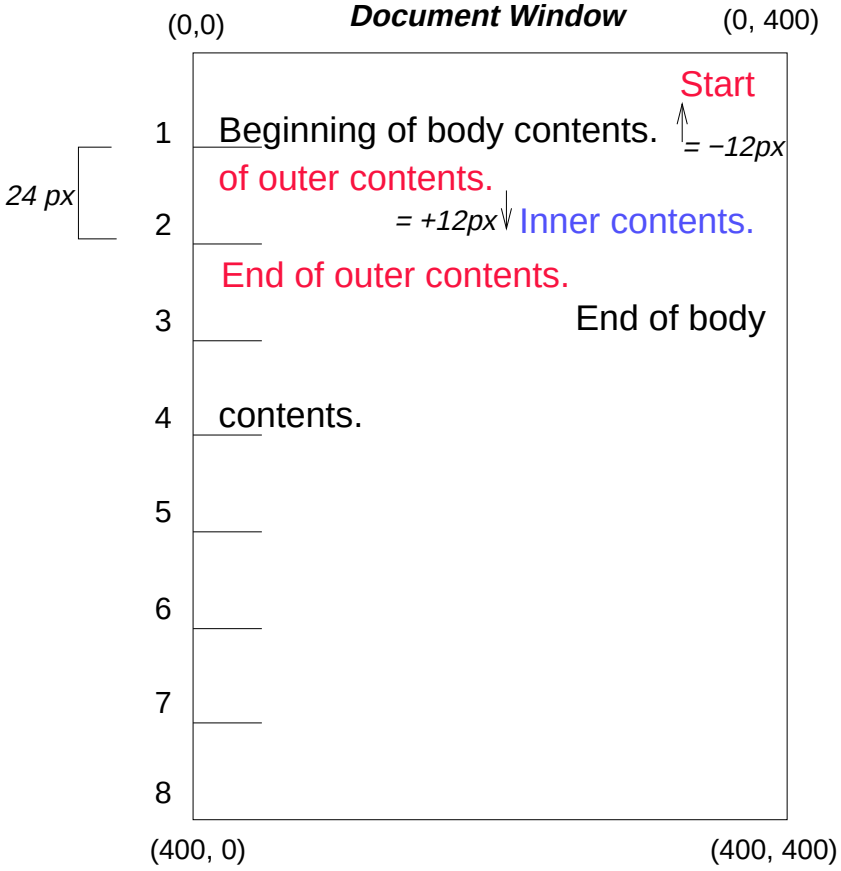
To see the effect of relative positioning [p. 82] , consider the following CSS rules:

```
BODY { line-height: 200% }
#outer { position: relative; top: -12px; color: red }
#inner { position: relative; top: 12px; color: blue }
```

First, the *outer* text is flowed into its normal flow position and dimensions at the end of line 1. Then, the entire box (distributed over three lines) is shifted upwards by 12px.

The contents of *inner*, as a child of *outer*, would normally flow immediately after the words "of outer contents" (on line 1.5). However, the *inner* contents are themselves offset relative to the *outer* contents by 12px downwards, back to their original position on line 2.

Note that the content following *outer* is not affected by the relative positioning of *outer*.



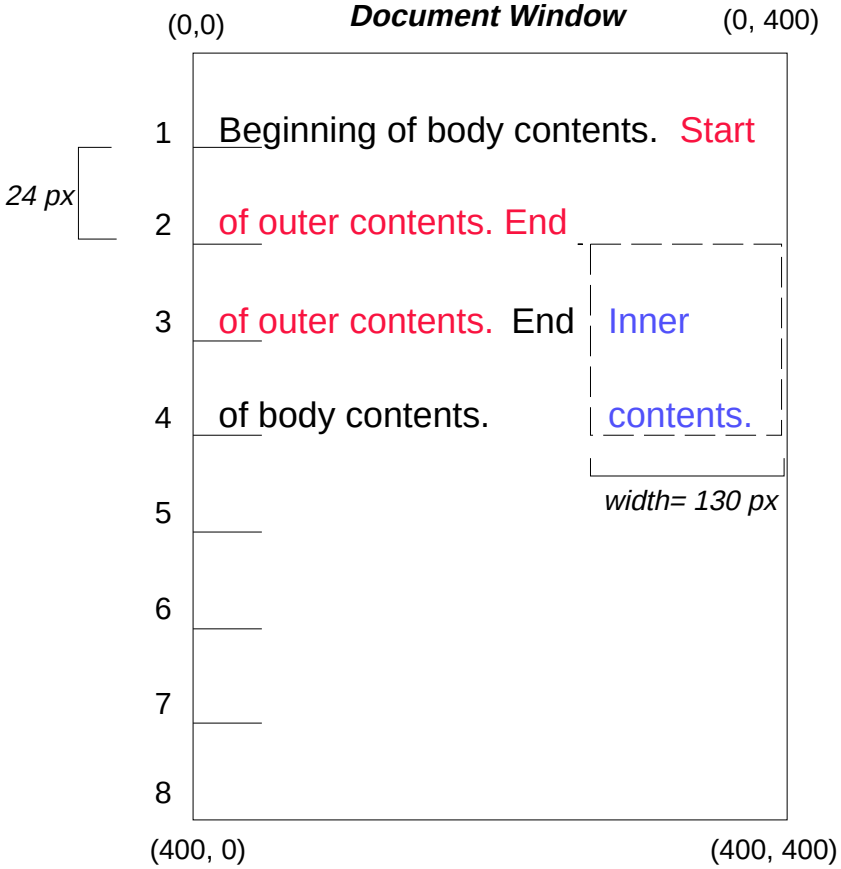
Note also that if the relative positioning of *outer* were -24px, the text of *outer* and the body text would have overlapped.

8.8.3 Floating a box

Now consider the effect of floating [p. 82] the *inner* element's text to the right by means of the following rules:

```
#outer { color: red }
#inner { float: right; width: 130px; color: blue }
```

First, the *inner* box (whose width has been set explicitly) is floated to the right margin. Then, the text of the *outer* element that follows the inner element text flows in the space vacated by the *inner* box. This flow respects the new right margin established by the left border of the *inner* box.



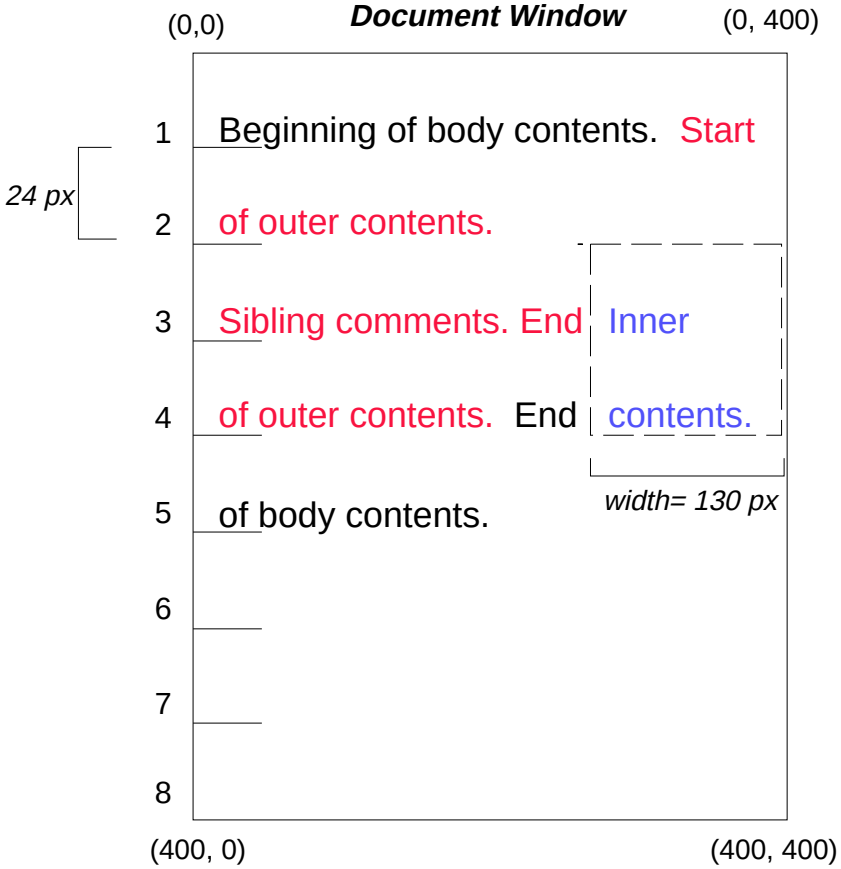
To show the effect of the 'clear' property, we add a *sibling* element to the example:

```
<HTML>
<HEAD>
<TITLE>Comparison of positioning schemes II</TITLE>
</HEAD>
<BODY>
  <P>Beginning of body contents.
  <SPAN id=outer> Start of outer contents.
  <SPAN id=inner> Inner contents.</SPAN>
  <SPAN id=sibling> Sibling contents.</SPAN>
  End of outer contents.</SPAN>
  End of body contents.
  </P>
</BODY>
</HTML>
```

The following rules:

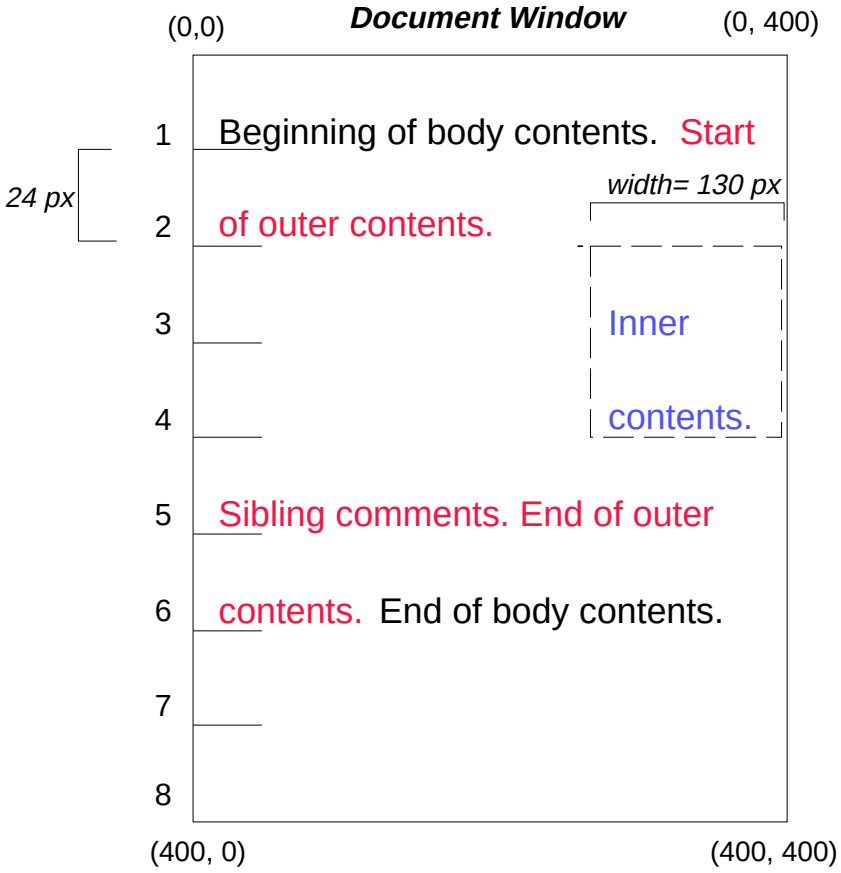
```
  #inner { float: right; width: 130px; color: blue }
  #sibling { color: red }
```

cause the *inner* box to float to the right and the text of the *sibling* element to flow in the vacated space:



However, if the 'clear' property on the *sibling* element is set to 'right' (i.e., the generated *sibling* box will not accept being positioned next to floating boxes to its right), the *sibling* box is moved below the float:

```
#inner { float: right; width: 130px; color: blue }
#sibling { clear: right; color: red }
```

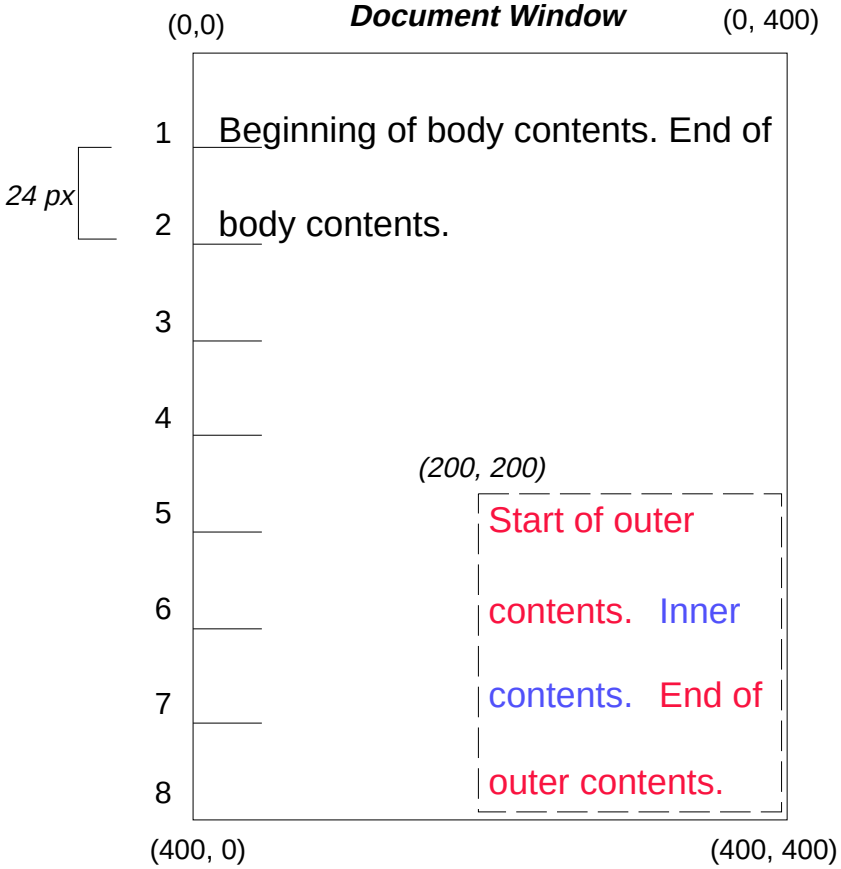


8.8.4 Absolute positioning

Finally, we consider the effect of absolute positioning [p. 85] . Consider the following CSS declarations for *outer* and *inner*:

```
#outer { position: absolute;
         top: 200px;
         left: 200px;
         width: 200px;
         color: red }
#inner { color: blue }
```

which cause the top of the *outer* box to be positioned with respect to the containing block (which we suppose is established by the initial containing block [p. 111]). The top side of the *outer* box is 200px from the top of the containing block and the left side is 200px from the left side. The child box of *outer* is flowed normally with respect to its parent.

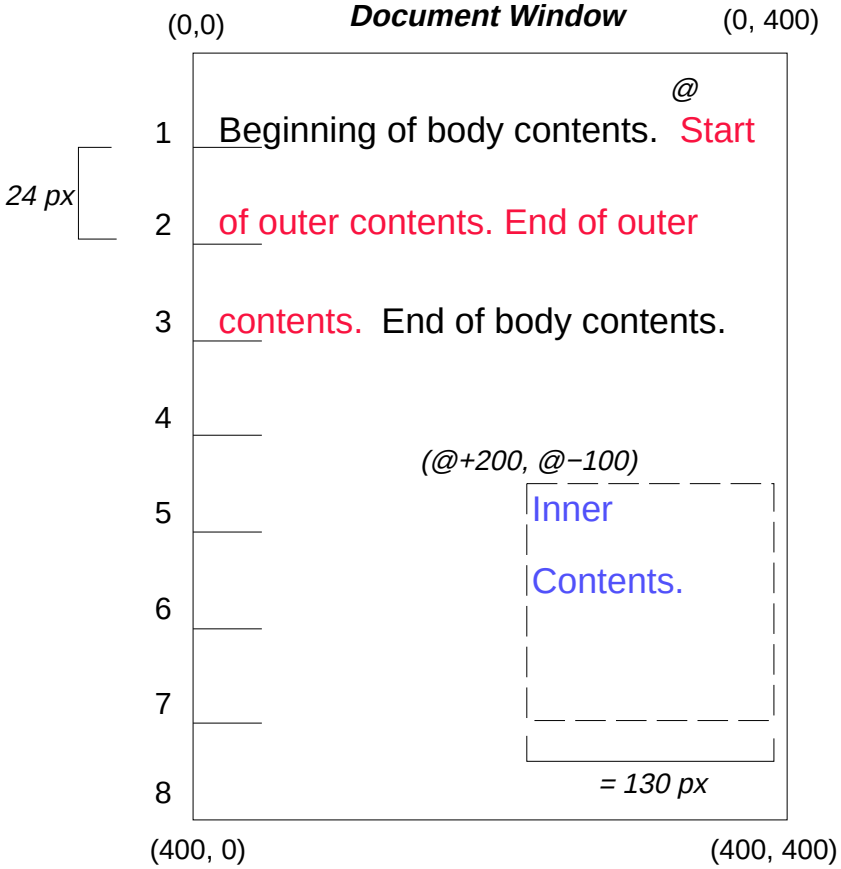


Note that because *outer* has been absolutely positioned, it establishes a new containing block for descendant boxes (there aren't any in this example).

The following example shows an absolutely positioned box that is a child of a relatively positioned box. Although the parent *outer* box is not actually offset, setting its 'position' property to 'relative' causes its box to serve as the containing block for any descendant boxes. Since the *outer* box is an inline box that is split across several lines, only the first box (whose upper left-hand corner is designated by a "@" in the illustration below) establishes the containing block for the descendants.

```
#outer { position: relative; color: red }
#inner { position: absolute;
         top: 200px;
         left: -100px;
         height: 130px;
         width: 130px;
         color: blue }
```

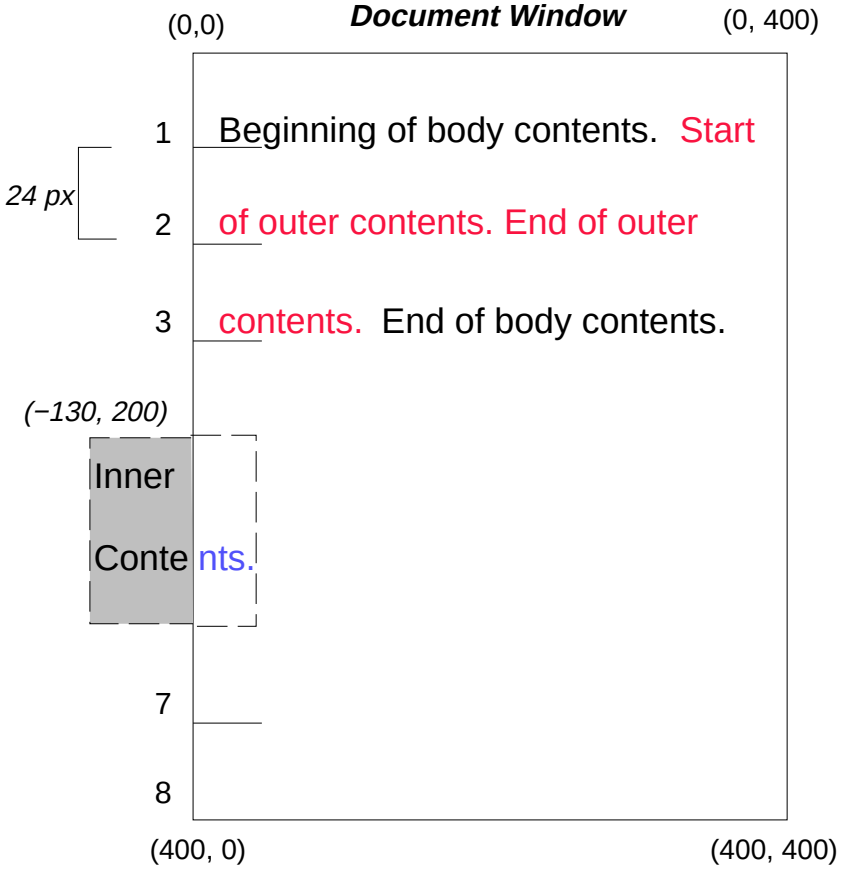
This results in something like the following:



The following rules don't establish a new positioning context for *inner*:

```
#outer { color: red }
#inner {position: absolute; top: 200px; left: -100px; height:
  130px; width: 130px; color: blue;}
```

but cause the *inner* box to be positioned with respect to the containing block (which we assume here is initial containing block [p. 111]).

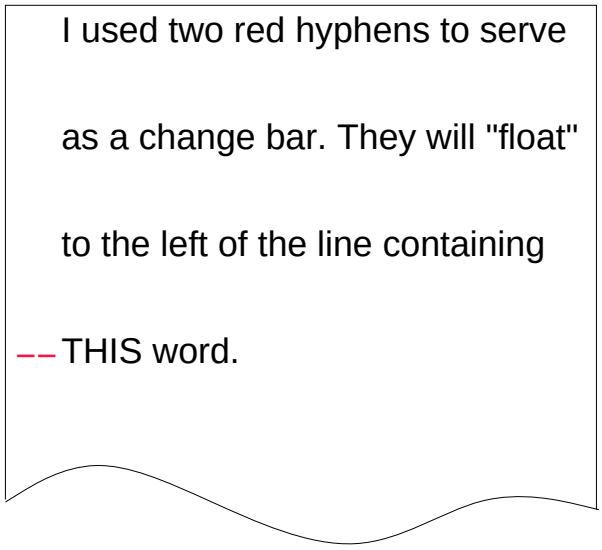


Relative and absolute positioning may be used to implement change bars, as shown in the following example. We use a value of 'auto' for the 'top' property, which results in the box being placed at the "current" location, just as if the box were being flowed into that space. The following HTML text:

```
<P style="position: relative; margin-right: 10px; left: 10px;">
I used two red hyphens to serve as a change bar. They
will "float" to the left of the line containing THIS
<SPAN style="position: absolute; top: auto; left: 0px; color: red;">--</SPAN>
word.</P>
```

might result in something like:

Document Window



8.9 Z-order: Layered presentation

In the following sections, the expression "in front of" means closer to the user as the user faces the screen.

CSS allows authors to specify the position of a box in three dimensions. The *stack level* of an element refers to the position of boxes it generates in front of or behind other boxes. The stack level is particularly relevant to boxes that overlap visually.

The stack level of an element may be determined in two ways:

- Implicitly, by virtue of the element's position in the document tree [p. 26] . Boxes are stacked in the order their source elements appear in the document tree. Thus, a box is stacked in front of its parent and preceding siblings and behind its children and following siblings.
- Explicitly, via the 'z-index' property.

8.9.1 Specifying the stack level: the 'z-index' property

'z-index'

Property name:	'z-index'
Value:	auto | <integer> | inherit
Initial:	auto
Applies to:	elements that generate absolutely and relatively positioned boxes
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

The ’z-index’ property is used to specify the stacking order of boxes outside the normal flow [p. 79] . An element for which this property is set establishes a *stacking context* in which its own ’z-index’ is 0.

The values of this property have the following meanings:

auto
: The stack level of the generated box is given by the element’s position in the document tree [p. 26] .

<integer>
: Specifies an explicit stack level for the generated box.

Elements for which the ’z-index’ property has been given an integer value behave as follows:

- Sibling boxes are stacked bottom-to-top in order of increasing ’z-index’ value. When two siblings have the same ’z-index’ value, the later sibling generates boxes in front of those generated by the earlier sibling.
- Elements with negative ’z-index’ values generate boxes behind level 0 boxes in the current stacking context. Elements with positive ’z-index’ values generate boxes in front of level 0 boxes in the current stacking context.
- A ’z-index’ value of 0 is equivalent to a value of ’auto’.

The relative z-order of two elements that are neither siblings nor parent/child can be determined by evaluation of the previous rules for both elements’ ancestors.

By default, an element will be placed just in front of its parent.

In the following example, the order of the elements, listed back-to-front is: "image", "text2", and "text1".

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Z-order positioning</TITLE>
<STYLE type="text/css">
.pile { position: absolute; left: 2in; top: 2in; width: 3in; height: 3in; }
</STYLE>
</HEAD>
<BODY>
<P>
<IMG alt="A butterfly image"
     src="butterfly.gif"
     class="pile"
     id="image"
     style="z-index: 1">

<DIV class="pile" id="text1" style="z-index: 3">
   This text will overlay the butterfly image.
</DIV>

<DIV class="pile" id="text2" style="z-index: 2">
   This text will underlay text1, but overlay the butterfly image
</DIV>
</BODY>
</HTML>
```

This example demonstrates the notion of *transparency*. The default behavior of a box is to allow boxes behind it to be visible through transparent areas in its content. In the example, each box transparently overlays the boxes below it. This behavior can be overridden by using one of the existing background properties [p. 142] .

9 Visual rendering model details

Contents

1. Margin, border, and padding properties [p. 99]
    1. Margin properties [p. 99] : ’margin-top’, ’margin-right’, ’margin-bottom’, ’margin-left’, and ’margin’
    2. Padding properties [p. 102] : ’padding-top’, ’padding-right’, ’padding-bottom’, ’padding-left’, and ’padding’
    3. Border properties [p. 104]
        1. Border width [p. 104] : ’border-top-width’, ’border-right-width’, ’border-bottom-width’, ’border-left-width’, and ’border-width’
        2. Border color [p. 106] : ’border-top-color’, ’border-right-color’, ’border-bottom-color’, ’border-left-color’, and ’border-color’
        3. Border style [p. 107] : ’border-top-style’, ’border-right-style’, ’border-bottom-style’, ’border-left-style’, and ’border-style’
        4. Border shorthand properties [p. 109] : ’border-top’, ’border-bottom’, ’border-right’, ’border-left’, and ’border’
2. Containing blocks [p. 111]
3. Box width calculations [p. 112]
    1. Content width [p. 112] : the ’width’ property
    2. Widths of boxes in the normal flow and floated boxes [p. 113]
        1. Determining the content width [p. 113]
        2. Computing margin widths [p. 114]
    3. Width of absolutely positioned elements [p. 114]
    4. Minimum and maximum widths [p. 114] : ’min-width’ and ’max-width’
4. Box height calculations [p. 115]
    1. Content height [p. 115] : the ’height’ property
    2. Determining the content height [p. 116]
    3. Height of absolutely positioned elements [p. 116]
    4. Minimum and maximum heights [p. 116] : ’min-height’ and ’max-height’
5. Collapsing margins [p. 117]
6. Line height calculations [p. 118] : the ’line-height’ and ’vertical-align’ properties
    1. Leading and half-leading [p. 118]
7. Floating constraints [p. 121]

9.1 Margin, border, and padding properties

The following sections include the definitions of box properties and other details about box generation.

9.1.1 Margin properties: ’margin-top’, ’margin-right’, ’margin-bottom’, ’margin-left’, and ’margin’

Margin properties specify the width of the margin area [p. 71] of a box. The ’margin’ shorthand property sets the margin for all four sides while the other margin properties only set their respective side. These properties apply to all elements generating block-level [p. 69] boxes \ that are not absolutely positioned [p. 85] and they apply to elements for which ’display’ is set to ’caption’.

The properties defined in this section refer to the **<margin-width>** value type, which may take one of the following values:

<length>
: Specifies a fixed width.

<percentage>
: Specifies a percentage width. The percentage is calculated with respect to the generated box’s containing block [p. 111] .

auto
: See the section on box width calculations [p. 112] for behavior.

Negative values for margin properties are allowed, but there may be implementation-specific limits. Percentage values for margin properties refer to the width of the generated box’s containing block [p. 111] .

’margin-top’

Property name:	’margin-top’
Value:	<margin-width> | inherit
Initial:	0
Applies to:	all elements
Inherited:	no
Percentage values:	refer to width of containing block
Media groups:	visual [p. 65]

This property sets the top margin of a generated box.

```
H1 { margin-top: 2em }
```

’margin-right’

Property name:	’margin-right’
Value:	<margin-width> | inherit
Initial:	0
Applies to:	all elements
Inherited:	no
Percentage values:	refer to width of containing block
Media groups:	visual [p. 65]

This property sets the right margin of a generated box.

```
H1 { margin-right: 12.3% }
```

’margin-bottom’

Property name: ’margin-bottom’
Value: <margin-width> | inherit
Initial: 0
Applies to: all elements
Inherited: no
Percentage values: refer to width of containing block
Media groups: visual [p. 65]

This property sets the bottom margin of a generated box.

```
H1 { margin-bottom: 3px }
```

’margin-left’

Property name: ’margin-left’
Value: <margin-width> | inherit
Initial: 0
Applies to: all elements
Inherited: no
Percentage values: refer to width of containing block
Media groups: visual [p. 65]

This property sets the left margin of a generated box.

```
H1 { margin-left: 2em }
```

’margin’

Property name: ’margin’
Value: <margin-width>{1,4} | inherit
Initial: not defined for shorthand properties
Applies to: all elements
Inherited: no
Percentage values: refer to width of containing block
Media groups: visual [p. 65]

The ’margin’ property is a shorthand property for setting ’margin-top’, ’margin-right’ ’margin-bottom’ and ’margin-left’ at the same place in the style sheet.

If there is only one value, it applies to all sides. If there are two values, the top and bottom margins are set to the first value and the right and left margins are set to the second. If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third. If there are four values, they apply to the top, right, bottom, and left, respectively.

```
BODY { margin: 2em } /* all margins set to 2em */
BODY { margin: 1em 2em } /* top & bottom = 1em, right & left = 2em */
BODY { margin: 1em 2em 3em } /* top=1em, right=2em, bottom=3em, left=2em */
```

The last rule of the example above is equivalent to the example below:

```
BODY {
  margin-top: 1em;
  margin-right: 2em;
  margin-bottom: 3em;
  margin-left: 2em;        /* copied from opposite side (right) */
}
```

9.1.2 Padding properties: ’padding-top’, ’padding-right’, ’padding-bottom’, ’padding-left’, and ’padding’

The padding properties specify the width of the content area [p. 71] of a box. The ’padding’ shorthand property sets the padding for all four sides while the other padding properties only set their respective side. These properties apply to all elements.

The properties defined in this section refer to the **<padding-width>** value type, which may take one of the following values:

<length>
: Specifies a fixed width.

<percentage>
: Specifies a percentage width. The percentage is calculated with respect to the generated box’s containing block [p. 111] .

Unlike margin properties, values for padding values cannot be negative. Like margin properties, percentage values for padding properties refer to the width of the generated box’s containing block.

’padding-top’

Property name:	’padding-top’
Value:	<padding-width> | inherit
Initial:	0
Applies to:	all elements
Inherited:	no
Percentage values:	refer to width of containing block
Media groups:	visual [p. 65]

This property sets the top padding of a generated box.

```
BLOCKQUOTE { padding-top: 0.3em }
```

’padding-right’

Property name:	’padding-right’
Value:	<padding-width> | inherit
Initial:	0
Applies to:	all elements
Inherited:	no
Percentage values:	refer to width of containing block
Media groups:	visual [p. 65]

This property sets the right padding of a generated box.

```
BLOCKQUOTE { padding-right: 10px }
```

’padding-bottom’

Property name:	’padding-bottom’
Value:	<padding-width> | inherit
Initial:	0
Applies to:	all elements
Inherited:	no
Percentage values:	refer to width of containing block
Media groups:	visual [p. 65]

This property sets the bottom padding of a generated box.

```
BLOCKQUOTE { padding-bottom: 2em }
```

’padding-left’

Property name:	’padding-left’
Value:	<padding-width> | inherit
Initial:	0
Applies to:	all elements
Inherited:	no
Percentage values:	refer to width of containing block
Media groups:	visual [p. 65]

This property sets the left padding of a generated box.

```
BLOCKQUOTE { padding-left: 20% }
```

’padding’

Property name:	’padding’
Value:	<padding-width>{1,4} | inherit
Initial:	not defined for shorthand properties
Applies to:	all elements
Inherited:	no
Percentage values:	refer to width of containing block
Media groups:	visual [p. 65]

The ’padding’ property is a shorthand property for setting ’padding-top’, ’padding-right’, ’padding-bottom’, and ’padding-left’ at the same place in the style sheet.

If there is only one value, it applies to all sides. If there are two values, the top and bottom paddings are set to the first value and the right and left paddings are set to the second. If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third. If there are four values, they apply to the top, right, bottom, and left, respectively.

The surface color or image of the padding area is specified via the ’background’ property:

```
H1 {
  background: white;
  padding: 1em 2em;
}
```

The example above specifies a ’1em’ vertical padding (’padding-top’ and ’padding-bottom’) and a ’2em’ horizontal padding (’padding-right’ and ’padding-left’). The ’em’ unit is relative [p. 36] to the element’s font size: ’1em’ is equal to the size of the font in use.

9.1.3 Border properties

The border properties specify the width, color, and style of the border area [p. 71] of a box. These properties apply to all elements.

Border width: ’border-top-width’, ’border-right-width’, ’border-bottom-width’, ’border-left-width’, and ’border-width’

The border width properties specify the width of the border area [p. 71] . The properties defined in this section refer to the **<border-width>** value type, which may take one of the following values:

thin
: A thin border.

medium
: A medium border.

thick
: A thick border.

<length>
: The border’s thickness has an explicit value. Explicit border widths cannot be negative.

The interpretation of the first three values depends on the user agent. The following relationships must hold, however:

’thin’ <=’medium’ <= ’thick’.

Furthermore, these widths must be constant throughout a document.

’border-top-width’

Property name:	’border-top-width’
Value:	<border-width> | inherit
Initial:	medium
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

This property sets the width of the top border of a box.

```
H1 { border: solid thick red }
P  { border: solid thick blue }
```

In the example above, H1 and P elements will have the same border width regardless of font size. To achieve relative widths, the 'em' unit can be used:

```
H1 { border: solid 0.5em }
```

'border-right-width'

Property name: 'border-right-width'
Value: <border-width> | inherit
Initial: medium
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

This property sets the width of the right border of a box.

'border-bottom-width'

Property name: 'border-bottom-width'
Value: <border-width> | inherit
Initial: medium
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

This property sets the width of the bottom border of a box.

'border-left-width'

Property name: 'border-left-width'
Value: <border-width> | inherit
Initial: medium
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

This property sets the width of the left border of a box.

'border-width'

Property name: 'border-width'
Value: <border-width>{1,4} | inherit
Initial: see individual properties
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

This property is a shorthand property for setting ’border-top-width’, ’border-right-width’, ’border-bottom-width’, and ’border-left-width’ at the same place in the style sheet.

If there is only one value, it applies to all sides. If there are two values, the top and bottom borders are set to the first value and the right and left are set to the second. If there are three values, the top is set to the first value, the left and right are set to the second, and the bottom is set to the third. If there are four values, they apply to the top, right, bottom, and left, respectively.

In the examples below, the comments indicate the resulting widths of the top, right, bottom and left borders:

```
H1 { border-width: thin }                  /* thin thin thin thin */
H1 { border-width: thin thick }            /* thin thick thin thick */
H1 { border-width: thin thick medium }     /* thin thick medium thick */
```

Border color: ’border-top-color’, ’border-right-color’, ’border-bottom-color’, ’border-left-color’, and ’border-color’

The border color properties specify the color of a box’s border.

’border-top-color’

Property name: ’border-top-color’
Value: <color> | inherit
Initial: the value of the ’color’ property
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

’border-right-color’

Property name: ’border-right-color’
Value: <color> | inherit
Initial: the value of the ’color’ property
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

’border-bottom-color’

Property name: ’border-bottom-color’
Value: <color> | inherit
Initial: the value of the ’color’ property
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

’border-left-color’

Property name:	’border-left-color’
Value:	<color> | inherit
Initial:	the value of the ’color’ property
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

’border-color’

Property name:	’border-color’
Value:	<color>{1,4} | inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

The ’border-color’ property sets the color of the four borders. ’border-color’ can have from one to four values, and the values are set on the different sides as for ’border-width’.

If an element’s border color is not specified with a border property, user agents must use the value of the element’s ’color’ property as the computed value [p. 57] for the border color.

In this example, the border will be a solid black line.

```
P {
  color: black;
  background: white;
  border: solid;
}
```

Border style: ’border-top-style’, ’border-right-style’, ’border-bottom-style’, ’border-left-style’, and ’border-style’

The border style properties specify the line style of a box’s border (solid, double, dashed, etc.). The properties defined in this section refer to the **<border-style>** value type, which make take one of the following:

none
: No border.

hidden
: Same as ’none’, except in tables [p. 207] .

dotted
: The border is a series of dots.

dashed
: The border is a series of short line segments.

solid
: The border is a single line segment.

double
: The border is two solid lines. The sum of the two lines and the space between them equals the value of ’border-width’.

groove
: The border looks as though it were carved into the canvas.

ridge
: The opposite of ’grove’: the border looks as though it were coming out of the canvas.

inset
: The border makes the entire box look as though it were embedded in the canvas.

outset
: The opposite of ’inset’: the order makes the entire box look as though it were coming out of the canvas.

All borders are drawn on top of the box’s background. The color of borders drawn for values of ’groove’, ’ridge’, ’inset’, and ’outset’ depends on the element’s ’color’ property.

Conforming HTML user agents [p. 27] may interpret ’dotted’, ’dashed’, ’double’, ’groove’, ’ridge’, ’inset’, and ’outset’ to be ’solid’.

’border-top-style’

Property name: ’border-top-style’
Value: <border-style> | inherit
Initial: none
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

’border-right-style’

Property name: ’border-right-style’
Value: <border-style> | inherit
Initial: none
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

’border-bottom-style’

Property name: ’border-bottom-style’
Value: <border-style> | inherit
Initial: none
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

’border-left-style’

Property name:	’border-left-style’
Value:	<border-style> | inherit
Initial:	none
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

’border-style’

Property name:	’border-style’
Value:	<border-style>{1,4} | inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

The ’border-style’ property sets the style of the four borders. It can have from one to four values, and the values are set on the different sides as for ’border-width’ above.

```
#xy34 { border-style: solid dotted }
```

In the above example, the horizontal borders will be ’solid’ and the vertical borders will be ’dotted’.

Since the initial value of the border styles is ’none’, no borders will be visible unless the border style is set.

Border shorthand properties: ’border-top’, ’border-bottom’, ’border-right’, ’border-left’, and ’border’

’border-top’

Property name:	’border-top’
Value:	[<’border-top-width’> || <’border-style’> || <color>] | inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

This is a shorthand property for setting the width, style, and color of the top border of a box.

```
H1 { border-bottom: thick solid red }
```

The above rule will set the width, style, and color of the border **below** the H1 element. Omitted values will be set to their initial values [p. 57] :

```
H1 { border-bottom: thick solid }
```

Since the color value is omitted in the above example, the border color will be the same as the 'color' value of the element itself.

Note that while the 'border-style' property accepts up to four values, this property only accepts one style value.

'border-bottom'

Property name:	'border-bottom'
Value:	[<'border-bottom-width'> || <'border-style'> || <color>] | inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

This is a shorthand property for setting the width, style, and color of the bottom border of a box. Otherwise, it behaves like 'border-top'.

'border-right'

Property name:	'border-right'
Value:	[<'border-right-width'> || <'border-style'> || <color>] | inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

This is a shorthand property for setting the width, style, and color of the right border of a box. Otherwise, it behaves like 'border-top'.

'border-left'

Property name:	'border-left'
Value:	[<'border-left-width'> || <'border-style'> || <color>] | inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

This is a shorthand property for setting the width, style, and color of the left border of a box. Otherwise, it behaves like 'border-top'.

’border’

Property name:	’border’
Value:	[<’border-width’> || <’border-style’> || <color>]	inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

The ’border’ property is a shorthand property for setting the same width, color, and style for all four borders of a box. Unlike the shorthand ’margin’ and ’padding’ properties, the ’border’ property cannot set different values on the four borders. To do so, one or more of the other border properties must be used. Note that while the ’border-width’ property accepts up to four length values, this property only accepts one.

For example, the first rule below is equivalent to the set of four rules shown after it:

```
P { border: solid red }
P {
  border-top: solid red;
  border-right: solid red;
  border-bottom: solid red;
  border-left: solid red
}
```

Since, to some extent, the properties have overlapping functionality, the order in which the rules are specified is important.

Consider this example:

```
BLOCKQUOTE {
  border-color: red;
  border-left: double;
  color: black
}
```

In the above example, the color of the left border is black, while the other borders are red. This is due to ’border-left’ setting the width, style, and color. Since the color value is not given by the ’border-left’ property, it will be taken from the ’color’ property. The fact that the ’color’ property is set after the ’border-left’ property is not relevant.

9.2 Containing blocks

In CSS2, all box positions are calculated with respect to the edges of a rectangular box called a *containing block*.

The root of the document tree [p. 26] generates a box that serves as the *initial containing block* for subsequent layout. If the ’width’ property for the root element has the value ’auto’, the user agent supplies an initial width (e.g., the user agent uses the current width of the viewport [p. 68]).

The initial containing block cannot be positioned (i.e., user agents skip [p. 31] the ’position’ property) or floated (i.e., user agents skip [p. 31] the ’float’ property).

The containing block for an box other than the root box determined as follows:

- If the value of the ’position’ property for the parent element is anything but ’normal’ then the containing block is established by the parent box.
- Otherwise, if the value of the ’display’ property for the parent element is anything but ’inline’ then the containing block is established by the parent box.
- Otherwise, the containing block is the parent’s containing block.

When the containing block is established by a block-level [p. 69] box, it has the same width, height, and position as the parent’s padding edge [p. 72] . When the containing block is established by an inline [p. 69] box, it has the same width, height, and position as the padding edge [p. 72] of the first line box [p. 79] generated by the inline box.

Relatively positioned [p. 82] inline [p. 69] boxes must be considered specially since (1) the rendered content [p. 25] of an inline element may not be rectangular in shape but (2) a relatively positioned box establishes a new containing block and containing blocks must be rectangular. The reference edges for descendants are the following:

- For left-to-right scripts, the left and top edges are those of the first line box. The bottom and right edges are undefined in this case.
- For right-to-left scripts, the right and top edges are those of the first line box. The bottom and left edges are undefined in this case.

9.3 Box width calculations

As discussed in the section on box dimensions [p. 71] , the content width [p. 72] of a box is either assigned explicitly (via the ’width’ property) or calculated "top-down" (based on box dimensions and the available horizontal space of its containing block). The following sections explain the exact computation of a box’s width-related properties.

9.3.1 Content width: the ’width’ property

’width’

Property name:	’width’
Value:	<length> | <percentage> | auto | inherit
Initial:	auto
Applies to:	all elements but non-replaced inline elements, table rows and row groups
Inherited:	no
Percentage values:	refer to width of containing block
Media groups:	visual [p. 65]

This property specifies a box’s content width [p. 72] . Values have the following meanings:

<length>
Specifies a fixed width.

<percentage>
Specifies a percentage width. The percentage is calculated with respect to the generated box’s containing block [p. 111] .

auto
See the section on widths of boxes in the normal flow and floated boxes [p. 113] .

Negative values for ’width’ are illegal. User agents must skip [p. 31] this property for elements with ’display’ set to ’row’ or ’row-span’

For example:

```
IMG.icon { width: 100px }
```

9.3.2 Widths of boxes in the normal flow and floated boxes

The ’width’ property does not apply to non-replaced inline [p. 69] elements. The width of the line boxes [p. 79] is determined by their containing block [p. 111] .

For other types of boxes in the normal flow [p. 79] and floats [p. 82] , a box’s margin, padding, border, and content width must equal the width of its containing block [p. 111] . These widths are specified with the following seven properties: ’margin-left’, ’border-left’, ’padding-left’, ’width’, ’padding-right’, ’border-right’, and ’margin-right’.

Three of these properties (’margin-left’, ’width’, and ’margin-right’) may take the value ’auto’. All properties with ’auto’ values must be assigned computed values [p. 57] by the user agent during layout. User agents must assign computed values as follows.

Determining the content width

When the ’width’ property has the value ’auto’, user agents should assign it a computed value [p. 57] based on computed margin widths [p. 114] . In addition:

1. For inline [p. 69] elements and those that generate floating [p. 82] boxes, the computed value is ’0’.
2. For replaced elements [p. 26] , user agents should use the intrinsic width [p. 73] of the element’s rendered content [p. 25] as the box’s computed [p. 57] width. (Note that for other values of ’width’, the replaced contents may be scaled).
3. Authors (and user agents) may specify minimum and maximum [p. 114] computed values for the ’width’ property via the ’min-width’ and ’max-width’ properties.

Computing margin widths

User agents should assign computed values [p. 57] for ’margin-left’, ’width’, and ’margin-right’ as follows:

1. If exactly one of ’margin-left’, ’width’ or ’margin-right’ has the value ’auto’, the computed value of that property is assigned so that the seven width properties add up to the width of the containing block [p. 111] .
2. If none of the properties has the value ’auto’, ’margin-right’ is assigned a computed value so that the seven width properties add up to the width of the containing block [p. 111] .
3. If more than one of the three has the value ’auto’, and one of them is ’width’, than the other two (’margin-left’ and/or ’margin-right’) will be assigned the computed value ’0’ and ’width’ will be assigned a computed value so that the seven width properties add up to the width of the containing block [p. 111] .
4. Otherwise, if both ’margin-left’ and ’margin-right’ have the value ’auto’, they will each be assigned a computed value that is half of the available horizontal space (this centers the box within its containing block).

9.3.3 Width of absolutely positioned elements

The width of an absolutely positioned [p. 85] box is specified via the source element’s ’width’ property.

However, if the ’width’ property has the value ’auto’, the width of the box is given by the ’left’ and ’right’ properties. Note that these take the place of the ’margin-left’ and ’margin-right’ properties, which don’t apply to absolutely positioned elements.

If all three properties have the value ’auto’, the box has exactly the width of its containing block [p. 111] .

9.3.4 Minimum and maximum widths: ’min-width’ and ’max-width’

’min-width’

Property name:	’min-width’
Value:	<length> | <percentage> | inherit
Initial:	0
Applies to:	all
Inherited:	no
Percentage values:	refer to parent’s width
Media groups:	visual [p. 65]

’max-width’

Property name:	’max-width’
Value:	<length> | <percentage> | inherit
Initial:	100%
Applies to:	all
Inherited:	no
Percentage values:	refer to parent’s width
Media groups:	visual [p. 65]

These two properties allow authors to constrain box widths to a certain range. Values have the following meanings:

<length>
Specifies a fixed minimum or maximum computed width.

<percentage>
Specifies a percentage for determining the computed value. The percentage is calculated with respect to the generated box’s containing block [p. 111] .

The following algorithm describes how the two properties influence the computed value [p. 57] of the ’width’ property:

1. the normal width [p. 113] is computed (without ’min-width’ and ’max-width’).
2. if the computed value of ’min-width’ is greater than the value of ’max-width’, ’max-width’ should be set to the value of ’min-width’.
3. if the computed width is greater than ’max-width’, the computed value of ’width’ is set to the value of ’max-width’.
4. if the computed width is smaller than ’min-width’, the computed value of ’width’ is set to the value of ’min-width’.

9.4 Box height calculations

As discussed in the section on box dimensions [p. 71] , the content height [p. 72] of a box is either assigned explicitly (via the ’height’ property) or is the minimal height necessary to include the vertical content of the element and that of all its flowed children. This is the height necessary *before* any relative offset of children.

9.4.1 Content height: the ’height’ property

’height’

Property name:	’height’
Value:	<length> | <percentage> | auto | inherit
Initial:	auto
Applies to:	all elements but non-replaced inline elements, table columns and column groups
Inherited:	no
Percentage values:	see prose
Media groups:	visual [p. 65]

This property specifies a box's content height [p. 72] . Values have the following meanings:

<length>
Specifies a fixed height.

<percentage>
Specifies a percentage height. The percentage is calculated with respect to the generated box's containing block [p. 111] .

auto
See the section on heights of boxes in the normal flow and floated boxes [p. 113] .

Negative values for 'height' are illegal. User agents must skip [p. 31] this property for elements with 'display' set to 'col' or 'column-span'

```
IMG.icon { height: 100px }
```

The height may be enforced by the user interface (e.g., a scrollbar).

9.4.2 Determining the content height

When the 'height' property has the value 'auto', user agents should assign it a computed value [p. 57] based on the space required by the element's rendered content [p. 25] . In addition:

1. For replaced elements [p. 26] , user agents should use the intrinsic height [p. 73] of the element's rendered content [p. 25] as the box's computed [p. 57] height. (Note that for other values of 'height', the replaced contents may be scaled).
2. Authors (and user agents) may specify minimum and maximum [p. 116] computed values for the 'height' property via the 'min-height' and 'max-height' properties.

9.4.3 Height of absolutely positioned elements

The height of an absolutely positioned box is specified via the source element's 'height' property. However, specifying a percentage value for 'height' when the parent element's height is set to 'auto' results in undefined behavior.

If the 'height' has the value 'auto', the height of the box is given by the 'top' and 'bottom' properties. Note that these take the place of the 'margin-top' and 'margin-bottom' properties, which don't apply to absolutely positioned elements.

If all three properties have the value 'auto', the box has exactly the height of its containing block [p. 111] .

9.4.4 Minimum and maximum heights: 'min-height' and 'max-height'

It is sometimes useful to constrain the height of elements to a certain range. Two properties offer this functionality:

’min-height’

Property name:	’min-height’
Value:	<length> | <percentage> | inherit
Initial:	0
Applies to:	all
Inherited:	no
Percentage values:	refer to parent’s height
Media groups:	visual [p. 65]

’max-height’

Property name:	’max-height’
Value:	<length> | <percentage> | inherit
Initial:	100%
Applies to:	all
Inherited:	no
Percentage values:	refer to parent’s height
Media groups:	visual [p. 65]

These two properties allow authors to constrain box heights to a certain range. Values have the following meanings:

<length>
Specifies a fixed minimum or maximum computed height.

<percentage>
Specifies a percentage for determining the computed value. The percentage is calculated with respect to the generated box’s containing block [p. 111] .

The following algorithm describes how the two properties influence the computed value [p. 57] of the ’height’ property:

1. the normal height [p. 116] is computed (without ’min-height’ and ’max-height’).
2. if the value of ’min-height’ is greater than the value of ’max-height’, ’max-height’ should be set to the value of ’min-height’
3. if the calculated height is greater than ’max-height’, the value of ’height’ is set to ’max-height’.
4. if the calculated height is smaller than ’min-height’, the value of ’height’ is set to ’min-height’.

9.5 Collapsing margins

In CSS2, horizontal margins never collapse.

Vertical margins may collapse for certain types of boxes:

- Two or more adjoining vertical margins (i.e., with no border, padding, or content between them) of block-level [p. 69] boxes in the normal flow collapse. The resulting margin width is the maximum of the adjoining margin widths. In the case of negative margins, the absolute maximum of the

negative adjoining margins is deducted from the maximum of the positive adjoining margins. If there are no positive margins, the absolute maximum of the negative adjoining margins is deducted from zero.

- Vertical margins between a floating [p. 82] box and another box do not collapse.
- Vertical margins between an absolutely positioned [p. 85] boxes do not collapse.

In most cases, after collapsing the vertical margins, the result is visually more pleasing and closer to what the designer expects. Please consult the examples of margin, padding, and borders [p. 73] for an illustration of collapsed margins.

9.6 Line height calculations: the ’line-height’ and ’vertical-align’ properties

As described in the section on inline formatting contexts [p. 79] , user agents flow inline boxes into a vertical stack of line boxes [p. 79] . The height of a line box is determined as follows:

1. The height of each inline box in the line box is calculated. The height of an inline box is given by the ’line-height’ property. The height is generally proportional to the font size of text in the box.

   In the case of an inline replaced element [p. 26] , the ’height’ property determines the height of the inline box.
2. The inline boxes are aligned vertically according to the ’vertical-align’ property value for source inline elements. If an element has the values ’top’ or ’bottom’ for this property, only the height of the generated boxes affects the line box height calculation; the boxes cannot be aligned until the line box has been fully constructed.
3. The line box height is distance between the uppermost box top and the lowermost box bottom.

Note that if all the boxes in the line box are aligned along their bottoms, the line box will be exactly the height of the tallest box. If, however, the boxes are aligned along a common baseline, the line box top and bottom may not touch the top and bottom of the tallest box.

Note that top and bottom margins, borders, and padding specified for inline [p. 69] elements do *not* enter into the calculation of line box heights.

9.6.1 Leading and half-leading

Since the height of an inline box may be greater than (’line-height’ > 1) the font size of text in the box, there may be space above and below rendered glyphs. The difference between the font size and the computed value of ’line-height’ is called the *leading*. Half the leading is called the *half-leading*.

User agents should center the glyph vertically in the box (adding half-leading on the top and bottom). For example, if a piece of text is ’12pt’ high and the ’line-height’ value is ’14pt’, 2pts of extra space should be added (e.g., 1pt above and 1pt below the box). Empty elements influence these calculations just like elements with content.

When the ’line-height’ value is less than the font size, the final line box height will be less than the font size and the rendered glyphs will "bleed" outside the box. If such a box touches the edge of a line box, the rendered glyphs will also "bleed" into the adjacent line box.

Although margins, borders, and padding do not enter into the line box height calculation, they are still rendered around inline boxes (except when boxes are split across lines). This means that if the height of a line box is shorter than the outer edges [p. 72] of the boxes it contains, backgrounds and colors may "bleed" into adjacent line boxes.

’line-height’

Property name:	’line-height’
Value:	normal | <number> | <length> | <percentage> | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentage values:	relative to the font size of the element itself
Media groups:	visual [p. 65]

The property specifies the height of an inline box. The height generally refers to the font size of the element’s text. For replaced elements [p. 26] , the height refers to the intrinsic height.

- If the property is set on a block-level [p. 69] element whose content is composed of inline [p. 69] elements, it specifies the *minimal* height of each inline box.
- If the property is set on an inline [p. 69] element, it specifies the *exact* height of each box generated by the element.

Values for this property have the following meanings:

normal
: Tells user agents to set the computed value [p. 57] to a "reasonable" value based on the font size of the element. The value has the same meaning as ’number’. We recommend a value in the range of 1.0 to 1.2.

<length>
: The box height is set to this length. Negative values are illegal.

<number>
: This number is actual value [p. 58] for the property (and thus, it is inherited). The computed value [p. 57] of the property is this number multiplied by the element’s font size. Negative values are illegal.

<percentage>
: The actual value [p. 58] of the property is this percentage multiplied by the element’s font size. Negative values are legal. Percentages raise the baseline of the element (or the bottom, if it has no baseline) the specified amount above the baseline of the parent. For example, a value of ’-100%’ will lower the element so that the baseline of the element ends up where the baseline of the next line should have been. This allows precise control over the vertical position of elements (such as images that are used in place of letters) that don’t have a baseline.

The three rules in the example below have the same resultant line height:

```
DIV { line-height: 1.2; font-size: 10pt }     /* number */
DIV { line-height: 1.2em; font-size: 10pt }   /* length */
DIV { line-height: 120%; font-size: 10pt }    /* percentage */
```

When an element contains text that is rendered in more than one font, user agents should determine the 'line-height' value according to the largest font size.

Note. *Generally, when there is only one value of 'line-height' affecting all line boxes in a paragraph (and not tall images), the above will ensure that baselines of successive lines are exactly 'line-height' apart. This is important when columns of text in different fonts have to be aligned, for example in a table.*

'vertical-align'

Property name:	'vertical-align'
Value:	baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <length> | inherit
Initial:	baseline
Applies to:	inline elements
Inherited:	no
Percentage values:	refer to the 'line-height' of the element itself
Media groups:	visual [p. 65]

This property affects the vertical positioning inside a line box of the boxes generated by an inline element. The following values only have meaning with respect to a parent inline element; they have no effect if no parent exists:

'baseline'
Align the baseline of text in the box with the baseline of text in the parent box. If the box doesn't contain text, align the bottom of the box with the parent's baseline.

'middle'
Align the vertical midpoint of the box with the baseline of text in the parent box plus half the x-height of the parent.

'sub'
Subscript the box. This value has no effect on the font size of the element's text.

'super'
Superscript the element. This value has no effect on the font size of the element's text.

'text-top'
Align the top of the box with the top of the parent element's font

'text-bottom'
Align the bottom of the box with the bottom of the parent element's font

The remaining values refer to the line box in which the generated box appears:

'top'
Align the top of the box with the top of the line box.

’bottom’

Align the bottom of the box with the bottom of the line box.

9.7 Floating constraints

A floating box is positioned subject to the following constraints:

1. The left outer edge [p. 72] of a left-floating box may not be to the left of the left inner edge [p. 72] of its containing block [p. 111] . Analogous rules hold for right-floating elements.
2. The left outer edge [p. 72] of a left-floating box must be to the right of the right outer edge [p. 72] of every preceding [p. 67] left-floating box or the top of the former must be lower than the bottom of the latter. Analogous rules hold for right-floating boxes.
3. The right outer edge [p. 72] of a left-floating box may not be to the right of the left outer edge [p. 72] of any right-floating box that is to the right of it. Analogous rules hold for right-floating elements.
4. A floating box’s top may not be higher than the inner [p. 72] top of its containing block [p. 111] .
5. The top of an element’s floating box may not be higher than the top of any block-level [p. 69] or floated [p. 82] box generated by a preceding [p. 26] element.
6. The top of an element’s floating box may not be higher than the top of any line-box [p. 79] containing a box generated by a preceding [p. 26] element.
7. A floating box must be placed as high as possible.
8. A left-floating box must be put as far to the left as possible, a right-floating box as far to the right as possible. A higher position is preferred over one that is further to the left/right.

10 Visual effects

Contents

1. Overflow and clipping [p. 123]
    1. Overflow [p. 123] : the ’overflow’ property
    2. Clipping [p. 125] : the ’clip’ property
2. Visibility [p. 127] : the ’visibility’ property

The visual effects discussed in these sections do not alter layout, only presentation.

10.1 Overflow and clipping

10.1.1 Overflow: the ’overflow’ property

Generally, a box is confined by its containing block [p. 67] . In certain cases, a box may *overflow*, meaning it lies partly or entirely outside of its containing block:

- It is floated [p. 82] and is too wide for the containing block.
- Its height exceeds an explicit height assigned to the containing block (through the ’height’ property).
- It is positioned absolutely [p. 85] .
- It has negative margins [p. 99] .

The ’overflow’ property specifies rendering in these cases.
’overflow’

Property name:	’overflow’
Value:	visible | hidden | scroll | auto | inherit
Initial:	visible
Applies to:	elements with the ’position’ property set to ’absolute’
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

This property specifies how the containing block [p. 67] of boxes generated for an element should behave when the boxes overflow. Note that in general, the width of the containing block is fixed (i.e., width calculations are top-down). In the case of overflow, width calculations may have a bottom-up effect on the containing block.

Values for this property refer to the behavior of the containing block for boxes generated for this element. They have the following meanings:

visible

This value indicates that the containing block [p. 67] should be enlarged enough to contain overflowing boxes. Any padding or border will remain outside the boxes. Any additional width will be added in the direction specified by the current value of the ’direction’ property. Additional height will be added to the bottom.

hidden

This value indicates that the dimensions of the containing block [p. 67] should not be adjusted to contain overflowing boxes and that no scrolling mechanism should be provided to view the partially or entirely clipped boxes; users will not have access to clipped content. Padding and border will be applied to the regular height and width of each overflowing box, as if it were not overflowing.

scroll

This value indicates that if the user agent supports a visible scrolling mechanism, that mechanism should be displayed for a containing block whether or not its boxes overflow. This avoids any problem with scrollbars appearing and disappearing in a dynamic environment.

auto

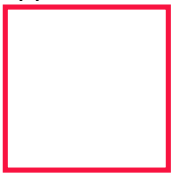
The behavior of the 'auto' value is user agent-dependent, but should cause a scrolling mechanism to appear on a containing block when one or more of its boxes overflows.

Even if 'overflow' is set to 'visible', contents may be clipped to a UA's document window by the native operating environment. In addition, the 'clip' property can cause otherwise visible "overflowed" contents to be clipped.

The examples below utilize the following style sheet, which describes a simple 100 pixel box with a thin, solid-red border:

```
#overlay {position: absolute; top: 50px; left: 50px; height: 100px;
          width: 100px; border: thin solid red;}
```

Applied to an empty <DIV>, the box would look something like:

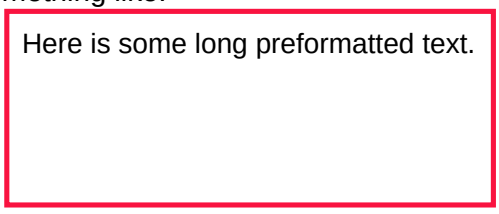


First, let's consider the initial value of 'overflow', which is 'visible'. This value indicates that all boxes should be visible, even if they overflow.

In HTML, the PRE element is a block-level [p. 69] element that acts as a containing block for inline [p. 69] contents. Consider what happens when the text content of PRE is longer than the width specified for PRE (here, 100px):

```
<PRE id="overlay">Here is some long preformatted text.</PRE>
```

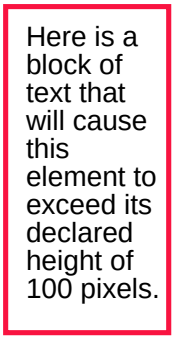
With 'overflow' set to 'visible', the containing block will increase to accommodate the generated inline boxes, and all of the text will be visible (and padding and border rendered as well). The example might be rendered something like:



Similarly, a value of ’visible’ will cause a containing block to grow if the height of rendered contents exceed the containing block’s declared height. Consider the following:

```
<BODY>
<DIV id="overlay">Here is a block of text that will
cause this element to exceed its declared height of 100 pixels.
</DIV>
</BODY>
```

This DIV element should be rendered something like this:



The value ’hidden’, on the other hand, specifies that a containing block should not grow to accommodate overflow. If we had set the ’overflow’ to ’hidden’ in the two previous examples, the containing block established by the PRE element would have maintained its specified size of 100px by 100px. The examples might have been rendered something like:

10.1.2 Clipping: the ’clip’ property

A *clipping region* defines what portion of an element’s rendered content [p. 25] is visible. The clipping region for an element is that of its parent, modified by the value of the element’s ’clip’ property.

’clip’

Property name:	’clip’
Value:	<shape> | auto | inherit
Initial:	auto
Applies to:	elements with the ’position’ property set to ’absolute’
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

Values have the following meanings:

auto

The clipping region for the element has the dimensions of the containing block of boxes generated for the element.

<shape>

In CSS2, the only legal <shape> value is: rect (<top> <right> <bottom> <left>)

where <top>, <bottom> <right>, and <left> specify offsets from the respective sides of the parent's (rectangular) clipping region.

<top>, <right>, <bottom>, and <left> may either have a <length> value or 'auto'. Negative lengths are permitted. The value 'auto' means that a given edge of the clipping region will be the same as the edge of the element's generated box.

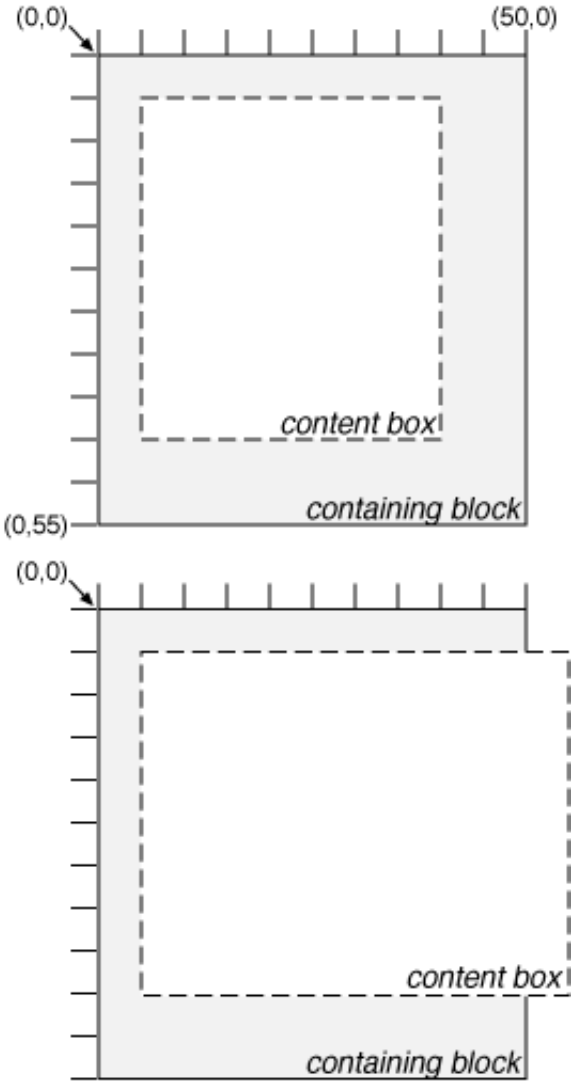
When converted to pixel coordinates, the bottom-right corner is excluded from the clipping rectangle. This rule is necessary to permit the definition of zero-width or zero-height rectangles.

If the clipping region exceeds the bounds of the UA's document window, contents may be clipped to that window by the native operating environment.

The following two rules:

```
P { clip: rect(5px, 10px, 10px, 5px); }
P { clip: rect(5px, -5px, 10px, 5px); }
```

will create the rectangular clipping regions delimited by the dotted line in the following illustrations:



Note. *For now, all clipping regions are rectangular. We anticipate future extensions to permit non-rectangular clipping.*

10.2 Visibility: the ’visibility’ property

’visibility’

Property name:	’visibility’
Value:	inherit | visible | hidden
Initial:	inherit
Applies to:	all elements
Inherited:	if value is ’inherit’
Percentage values:	N/A
Media groups:	visual [p. 65]

The ’visibility’ property specifies whether the boxes generated by an element are rendered. Invisible boxes still affect layout (set the ’display’ property to ’none’ to suppress box generation altogether). This property may be used in conjunction with scripts to create dynamic effects.

In the following example, pressing either form button invokes a user-defined script function that causes the corresponding box to become visible and the other to be hidden. Since these boxes have the same size and position, the effect is that one replaces the other.

```
<HTML>
<HEAD>
<STYLE type="text/css">
<!--
   #container1 { position: absolute; top: 2in; left: 2in; width: 2in}
   #container2 { position: absolute; top: 2in; left: 2in; width: 2in;
                 visibility: hidden; }
-->
</STYLE>
</HEAD>
<BODY>
<P>Choose a suspect:</P>
<DIV id="container1">
   <IMG alt="Al Capone" width="100" height="100" src="suspect1.jpg">
   <P>Name: Al Capone</P>
   <P>Residence: Chicago</P>
</DIV>

<DIV id="container2">
   <IMG alt="Lucky Luciano" width="100" height="100" src="suspect2.jpg">
   <P>Name: Lucky Luciano</P>
   <P>Residence: New York</P>
</DIV>

<FORM method="post" action="http://www.suspect.org/process-bums">
   <P>
   <INPUT name="Capone" type="button" value="Capone" onclick='show("container1");hide("container2")'>
   <INPUT name="Luciano" type="button" value="Luciano" onclick='show("container2");hide("container1")'>
</FORM>
</BODY>
</HTML>
```

Note that the 'position' property of each DIV element has the value 'relative'. A more visually appealing version of the above might be designed using overlapping absolutely positioned [p. 85] boxes.

11 Generated content and automatic numbering

Contents

1. The [p. 129] :before and :after pseudo elements and the ’content’ property
2. Automatic counters and numbering [p. 130]

In some cases, authors may want user agents to display content that does not come from the document tree [p. 26] . One familiar example of this is a numbered list; the author does not want to list the numbers explicitly, he or she wants the user agent to generate them automatically. Similarly, authors may want the user agent to insert the word "Figure" before the caption of a figure, or "Chapter 7" before the seventh chapter title. For audio or braille in particular, user agents should be able to insert these strings.

In CSS2, inserted content can consist of combinations of the following:

- fixed strings (e.g., "Figure"),
- fixed images (such as an icon: [ICON HERE]),
- counters that are automatically incremented.

Below we describe the mechanisms CSS2 offers for specifying generated content.

11.1 The :before and :after pseudo elements and the ’content’ property

Authors specify generated content with the :before and :after pseudo-elements. The ’content’ property of these pseudo-elements specifies what is inserted. :before and :after inherit [p. 58] any inheritable properties from the element to which they are attached.

For example, the following rule inserts a double quote mark before and after every Q element. The color of the quote will be red, but the font will be the same as the font of the rest of the Q element:

```
Q:before, Q:after {
    content: "\"";
    color: red
}
```

In a :before or :after pseudo-element declaration, non-inherited properties take their initial value [p. 57] .

Because the initial value of the ’display’ property is ’inline’, the quote in the previous example is inserted as an inline box, and not above (or below) the content of the Q. The next example explicitly sets the ’display’ property to ’block’, so that the inserted text becomes a block:

```
BODY:after {
    content: "The End";
    display: block;
    margin-top: 2em;
    text-align: center;
}
```

’content’

Property name: ’content’
Value: [<string> | <uri> | <counter>]+ | inherit
Initial: empty string
Applies to: :before and :after
Inherited: no
Percentage values: N/A
Media groups: all [p. 65]

The value of the ’content’ property is a sequence of strings, URIs, and counters. The various parts are laid out as inline boxes inside the element.

The values have the following meanings:

<string>
??
<counter>
??

Note. *In future levels of CSS, the ’content’ property may accept additional values, allowing it to vary the style of the generated content, but in CSS2 the content of the :before or :after pseudo-element is all in one style.*

[How to do language-dependent quotes? [LANG=fr] Q:before{content: "«"} doesn’t work. How about Q:fr-fr:before {content: "«"} ? or @lang fr-fr {Q:before{content: "«"}} ?]

Newlines can be included in the generated content by writing the "\A" (or "\00000A") escape sequence in one of the strings after the ’content’ property. This inserts a forced line break, similar to the BR element in HTML. See "Strings" [p. 41] and "Characters and case" [p. 32] for more information on the "\A" escape sequence.

Example:

```
H1:before {
    display: block;
    content: "chapter\00000Achapter\00000Achapter"
}
```

11.2 Automatic counters and numbering

This is a placeholder.

12 Paged media

Contents

1. Introduction to paged media [p. 131]
2. Page boxes [p. 132] : the @page rule
    1. Page margins [p. 132]
    2. Page size [p. 133] : the ’size’ property
        1. Rendering page boxes that do not fit a target sheet [p. 134]
        2. Positioning the page box on the sheet [p. 135]
    3. Crop marks [p. 135] : the ’marks’ property
    4. Left, right, and first pages [p. 135]
    5. Content outside the page box [p. 136]
3. Page breaks [p. 137]
    1. Page break properties [p. 137] : ’page-break-before’, ’page-break-after’, ’orphans’, and ’widows’
    2. Allowed page breaks [p. 138]
    3. Forced page breaks [p. 139]
    4. "Best" page breaks [p. 139]
4. Cascading in the page context [p. 140]

12.1 Introduction to paged media

Paged media -- paper, transparencies, pages that are displayed on computer screens, etc. -- differ from continuous media [p. 65] in that formatting algorithms for pages must manage page breaks. To handle page breaks, CSS2 extends the visual rendering model [p. 67] as follows:

1. The page box [p. 132] extends the box model [p. 68] to allow authors to specify the size of a page, its margins, etc.
2. The *page model* extends the visual rendering layout model to account for page breaks. [p. 137]

The CSS2 page model specifies how a document is formatted within a rectangular area -- the page box [p. 132] -- that has a finite width and height. The page box does not necessarily correspond to the real *sheet* where the document will ultimately be rendered (paper, transparency, screen, etc.). The CSS page model specifies formatting in the page box, but it is the user agent’s responsibility to transfer the page box to the sheet. Some transfer possibilities include:

- Transferring one page box to one sheet (e.g., single-sided printing).
- Transferring two page boxes to both sides of the same sheet (e.g., double-sided printing).
- Transferring N (small) page boxes to one sheet (called "n-up").
- Transferring one (large) page box to N x M sheets (called "tiling").
- Creating signatures. A signature is a group of pages printed on a sheet, which, when folded and trimmed like a book, appear in their proper sequence.
- Printing one document to several output trays.

- Outputting to a file.

Although CSS2 does not specify how user agents transfer page boxes to sheets, it does include certain mechanisms for telling user agents about the target sheet size and orientation [p. 133] .

12.2 Page boxes: the @page rule

The *page box* is a rectangular region that contains two embedded areas:

- The *page area*. The page area includes the boxes laid out on that page. The edges of the page area act as a containing block [p. 111] for layout that occurs between page breaks. It also acts as the initial containing block [p. 111] .
- The margin area (see the margin area [p. 71] of the box model).

Note. *In CSS2, the border properties [p. 104] and padding properties [p. 102] do not apply to pages; they may in the future.*

Authors specify the dimensions, orientation, margins, etc. of a page box within an @page rule.

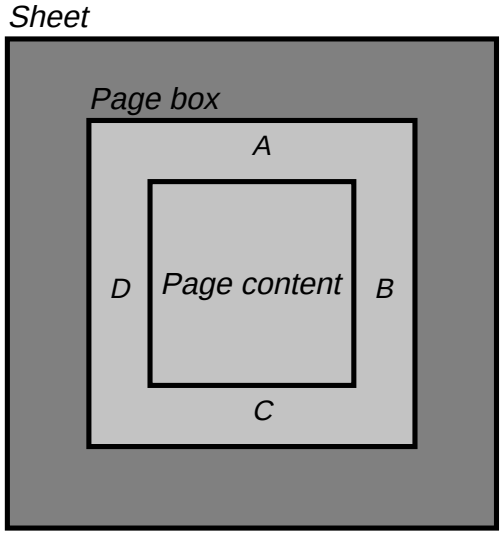
For example, the following @page rule sets the margins of the page to 2cm.

```
@page { margin: 2cm }
```

Declarations inside the curly braces of the @page rule apply to every page of a document; these declarations are said to be in the *page context*. Authors specify the dimensions of the page box with the 'size' property. The 'marks' property is used to specify crop and cross marks.

12.2.1 Page margins

The margin properties [p. 99] ('margin-top', 'margin-right', 'margin-bottom', 'margin-left', and 'margin') apply within the page context [p. 132] . The following diagram shows the relationships between the sheet, page box, and page margins:



A: margin−top
B: margin−right
C: margin−bottom
D: margin−left

The CSS2 rules for collapsing vertical margins [p. 117] apply to page margins as well. For example, the margin of the first box on a page will collapse with the page margin.

The page context [p. 132] has no notion of fonts, so ’em’ and ’ex’ units are not allowed. Percentage values on the margin properties are relative to the dimensions of the page box [p. 132] . All other units associated with the respective CSS2 properties are allowed.

Due to negative margin values (either on the page box or on elements) or absolute positioning [p. 85] content may end up outside the page box, but this content may be "cut" -- by the user agent, the printer, or ultimately, the paper cutter.

12.2.2 Page size: the ’size’ property

’size’

Property name:	’size’
Value:	<length>{1,2} | auto | portrait | landscape | inherit
Initial:	auto
Applies to:	page context
Inherited:	N/A
Percentage values:	N/A
Media groups:	visual [p. 65] , paged [p. 65]

This property specifies the size and orientation of a page box.

The size of a page box may either be "absolute" (fixed size) or "relative" (scalable, i.e., fitting available sheet sizes). Relative page boxes allow user agents to scale a document and make optimal use of the target size. Absolute page boxes ensure precise formatting when that is the author’s prerogative.

Three values for the ’size’ property create a relative page box:

auto
: The page box will be set to the size and orientation of the target sheet.

landscape
: Overrides the target’s orientation. The page box is the same size as the target, and the normal direction of layout runs parallel to the longer target side.

portrait
: Overrides the target’s orientation. The page box is the same size as the target, and the normal direction of layout runs parallel to the shorter target side.

In the following example, the outer edges of the page box will align with the target. The percentage value on the ’margin’ property is relative to the target size so if the target sheet dimensions are 21.0cm x 29.7cm (i.e., A4), the margins are 2.10cm and 2.97cm.

```
@page {
  size: auto;   /* auto is the initial value */
  margin: 10%;
}
```

Length values for the ’size’ property create an absolute page box. If only one length value is specified, it sets both the width and height of the page box (i.e., the box is a square). Since the page box is the initial containing block [p. 111] , percentage values are not allowed for the ’size’ property.

For example:

```
@page {
  size: 8.5in 11in;  /* width height */
}
```

The above example set the width of the page box to be 8.5in and the height to be 11in. The page box in this example requires a target sheet size of 8.5"x11" or larger.

User agents may allow users to control the transfer of the page box to the sheet (e.g., rotating an absolute page box that’s being printed).

Rendering page boxes that do not fit a target sheet

If page box does not fit the target sheet dimensions, the user agent may choose to:

- Rotate the page box 90° if this will make the page box fit.
- Scale the page to fit the target.

The user agent should consult the user before performing these operations.

Positioning the page box on the sheet

When the page box is smaller than the target size, the user agent is free to place the page box anywhere on the sheet. However, it is recommended that the page box be centered on the sheet since this will align double-sided pages and avoid accidental loss of information that is printed near the edge of the sheet.

12.2.3 Crop marks: the ’marks’ property

’marks’

Property name:	’marks’
Value:	crop || cross | none | inherit
Initial:	none
Applies to:	page context
Inherited:	N/A
Percentage values:	N/A
Media groups:	visual [p. 65] , paged [p. 65]

In high-quality printing, marks are often added outside the page box. This property describes specifies whether cross marks or crop marks or both should be rendered just outside the page box [p. 132] edge. Values have the following meanings:

cross
: *Crop marks* indicate where the page should be cut.

crop
: *cross marks* (also known as register marks or registration marks) are used to align sheets.

none
: Don’t render any special marks.

Marks are only visible on absolute page boxes. In relative page boxes, the page box will be aligned with the target and the marks will be outside the printable area.

The size, style, and position of cross marks depends on the user agent.

12.2.4 Left, right, and first pages

When printing double-sided documents, the page boxes [p. 132] on left and right pages should be different. This can be expressed through two CSS pseudo-classes that may be defined in the page context [p. 132] .

All pages are automatically classified by user agents into either the :left or :right pseudo-class.

```
@page :left {
  margin-left: 4cm;
  margin-right: 3cm;
}

@page :right {
  margin-left: 3cm;
  margin-right: 4cm;
}
```

If different declarations have been given for left and right pages, the user agent must honor these declarations even if the user agent does not transfer the page boxes to left and right sheets (e.g., a printer that only prints single-sided).

Authors may also specify style for the first page of a document with the :first pseudo-class:

```
@page { margin: 2cm } /* All margins set to 2cm */

@page :first {
  margin-top: 10cm    /* Top margin on first page 10cm */
}
```

Whether the first page of a document is :left or :right depends on the major writing direction of the document and is outside the scope of this document. However, to force a :left or :right first page, authors may insert a page break before the first generated box (e.g., in HTML, specify this for the BODY element).

Properties specified in a :left (or :right) @page rule override those specified in an @page rule that has no pseudo-class specified. Properties specified in a :first @page rule override those specified in :left (or :right) @page rules.

Note. *Adding declarations to the :left or :right pseudo-class does not influence whether the document comes out of the printer double- or single-sided (which is outside the scope of this specification).*

Note. *Future versions of CSS may include other page pseudo-classes.*

12.2.5 Content outside the page box

When formatting content in the page model, some content may end up outside the page box. For example, an element whose ’white-space’ property has the value ’pre’ may generate a box that is wider than the page box. Also, when boxes are positioned absolutely [p. 85] or floated [p. 82] , they may end up in "inconvenient" locations. For example, images may be placed on the edge of the page box or 100,000 inches below the page box.

A specification for the exact formatting of such elements lies outside the scope of this document. However, we recommend that authors and user agents observe the following general principles concerning content outside the page box:

- Content should be allowed slightly beyond the page box to allow pages to "bleed".
- User agents should avoid generating a large number of empty page boxes to honor the positioning of elements (e.g., you don’t want to print 100 blank pages). Note, however, that generating a small number of empty page boxes may be necessary to honor the ’left’ and ’right’ values for ’page-break-before’ and ’page-break-after’.

- Authors should not position elements in inconvenient locations as a means to avoid laying them out. Instead:
  - To suppress box generation entirely, set the 'display' property to 'none'.
  - To render a box invisible, use the 'visibility' property.
- User agents may handle boxes positioned outside the page box in several ways, including discarding them or creating page boxes for them at the end of the document.

12.3 Page breaks

The following sections explain page formatting in CSS2. Four properties indicate where the user agent may or should break pages, and on what page (left or right) the subsequent content should resume. Each page break ends layout in the current page box [p. 132] and causes remaining pieces of the document tree [p. 26] to be laid out in a new page box.

12.3.1 Page break properties: 'page-break-before', 'page-break-after', 'orphans', and 'widows'

'page-break-before'

Property name:	'page-break-before'
Value:	auto | always | avoid | left | right | inherit
Initial:	auto
Applies to:	block-level and inline elements except those in tables
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65] , paged [p. 65]

'page-break-after'

Property name:	'page-break-after'
Value:	auto | always | avoid | left | right | inherit
Initial:	auto
Applies to:	block-level and inline elements except those in tables
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65] , paged [p. 65]

Values for these properties have the following meanings:

auto
Neither force nor forbid a page break before (resp., after) the generated box.
always
Always force a page break before (resp., after) the generated box.
avoid
Avoid a page break before (resp., after) the generated box.
left
(For rendering to left and right sheets.) Force one or two page breaks before

(resp., after) the generated box so that the next page is formated as a left page.

right

(For rendering to left and right sheets.) Force one or two page breaks before (resp., after) the generated box so that the next page is formatted as a right page.

When both properties have values other than ’auto’, the values ’always’, ’left’, and ’right’ take precedence over ’avoid’. See the section on allowed page breaks [p. 138] for the exact rules on how these values force or suppress a page break.

’orphans’

Property name:	’orphans’
Value:	<integer> | inherit
Initial:	2
Applies to:	block-level elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65] , paged [p. 65]

’widows’

Property name:	’widows’
Value:	<integer> | inherit
Initial:	2
Applies to:	block-level elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65] , paged [p. 65]

These properties specify the minimum number of lines of a paragraph that must be left at the bottom (’orphans’) and top (’widows’) of a page (see the section on line boxes [p. 79] for information about paragraph formatting).

12.3.2 Allowed page breaks

In the normal flow, page breaks can occur at the following places:

1. In the vertical margin between block-level boxes. When a page break occurs here, the computed values [p. 57] of the relevant ’margin-top’ and ’margin-bottom’ properties are set to ’0’.
2. Between line boxes [p. 79] inside a block-level [p. 69] box.

These breaks are subject to the following rules:

1. Breaking at (1) is only allowed if the ’page-break-after’ and ’page-break-before’ properties of all the elements generating boxes that meet at this margin allow it, which is when at least one of them has the value ’always’, ’left’, or ’right’, or when all of them are ’auto’.
2. Breaking at (2) is only allowed if the number of line boxes [p. 79] between the break and the start of the block is ’orphans’ or more, and the number of

line boxes between the break and the end of the block is ’widows’ or more.

There is an exception to both rules:

3. Breaking at (1) and (2) is also allowed if, between the last page break and the next one that would be allowed under (A) and (B), there is so much content that it can’t fit within a single page box [p. 132] .

Page breaks cannot occur inside boxes that are absolutely-positioned [p. 85] .

12.3.3 Forced page breaks

A page break *must* occur at (1) if, among the ’page-break-after’ and ’page-break-before’ properties of all the elements generating boxes that meet at this margin, there is at least one with the value ’always’, ’left’, or ’right’.

12.3.4 "Best" page breaks

CSS does *not* define which of a set of allowed page breaks must be used; CSS does not forbid a user agent from breaking at every possible break point, or not to break at all. But CSS does recommend that user agents observe the following heuristics (while recognizing that they are sometimes contradictory):

- Break as few times as possible.
- Make all pages that don’t end with a forced break appear to have about the same height.
- Avoid breaking inside a block that has a border.
- Avoid breaking inside a table.
- Avoid breaking inside a floating element

Suppose, for example, that ’orphans’=4, ’widows’=2, and there are 20 lines (line boxes [p. 79] available at the bottom of the current page:

- If a paragraph at the end of the current page contains 20 lines or fewer, it should be placed on the current page.
- If the paragraph contains 21 - 22 lines, the second part of the paragraph must not violate the ’widows’ constraint, and so the second part must contain exactly two lines
- If the paragraph contains 23 lines or more, the first part should contain 20 lines and the second part the remaining lines.

Now suppose that ’orphans’=10, ’widows’=20, and there are 8 lines available at the bottom of the current page:

- If a paragraph at the end of the current page contains 8 lines or fewer, it should be placed on the current page.
- If the paragraph contains 9 lines or more, it cannot be split (that would violate the orphan constraint), so it should move as a block to the next page.

12.4 Cascading in the page context

Declarations in the page context [p. 132] obey the cascade [p. 57] just like normal CSS2 declarations.

Consider the following example:

```
@page {
  margin-left: 3cm;
}

@page :left {
  margin-left: 4cm;
}
```

Due to the higher specificity [p. 60] of the pseudo-class selector, the left margin on left pages will be ’4cm’ and all other pages (i.e., the right pages) will have a left margin of ’3cm’.

13 Colors and Backgrounds

Contents

1. Foreground color [p. 141] : the ’color’ property
2. The background [p. 141]
    1. Background properties [p. 142] : ’background-color’, ’background-image’, ’background-repeat’, ’background-attachment’, ’background-position’, and ’background’

CSS properties allow authors to specify the foreground color and background of an element. Backgrounds may be colors or images. Background properties allow authors to position the image, repeat it, and declare whether it should be fixed with respect to the viewport [p. 68] or scrolled along with the document.

See the section on color units [p. 39] for the syntax of legal color values.

13.1 Foreground color: the ’color’ property

’color’

Property name:	’color’
Value:	<color> | inherit
Initial:	depends on user agent
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

This property describes the foreground color of an element’s text content. There are different ways to specify red:

```
EM { color: red }              /* predefined color name */
EM { color: rgb(255,0,0) }     /* RGB range 0-255   */
```

13.2 The background

Authors may specify the background of an element (i.e., its rendering surface) as either a color or an image. In terms of the box model [p. 68] , "background" refers to the background of the content [p. 71] and the padding [p. 71] area. Border colors and styles are set with the border properties [p. 104] . Margins are always transparent so the background of the parent box always shines through.

Background properties do not inherit, but the parent box’s background will shine through by default because of the initial ’transparent’ value on ’background-color’.

The background of the box generated by the root element covers the entire canvas [p. 22] .

For HTML documents, however, we recommend that authors specify the background for the BODY element rather than the HTML element. User agents should observe the following precedence rules to fill in the background: if the value of the ’background’ property for the HTML element is different from

’transparent’ then use it, else use the value of the ’background’ property for the BODY element. If the resulting value is ’transparent’, the rendering is undefined.

According to these rules, the canvas underlying the following HTML document will have a "marble" background and the background of the box generated by the BODY element will be red:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<STYLE type="text/css">
   { background: url(http://style.com/marble.png) }
</STYLE>
<TITLE>Setting the canvas background</TITLE>
</HEAD>
<BODY style="background: red">
<P>My background is red.
</BODY>
</HTML>
```

Note. *The document language may not define an element that gives direct access to the canvas.*

13.2.1 Background properties: ’background-color’, ’background-image’, ’background-repeat’, ’background-attachment’, ’background-position’, and ’background’

’background-color’

Property name:	’background-color’
Value:	<color> | transparent | inherit
Initial:	transparent
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

This property sets the background color of an element. Values have the following meanings:

<color>
Specifies a color name or value.

transparent
The background is transparent, so all underlying colors shine through.

```
H1 { background-color: #F00 }
```

’background-image’

Property name: 'background-image'
Value: <uri> | none | inherit
Initial: none
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

This property sets the background image of an element to be an image. When setting a background image, authors should also specify a background color that will be used when the image is unavailable. When the image is available, it is rendered on top of the background color.

Values have the following meanings:

<uri>
Specifications the location of the image.

none
No background image is used.

```
BODY { background-image: url(marble.gif) }
P { background-image: none }
```

'background-repeat'

Property name: 'background-repeat'
Value: repeat | repeat-x | repeat-y | no-repeat | inherit
Initial: repeat
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: visual [p. 65]

If a background image is specified, this property specifies whether the image is repeated, and how. Values have the following meanings:

repeat
The image is repeated both horizontally and vertically.

repeat-x
The image is repeated horizontally only.

repeat-y
The image is repeated vertically only.

no-repeat
The image is not repeated.

```
BODY {
  background: red url(pendant.gif);
  background-repeat: repeat-y;
}
```

In the example above, the image will only be repeated vertically.

’background-attachment’

Property name:	’background-attachment’
Value:	scroll | fixed | inherit
Initial:	scroll
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

If a background image is specified, this property specifies whether it is fixed with regard to the viewport [p. 68] or scrolls along with the document. Values have the following meanings:

fixed
The image is fixed with respect to the viewport.

scroll
The image is scrolled along with the document.

```
BODY {
  background: red url(pendant.gif);
  background-repeat: repeat-y;
  background-attachment: fixed;
}
```

User agents may treat ’fixed’ as ’scroll’. However, it is recommended they interpret ’fixed’ correctly, at least for the HTML and BODY elements, since there is no way for an author to provide an image only for those browsers that support ’fixed’. See the section on conformance [p. 27] for details.

’background-position’

Property name:	’background-position’
Value:	[[<percentage> | <length>]{1,2} | [top | center | bottom] || [left | center | right]] | inherit
Initial:	0% 0%
Applies to:	block-level and replaced elements
Inherited:	no
Percentage values:	refer to the size of the element itself
Media groups:	visual [p. 65]

If a background image has been specified, this property specifies its initial position. Values have the following meanings:

<percentage> <percentage>
With a value pair of ’0% 0%’, the upper left corner of the image is aligned with the upper left corner of the box’s content edge [p. 72] . A value pair of ’100% 100%’ places the lower right corner of the image in the lower right corner of the box. With a value pair of ’14% 84%’, the point 14% across and 84% down the image is to be placed at the point 14% across and 84% down the box.

<length> <length>
With a value pair of ’2cm 2cm’, the upper left corner of the image is placed 2cm to the right and 2cm below the upper left corner of the box.

’top left’ and **’left top’**
Same as ’0% 0%’.

’top’, **’top center’**, and **’center top’**
Same as ’50% 0%’.

’right top’ and **’top right’**
Same as ’100% 0%’.

’left’, **’left center’**, and **’center left’**
Same as ’0% 50%’.

’center’ and **’center center’**
Same as ’50% 50%’.

’right’, **’right center’**, and **’center right’**
Same as ’100% 50%’.

’bottom left’ and **’left bottom’**
Same as ’0% 100%’.

’bottom’, **’bottom center’**, and **’center bottom’**
Same as ’50% 100%’.

’bottom right’ and **’right bottom’**
Same as ’100% 100%’.

If only one percentage or length value is given, it sets the horizontal position only, the vertical position will be 50%. If two values are given, the horizontal position comes first. Combinations of length and percentage values are allowed, (e.g., ’50% 2cm’). Negative positions are allowed. Keywords cannot be combined with percentage values or length values (all possible combinations are given above).

Examples:

```
BODY { background: url(banner.jpeg) right top }    /* 100%   0% */
BODY { background: url(banner.jpeg) top center }   /*  50%   0% */
BODY { background: url(banner.jpeg) center }       /*  50%  50% */
BODY { background: url(banner.jpeg) bottom }       /*  50% 100% */
```

If the background image is fixed within the viewport (see the ’background-attachment’ property), the image is placed relative to the viewport instead of the element’s box. For example,

```
BODY {
  background-image: url(logo.png);
  background-attachment: fixed;
  background-position: 100% 100%;
}
```

In the example above, the image is placed in the lower-right corner of the viewport.

’background’

Property name: ’background’
Value: [<’background-color’> || <’background-image’> || <’background-repeat’> || <’background-attachment’> || <’background-position’>] | inherit
Initial: not defined for shorthand properties
Applies to: all elements
Inherited: no
Percentage values: allowed on ’background-position’
Media groups: visual [p. 65]

The ’background’ property is a shorthand property for setting the individual background properties (i.e., ’background-color’, ’background-image’, ’background-repeat’, ’background-attachment’ and ’background-position’) at the same place in the style sheet.

The ’background’ property always sets all the individual background properties. The ’background’ property helps authors remember to specify all aspects of a background which they might otherwise neglect by using the individual background properties.

In the first rule of the following example, only a value for ’background-color’ has been given and the other individual properties are set to their initial value. In the second rule, all individual properties have been specified.

```
BODY { background: red }
P { background: url(chess.png) gray 50% repeat fixed }
```

14 Fonts

Contents

1. Introduction [p. 148]
2. Font specification [p. 149]
    1. Font specification properties [p. 149]
    2. Font family [p. 150] : the ’font-family’
    3. Font style [p. 151] : the ’font-style’, ’font-variant’, and ’font-weight’ properties
    4. Font size [p. 154] : the ’font-size’ and ’font-size-adjust’ properties
    5. Shorthand font property [p. 158] : the ’font’ property
    6. Generic font families [p. 159]
        1. serif [p. 160]
        2. sans-serif [p. 160]
        3. cursive [p. 161]
        4. fantasy [p. 161]
        5. monospace [p. 161]
3. Font selection [p. 162]
    1. Font Descriptions and @font-face [p. 163]
    2. Descriptors for Selecting a Font [p. 165] : ’font-family’, ’font-style’, ’font-variant’, ’font-weight’, and ’font-size’
    3. Descriptors for Font Data Qualification [p. 166] : ’unicode-range’
    4. Descriptor for Numeric Values [p. 168] : ’units-per-em’
    5. Descriptor for Referencing [p. 168] : ’src’
    6. Descriptors for Matching [p. 169] : ’panose-1’, ’stemv’, ’stemh’, ’slope’, ’cap-height’, ’x-height’, ’ascent’, and ’descent’
    7. Descriptors for Synthesis [p. 171] : ’widths’ and ’definition-src’
    8. Descriptors for Alignment [p. 172] : ’baseline’, ’centerline’, ’mathline’, and ’topline’
4. Font Characteristics [p. 173]
    1. Introducing Font Characteristics [p. 173]
    2. Adorned font name [p. 174]
    3. Central Baseline [p. 175]
    4. Co-ordinate units on the em square [p. 175]
    5. Font encoding tables [p. 176]
    6. Font family name [p. 176]
    7. Glyph Representation widths [p. 176]
    8. Horizontal stem width [p. 176]
    9. Height of capital glyph representations [p. 176]
    10. Height of lowercase glyph representations [p. 177]
    11. Lower Baseline [p. 177]
    12. Mathematical Baseline [p. 178]
    13. Maximal bounding box [p. 178]
    14. Maximum unaccented height [p. 178]
    15. Maximum unaccented depth [p. 178]
    16. Panose-1 number [p. 178]

17. Range of ISO10646characters [p. 179]
18. Top Baseline [p. 180]
19. Vertical stem width [p. 180]
20. Vertical stroke angle [p. 180]

5. Font matching algorithm [p. 180]
    1. Examples of font matching [p. 182]

14.1 Introduction

When a document's text is to be displayed visually, each character (abstract information element) must be mapped to some representation that may be drawn on the screen, paper, etc. Each of these glyphs constitutes a graphical depiction of a character. (as opposed to, for example, an aural, textual, or numerical depiction of that character) One or more characters may be depicted by one or more glyphs, in a possibly context-dependent fashion. A *glyph representation* is the actual artistic representation of an abstract glyph, in some typographic style, in the form of outlines or bitmaps. A font is a set of glyph representations, all observing same basic motif according to design, size, appearance, and other attributes associated with the entire set.

A visual user agent must address the following issues before actually rendering a character:

- Has the author specified a font for this character?
- Does the client's user agent have this font available?
- If so, what glyph or glyphs does this character (or sequence of characters) map to?
- If not, what should be done? Should a different font be substituted? Can the font be synthesized? Can it be retrieved from the Web?

In both CSS1 and CSS2, authors specify font characteristics via a series of font properties.

What use the user agent makes of these properties differs greatly between CSS1 and CSS2. In CSS1, all fonts were assumed to be present on the client system and were identified solely by name. Alternate fonts could be specified with the properties, but beyond that, user agents had no way to suggest any other fonts (even stylistically similar fonts that the user agent had available) other than generic default fonts.

CSS2 changes all that, and allows user agents much greater liberty in selecting a font when an author's requested font is not immediately available. CSS2 improves client-side font matching, enables font synthesis and progressive rendering, and enables fonts to be downloaded over the Web.

In the CSS font model, each user agent has a "font database" at its disposition. CSS2 allows style sheet authors to contribute towards that database. When asked to display a character with a particular font, the user agent first identifies the font in the database that "best fits" the specified font (according to the font matching algorithm) [p. 180] Once it has identified a font, it retrieves the font data locally or from the Web, and may display the character using those glyph representations.

In light of this simple model, we have organized the specification into two sections. The first concerns the font specification mechanism [p. 149] , whereby authors specify which fonts they would like to have used. The second concerns the font selection mechanism [p. 162] , whereby the client's user agent identifies and loads a font that best fits the author's specification.

How the user agent constructs the font database lies outside the scope of this specification since the database's implementation depends on the operating system, the windowing system, the client, etc. Similarly, this specification does not mandate how the user agent should handle error conditions such as when none of the desired fonts are available.

14.2 Font specification

The first phase of the CSS font mechanism concerns how authors specify which fonts should be used by a client user agent. Unfortunately, there exists no well-defined and universally accepted taxonomy for classifying fonts, and terms that apply to one font family may not be appropriate for others. For example, the term 'italic' is commonly used to label slanted text, but slanted text may also be labeled *Oblique, Slanted, Incline, Cursive* or *Kursiv*.

Since it is not possible to provide authors with a perfect font naming scheme, CSS has authors refer to pertinent characteristics of a font through a series of properties. The property values form the basis of the user agent's font selection [p. 162] .

14.2.1 Font specification properties

CSS2 specifies fonts by using the following properties:

Font family
: A font family is a group of fonts that resemble one another. One member of the family may be italic, another other bold, another bold and italic, etc. Examples of font family names include Helvetica, New Century Schoolbook, Kyokasho ICA L. Font family names are not restricted to Latin characters. Font families may be grouped into different categories: those with or without serifs, those whose characters are or are not proportionally spaced, those that resemble handwriting, those that are fantasy fonts, etc.

Font style
: The font style specifies whether the specified font is normal, italic, or oblique (italic and oblique fonts are similar, but not the same, especially for fonts with serifs).

Font variant
: The font variant indicates whether the font contains normal upper and lower case characters or whether it contains small-caps characters.

Font weight
: The font weight refers to the boldness or lightness of a font's glyphs.

Font size
: The font size refers to the size of the font.

On all properties except ’font-size’, ’em’ and ’ex’ length values refer to the font size of the current element. For ’font-size’, these length units refer to the font size of the parent element. Please consult the section on length units [p. 36] for more information.

For information about the classification of fonts in general, please consult the section on font descriptors [p. 173] .

14.2.2 Font family: the ’font-family’

’font-family’

Property name:	’font-family’
Value:	[[[<family-name> | <generic-family>],]* [<family-name> | <generic-family>]] | inherit
Initial:	depends on user agent
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

This property specifies a prioritized list of font family names and/or generic family names. To deal with the problem that a single font may not be enough to display all the characters in a document, or even a single element, this property allows authors to specify a list of fonts, all of the same style and size, that are tried in sequence to see if they contain a glyph for a certain character. This list is called a *font set*.

For example, text that contains English text mixed with mathematical symbols may need a font set of two fonts, one containing letters and digits, the other containing mathematical symbols. Here is an example of a font set suitable for a text that is expected to contain text with Latin characters, Japanese characters, and mathematical symbols:

```
BODY { font-family: Baskerville, "Heisi Mincho W3", Symbol, serif }
```

The characters available in the Baskerville font (a font with only Latin characters) will be taken from that font, Japanese will be taken from Heisi Mincho W3, and the mathematical symbols will come from Symbol. Any other characters will (hopefully) come from the generic font family ’serif’. The ’serif’ font family will also be used if one or more of the other fonts is unavailable.

There are two types of list values:

<family-name>
: The name of a font family of choice. In the last example, "gill" and "Helvetica" are font families.

<generic-family>
: In the example above, the last value is a generic family name [p. 159] . The following generic families are defined: ’serif’,’sans-serif’, ’cursive’, ’fantasy’ and ’monospace’.

Authors are encouraged to offer a generic font family as a last alternative.

Font names containing whitespace [p. 31] should be quoted.
For example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<STYLE type="text/css">
  BODY { font-family: "new century schoolbook", serif }
</STYLE>
</HEAD>
<BODY style="font-family: 'My own font', fantasy">
<P>What's up, Doc?
</BODY>
</HTML>
```

If quoting is omitted, any whitespace [p. 31] characters before and after the font name are skipped [p. 31] and any sequence of whitespace characters inside the font name is converted to a single space.

The generic font family values are considered keywords and therefore must not be quoted.

14.2.3 Font style: the 'font-style', 'font-variant', and 'font-weight' properties

'font-style'

Property name:	'font-style'
Value:	normal | italic | oblique | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

The 'font-style' property selects between normal (sometimes referred to as "roman" or "upright"), italic and oblique faces within a font family.

A value of 'normal' selects a font that is classified as 'normal' in the UA's font database, while 'oblique' selects a font that is labeled 'oblique'. A value of 'italic' selects a font that is labeled 'italic', or, if that is not available, one labeled 'oblique'.

The font that is labeled 'oblique' in the UA's font database may actually have been generated by electronically slanting a normal font.

Fonts with Oblique, Slanted or Incline in their names will typically be labeled 'oblique' in the font database. Fonts with *Italic, Cursive* or *Kursiv* in their names will typically be labeled 'italic'.

```
H1, H2, H3 { font-style: italic }
H1 EM { font-style: normal }
```

In the example above, normal text in an H1, H2, or H3 element will be displayed with an italic font. However, emphasized text within H1 will appear in a normal face.

’font-variant’

Property name:	’font-variant’
Value:	normal | small-caps | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

In a small-caps font, the lower case letters look similar to the uppercase ones, but in a smaller size and with slightly different proportions. The ’font-variant’ property selects that font. This property has no visible effect for scripts which are unicameral (having only one case).

A value of ’normal’ selects a font that is not labeled as a small-caps font, ’small-caps’ selects a small-caps font. If a genuine small-caps font is not available, it is acceptable (but not required) in CSS2 if the small-caps font is a created by taking a normal font and replacing the lower case letters by scaled uppercase characters. As a last resort, unscaled uppercase letters will be used as replacement for a small-caps font so that the text appears in all capitals.

The following example results in an H3 element in small-caps, with emphasized words in oblique small-caps:

```
H3 { font-variant: small-caps }
EM { font-style: oblique }
```

There may be other variants in the font family as well, such as fonts with old-style numerals, small-caps numerals, condensed or expanded letters, etc. CSS2 has no properties that select those.

Insofar as this property causes text to be transformed to uppercase, the same considerations as for ’text-transform’ apply.

’font-weight’

Property name:	’font-weight’
Value:	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

The ’font-weight’ property selects the weight of the font. The values ’100’ to ’900’ form an ordered sequence, where each number indicates a weight that is at least as dark as its predecessor. The keyword ’normal’ is synonymous with ’400’, and ’bold’ is synonymous with ’700’. Keywords other than ’normal’ and ’bold’ have been shown to be often confused with font names and a numerical scale

was therefore chosen for the 9-value list.

```
P { font-weight: normal }   /* 400 */
H1 { font-weight: 700 }     /* bold */
```

The ’bolder’ and ’lighter’ values select font weights that are relative to the weight inherited from the parent:

```
STRONG { font-weight: bolder }
```

Child elements inherit the resultant weight, not the keyword value.

Fonts (the font data) typically have one or more properties whose values are names that are descriptive of the "weight" of a font. There is no accepted, universal meaning to these weight names. Their primary role is to distinguish faces of differing darkness within a single font family. Usage across font families is quite variant; for example a font that you might think of as being bold might be described as being *Regular, Roman, Book, Medium, Semi-* or *DemiBold, Bold,* or *Black,* depending on how black the "normal" face of the font is within the design. Because there is no standard usage of names, the weight property values in CSS2 are given on a numerical scale in which the value ’400’ (or ’normal’) corresponds to the "normal" text face for that family. The weight name associated with that face will typically be *Book, Regular, Roman, Normal* or sometimes *Medium*.

The association of other weights within a family to the numerical weight values is intended only to preserve the ordering of darkness within that family. However, the following heuristics tell how the assignment is done in typical cases:

- If the font family already uses a numerical scale with nine values (as e.g. *OpenType* does), the font weights should be mapped directly.
- If there is both a face labeled *Medium* and one labeled *Book, Regular, Roman* or *Normal,* then the *Medium* is normally assigned to the ’500’.
- The font labeled "Bold" will often correspond to the weight value ’700’.
- If there are fewer then 9 weights in the family, the default algorithm for filling the "holes" is as follows. If ’500’ is unassigned, it will be assigned the same font as ’400’. If any of the values ’600’, ’700’, ’800’ or ’900’ remains unassigned, they are assigned to the same face as the next darker assigned keyword, if any, or the next lighter one otherwise. If any of ’300’, ’200’ or ’100’ remains unassigned, it is assigned to the next lighter assigned keyword, if any, or the next darker otherwise.

The following two examples illustrate the process. Assume four weights in the "Example1" family, from lightest to darkest: *Regular, Medium, Bold, Heavy.* And assume six weights in the "Example2" family: *Book, Medium, Bold, Heavy, Black, ExtraBlack.* Note how in the second example it has been decided *not* to assign "Example2 ExtraBlack" to anything.

Available faces	Assignments	Filling the holes
"Example1 Regular"	400	100, 200, 300
"Example1 Medium"	500	
"Example1 Bold"	700	600
"Example1 Heavy"	800	900

```
Available faces       |  Assignments  |  Filling the holes
----------------------+---------------+-------------------
"Example2 Book"       |  400          |  100, 200, 300
"Example2 Medium"     |  500          |
"Example2 Bold"       |  700          |  600
"Example2 Heavy"      |  800          |
"Example2 Black"      |  900          |
"Example2 ExtraBlack" |  (none)       |
```

Since the intent of the relative keywords ’bolder’ and ’lighter’ is to darken or lighten the face *within the family* and because a family may not have faces aligned with all the symbolic weight values, the matching of ’bolder’ is to the next darker face available on the client within the family and the matching of ’lighter’ is to the next lighter face within the family. To be precise, the meaning of the relative keywords ’bolder’ and ’lighter’ is as follows:

- ’bolder’ selects the next weight that is assigned to a font that is darker than the inherited one. If there is no such weight, it simply results in the next darker numerical value (and the font remains unchanged), unless the inherited value was ’900’ in which case the resulting weight is also ’900’.
- ’lighter’ is similar, but works in the opposite direction: it selects the next lighter keyword with a different font from the inherited one, unless there is no such font, in which case it selects the next lighter numerical value (and keeps the font unchanged).

There is no guarantee that there will be a darker face for each of the ’font-weight’ values; for example, some fonts may have only a normal and a bold face, others may have eight different face weights. There is no guarantee on how a UA will map font faces within a family to weight values. The only guarantee is that a face of a given value will be no less dark than the faces of lighter values.

14.2.4 Font size: the ’font-size’ and ’font-size-adjust’ properties

’font-size’

Property name:	’font-size’
Value:	<absolute-size> | <relative-size> | <length> | <percentage> | inherit
Initial:	medium
Applies to:	all elements
Inherited:	yes
Percentage values:	relative to parent element’s font size
Media groups:	visual [p. 65]

<absolute-size>

An <absolute-size> keyword is an index to a table of font sizes computed and kept by the UA. Possible values are:

[xx-small | x-small | small | medium | large | x-large | xx-large]

On a computer screen a scaling factor of 1.5 is suggested between adjacent indexes; if the ’medium’ font is 10pt, the ’large’ font could be 15pt. Different media may need different scaling factors. Also, the UA should take the quality and availability of fonts into account when computing the table. The table may be different from one font family to another.

<relative-size>

A <relative-size> keyword is interpreted relative to the table of font sizes and the font size of the parent element. Possible values are:

[larger | smaller]

For example, if the parent element has a font size of ’medium’, a value of ’larger’ will make the font size of the current element be ’large’. If the parent element’s size is not close to a table entry, the UA is free to interpolate between table entries or round off to the closest one. The UA may have to extrapolate table values if the numerical value goes beyond the keywords.

Length and percentage values should not take the font size table into account when calculating the font size of the element.

Negative values are not allowed.

An application may reinterpret an explicit size, depending on the context, for example, inside a VR scene a font may get a different size because of perspective distortion.

Examples:

```
P { font-size: 12pt; }
BLOCKQUOTE { font-size: larger }
EM { font-size: 150% }
EM { font-size: 1.5em }
```

’font-size-adjust’

Property name:	’font-size-adjust’
Value:	z | none | inherit
Initial:	none
Applies to:	all elements
Inherited:	yes, but not adjusted values
Percentage values:	relative to parent element’s font size
Media groups:	visual [p. 65]

This property preserves the x-height of the first choice font in substituted fonts. Values have the following meanings:

z

This value refers to the ratio of em- to x-heights of the "first choice" font in decimal notation.

none

Do not preserve the font’s x-height.

For example, the popular font Verdana has a z value of 1.72; in an instance of Verdana at 100 pt (em-height), the x-height will be 58 pt (100/58=1.72). For comparison, Times New Roman has a z value of 2.17.

This property uses z to compute an appropriate scaling factor for available fonts, according to the following formula:

```
a(z′/z) = b
```

where:

```
a = ′font-size′ of first-choice font
z′ = z of available font
b = ′font-size′ to apply to available font
```

The subjective "bigness" and general character of a font on screen is less dependent on the size of its em than on the size of its x-height, or, more usefully, on its z. The lower the value of z, the more legible at smaller em sizes it will likely be. Inversely, faces with a higher z value will become illegible more rapidly below a given threshold size than faces with a lower z. Substitution can easily fall foul of this threshold.

The first image below shows several typefaces as rasterized on a Macintosh at 12 point, together with their z values. No matter what the typographical family (or nominal base size, unit, or platform), faces with lower z look bigger than those with higher z. Faces with very high z are illegible at the size shown.

The next image below shows the results of 'font-size-adjust' with the fonts shown in the previous image. These fonts' em sizes range all the way up to 25 pt, but appear subjectively to be about as large as 12-pt Verdana, whose z served as the base for scaling. As adjusted, the legibility characteristics of these faces on screen are nearly linear across faces. Note that 'font-size-adjust' tends to stabilize the horizontal metrics of lines, as well.

14.2.5 Shorthand font property: the ’font’ property

’font’

Property name:	’font’
Value:	[[<’font-style’> || <’font-variant’> || <’font-weight’>]? <’font-size’> [/ <’line-height’>]? <’font-family’>] | caption | icon | menu | messagebox | smallcaption | statusbar | inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	yes
Percentage values:	allowed on ’font-size’ and ’line-height’
Media groups:	visual [p. 65]

The ’font’ property is a shorthand property for setting ’font-style’, ’font-variant’, ’font-weight’, ’font-size’, ’line-height’, and ’font-family’, at the same place in the style sheet. The syntax of this property is based on a traditional typographical shorthand notation to set multiple properties related to fonts.

For a definition of allowed and initial values, see the previously defined properties. Properties for which no values are given are set to their initial value.

The following values refer to system fonts:

caption
: Used for (captioned controls (e.g. buttons, drop-downs, etc.).

icon
: Used to label icons.

menu
: Used in menus (e.g., dropdown menus and menu lists).

messagebox
: Used in dialog boxes.

smallcaption
: Used for labeling small controls.

statusbar
: Used in window status bars.

System fonts can only be accessed as a whole; that is, the font-family, size, weight, style, etc. are all set at the same time. The semantic meaning of each of the values follows - if any of these semantics do not exist on a given platform, the UA should either intelligently substitute (e.g., a smaller version of the ’caption’ font might be used for the ’smallcaption’ font), or substitute its UA default font. As for regular fonts, if, for a system font, any of the individual properties are not part of the operating system’s available user preferences, those properties should be set to their initial values.

Examples:

```
P { font: 12pt/14pt sans-serif }
P { font: 80% sans-serif }
P { font: x-large/110% "new century schoolbook", serif }
P { font: bold italic large Palatino, serif }
P { font: normal small-caps 120%/120% fantasy }
P { font: messagebox }
INPUT.small { font: smallcaption }
```

In the second rule, the font size percentage value (’80%’) refers to the font size of the parent element. In the third rule, the line height percentage refers to the font size of the element itself.

In the first three rules above, the ’font-variant’ and ’font-weight’ are not explicitly mentioned, which means they are all three set to their initial value (’normal’). The fourth rule sets the ’font-weight’ to ’bold’, the ’font-style’ to ’italic’ and implicitly sets ’font-variant’ to ’normal’.

The fifth rule sets the ’font-variant’ (’small-caps’), the ’font-size’ (120% of the parent’s font), the ’line-height’ (120% times the font size) and the ’font-family’ (’fantasy’). It follows that the keyword ’normal’ applies to the two remaining properties: ’font-style’ and ’font-weight’.

The sixth and seventh rule set the font properties to the appropriate values for these elements in the user’s environment.

14.2.6 Generic font families

Generic font families are a fallback mechanism, a means of preserving some of the style sheet writer’s intent in the worst case when none of the specified fonts can be selected. For optimum typographic control, particular named fonts should be used in style sheets.

All five generic font families may be assumed to exist in all CSS implementations (they need not necessarily map to five distict actual fonts, in all cases). UAs should provide reasonable default choices for the generic font families, which express the characteristics of each family as well as possible within the limits of the underlying technology allows.

UAs are encouraged to allow users to select alternative choices for the generic fonts.

serif

Serif fonts, as the term is used in CSS, have the characteristic that the ends of the strokes have finishing strokes, flared or tapering ends, or have actual serifed endings (including slab serifs). Serif fonts are typically proportionately spaced. They often display a greater variation between thick and thin strokes than fonts from the ’sans-serif’ generic font family. CSS uses the term ’serif’ to apply to a font for any script, although other names may be more familiar for particular scripts, such as Mincho (Japanese), Sung or Song (Chinese), Totum or Kodig (Korean) and any font which is so described may be used to represent the generic ’serif’ family.

Examples of fonts which fit this description include:

Latin fonts
: Times New Roman, Bodoni, Garamond, Minion Web, ITC Stone Serif, MS Georgia, Bitstream Cyberbit

Greek fonts
: Bitstream Cyberbit

Cyrillic fonts
: Adobe Minion Cyrillic, Excelcior Cyrillic Upright, Monotype Albion 70, Bitstream Cyberbit, ER Bukinst

Hebrew fonts
: New Peninim, Raanana, Bitstream Cyberbit

Japanese fonts
: Ryumin Light-KL, Kyokasho ICA, Futo Min A101

Arabic fonts
: Bitstream Cyberbit

Cherokee fonts
: Lo Cicero Cherokee

sans-serif

Sans-serif fonts, as the term is used in CSS, have the characteristic that the ends of their strokes have abrupt or butted ends. Sans-serif fonts are typically proportionately spaced. They often have little variation between thick and thin strokes, compared to fonts from the ’serif’ family. CSS uses the term ’sans-serif’ to apply to a font for any script, although other names may be more familiar for particular scripts, such as Gothic (Japanese), Kai (Chinese), Pathang (Korean) and any font which is so described may be used to represent the generic ’sans-serif’ family.

Examples of fonts which fit this description include:

Latin fonts
: MS Trebuchet, ITC Avant Garde Gothic, MS Arial, MS Verdana, Univers, Futura, ITC Stone Sans, Gill Sans, Akzidenz Grotesk, Helvetica

Greek fonts
: Attika, Typiko New Era, MS Tahoma, Monotype Gill Sans 571, Helvetica Greek

Cyrillic fonts
: Helvetica Cyrillic, ER Univers, Lucida Sans Unicode, Bastion

Hebrew fonts
: Arial Hebrew, MS Tahoma

Japanese fonts
: Shin Go, Heisei Kaku Gothic W5

Arabic fonts
: MS Tahoma

cursive

Cursive fonts, as the term is used in CSS, have the characteristic that the glyphs are partially or completely connected, and that the result looks more like handwritten pen or brush writing than printed letterwork. Fonts for some scripts, such as Arabic, are almost always cursive. CSS uses the term ’cursive’ to apply to a font for any script, although other names such as Chancery, Brush, Swing and Script are also used in font names.

Examples of fonts which fit this description include:

Latin fonts
: Caflisch Script, Adobe Poetica, Sanvito, Ex Ponto, Snell Roundhand, Zapf-Chancery

Cyrillic fonts
: ER Architekt

Hebrew fonts
: Corsiva

Arabic fonts
: DecoType Naskh, Monotype Urdu 507

fantasy

Fantasy fonts, as used in CSS, are primarily decorative whilst still containing representations of characters (as opposed to Pi or Picture fonts, which do not represent characters).

Latin fonts
: Alpha Geometrique, Critter, Cottonwood, FB Reactor, Studz

monospace

The sole criterion of a monospace font is that all glyph representations have the same fixed width. This can make some scripts, such as Arabic, look most peculiar. The effect is similar to a manual typewriter, and is often used to simulate computer code.

Examples of fonts which fit this description include:

Latin fonts
: Courier, MS Courier New, Prestige, American Typewriter, Everson Mono

Greek Fonts
: MS Courier New, Everson Mono

Cyrillic fonts
: ER Kurier, Everson Mono

Japanese fonts
: Osaka Monospaced

Cherokee fonts
: Everson Mono

14.3 Font selection

The second phase of the CSS2 font mechanism concerns the user agent's selection of a font based on author-specified font properties, available fonts, etc. The details of the font matching algorithm [p. 180] are provided below.

There are four possible font selection actions: matching, intelligent matching, synthesis, and download.

- *font name matching*
  In this case, the user agent uses an existing, accessible font that has the same family name as the requested font (note that the appearance and the metrics might not necessarily match, if the font that the document author used and the font on the client system are from different foundries). The matching information is restricted to the CSS font properties, including the family name.
- *intelligent font name matching*
  In this case, the user agent uses an existing, accessible font that is the closest match in appearance to the requested font. (Note that the metrics might not match exactly). The matching information includes information about the kind of font (text or symbol), nature of serifs, weight, cap height, x height, ascent, descent, slant, etc.
- *font synthesis*
  In this case, the user agent creates a font that is not only a close match in appearance, but also matches the metrics of the requested font. The synthesizing information includes the matching information and typically requires more accurate values for the parameters than are used for some matching schemes. In particular, synthesis requires accurate width metrics and character to glyph substitution and position information if all the layout characteristics of the specified font are to be preserved.
- *Download*
  Finally, the user agent may retrieve a font over the Web. This is similar to the process of fetching images, sounds or applets over the Web for display in the current document, and likewise can cause some delay before the page can be displayed.

progressive rendering is a combination of download and one of the other methods; it provides a temporary substitute font (using name matching, intelligent matching, or synthesis) to allow content to be read while the requested font downloads. Once the real font has been successfully downloaded, it replaces the temporary font, hopefully without the need to reflow.

In CSS2, authors may specify which, if any, of these mechanisms should be invoked by the user agent if a particular font is not immediately available. Authors add *font descriptions* to style sheets for this purpose. A font description is a set of *font descriptors*, individual pieces of information about a font, possibly including a URI describing the font's location on the Web.

Note. *Progressive rendering requires metric information about the font in order to avoid re-layout of the content when the actual font has been loaded and rendered. This metric information is sufficiently verbose that it should only be specified at most once per font in a document.*

14.3.1 Font Descriptions and @font-face

The font description provides the bridge between an author's font specification and the *font data*, which is the data needed to format text and to render the glyph representations to which the characters map - the actual scalable outlines or bitmaps needed to render the glyph representations to which the characters map. Fonts are *referenced* by style sheet properties.

The *font description* is used to select the relevant font data. The font description contains descriptors that provide the location of the font data on the Web, and/or characterize that font data. The font descriptors are also needed to match the style sheet font properties to particular font data. The level of detail of a font description can vary from just the name of the font up to a list of glyph representation widths. This data is a subset of the glyph representation metrics contained in the font.

Font descriptors may be classified into three types:

1. those that provide the link between the CSS usage of the font and the font description (these have the same names as the corresponding CSS font properties),
2. the URI for the location of the font data,
3. those that further characterize the font, to provide a link between the font description and the font data.

All font descriptions are specified via a *@font-face* at-rule. The general form of this rule is:

`@font-face {<font-description> }`

where the <font-description> has the form:

```
descriptor: value;
descriptor: value;
[...]
descriptor: value;
```

Each @font-face rule specifies a value for every font descriptor, either implicitly or explicitly. Those not given explicit values in the rule take the initial value listed with each descriptor in this specification. These descriptors apply solely within the context of the @font-face rule in which they are defined, and do not apply to

document language elements. Thus, there is no notion of which elements the descriptors apply to, or whether the values are inherited by child elements.

The available font descriptors are described in later sections of this specification.

For example, here the font ’Robson Celtic’ is defined and referenced in a style sheet contained in an HTML document.

```
<HTML>
  <HEAD>
    <TITLE>Font test</TITLE>
    <STYLE TYPE="text/css" MEDIA="screen, print">
      @font-face {
        font-family: "Robson Celtic";
        src: url(http://site/fonts/rob-celt)
      }
      H1 {font-family: "Robson Celtic", serif}
    </STYLE>
  </HEAD>

  <BODY>
    <H1> This heading is displayed using Robson Celtic</H1>
  </BODY>
</HTML>
```

The style sheet (in the STYLE element) contains a CSS rule that sets all H1 elements to use the ’Robson Celtic’ font family.

A CSS1 implementation will search the client for a font whose family name and other properties match "Robson Celtic" and, if it fails to find it, will use the UA-specific fallback serif font (which is defined to exist [p. 150]).

A user agent implementing CSS2 will first examine @font-face rules in search of a font description defining Robson Celtic. This example contains a rule which matches. Although this rule doesn’t contain much font data, it does have a URI where the font can be retrieved for rendering this document. Downloaded fonts should not be made available to other applications. If no matching @font-face is found, the user agent will attempt the same match as a user agent implementing CSS1.

Note that if the font Robson Celtic *had* been installed on the client system, this would cause the UA to construct an @font-face rule for the installed copy as described in the section on the font matching algorithm [p. 180] . The installed copy would have been matched before the downloadable font in the example above.

CSS1 implementations, which do not understand the @font-face rule will encounter the opening curly brackets and will skip [p. 31] forward until the matching closing curly brackets. This at-rule conforms with the forward-compatible parsing [p. 29] requirement of CSS. Parsers may skip [p. 31] these rules without error.

Also, any descriptors which are not recognized or useful to the user agent should be skipped [p. 31] . This allows adding in the future optional descriptors for the purpose of better font substitution, matching, or synthesis.

14.3.2 Descriptors for Selecting a Font: ’font-family’, ’font-style’, ’font-variant’, ’font-weight’, and ’font-size’

The following descriptors have the same names as the corresponding CSS2 font properties, and take a single value or comma-separated list of values.

The values within that list are exactly the same as those specified for CSS2. If there is a single value, that is the value that must be matched. If there is a list, any of the list items constitutes a match. If the descriptor is omitted from the @font-face, the initial value is used.

’font-family’ (Descriptor)

Descriptor name:	’font-family’
Value:	[<family-name> | <generic-family>] [, [<family-name> | <generic-family>]]*
Initial:	depends on user agent
Applies to media:	visual

This is the descriptor for the family name [p. 150] of a font and takes the same values as the ’font-family’ property.

’font-style’ (Descriptor)

Descriptor name:	’font-style’
Value:	[normal | italic | oblique] [, [normal | italic | oblique]]*
Initial:	normal
Applies to media:	visual

This is the descriptor for the style of a font and takes the same values as the ’font-style’ property except that a comma separated list is permitted. The value ’normal’ indicates that this is the normal face of a font; it is either the only face in a font, or it is the face which is intended to be used alongside other companion faces. The value ’oblique’ indicates that this face is a more slanted companion face than the normal face. The value ’italic’ indicates that this is a more cursive companion face to the normal face. This avoids having to label slightly slanted normal faces as oblique, or Greek faces as italic.

’font-variant’ (Descriptor)

Descriptor name:	’font-variant’
Value:	[normal | small-caps] [,[normal | small-caps]]*
Initial:	normal
Applies to media:	visual

This is the CSS indication whether this face is a small-caps variant of a font. It takes the same values as the ’font-variant’ property except that a comma separated list is permitted. Cyrillic *pryamoĭ* faces may be labeled with a ’font-variant’ of small-caps, which will give better consistency with Latin faces (and the companion *kursiv* face labeled with ’font-style’ italic for the same reason).

’font-weight’ (Descriptor)

Descriptor name:	’font-weight’
Value:	all | [normal | bold | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900] [, [normal | bold | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900]]*
Initial:	normal
Applies to media:	visual

This is the descriptor for the weight of a face relative to others in the same font family. It takes the same values as the ’font-weight’ property with three exceptions:

1. relative keywords (bolder, lighter) are not permitted
2. a comma separated list of values is permitted
3. an additional keyword, ’all’ is permitted

’font-size’ (Descriptor)

Descriptor name:	’font-size’
Value:	all | [<length> [,[<length>]]*]
Initial:	all
Applies to media:	visual

This is the descriptor for the sizes provided by this font. Only absolute length [p. 36] units are permitted, in contrast to the ’font-size’ property, which allows both relative and absolute lengths and sizes. A comma separated list of absolute lengths is permitted.

The initial value of ’all’ is suitable for scalable fonts, so this descriptor will only be useful in an @font-face for bitmap fonts, or for scalable fonts which have hand-tuned bitmaps at specific point sizes.

14.3.3 Descriptors for Font Data Qualification: ’unicode-range’

The following descriptor is optional within a font definition, but is used to avoid checking or downloading a font that does not have sufficient glyphs to render a particular character.

’unicode-range’ (Descriptor)

Descriptor name:	’unicode-range’
Value:	<urange>+
Initial:	U+0-7FFFFFFF
Applies to media:	visual

This is the descriptor for the range of [ISO10646] [p. 267] characters covered by the font. Since this is sparse (most fonts do not cover the whole of [ISO10646] [p. 267]) this descriptor lists blocks or ranges which do have *some* coverage (no

promise is made of complete coverage). This method is extensible to future allocation of characters in Unicode, without change of syntax and without invalidating existing content.

The values of <urange> are expressed using hexadecimal numbers prefixed by "U+", corresponding to character code positions in [ISO10646] [p. 267] (which specifies the document character set of [HTML40] [p. 268]). For example, `U+05D1` is the [ISO10646] [p. 267] character 'Hebrew letter bet'. For values outside the Basic Multilingual Plane (BMP), additional leading digits corresponding to the plane number are added, also in hexadecimal, like this: `U+A1234` which is the character on Plane 10 at hexadecimal code position 1234. At the time of writing no characters had been assigned outside the BMP. Leading zeros (for example, 0000004D) are legal, but not required.

The initial value (i.e., the value used when no value is given in the style sheet) covers not only the entire Basic Multilingual Plane (BMP), which would be expressed as U+0-FFFF, but also the whole repertoire of ISO 10646. Thus, the initial value says that the font may have glyph representations for characters anywhere in [ISO10646] [p. 267] . Specifying a value for 'unicode-range' provides information to make searching efficient, by declaring a constrained range in which the font may have glyph representations for characters. The font need not be searched for characters outside this range.

Values may be written with any number of digits. For single numbers, the character '?' is assumed to mean 'any value' which creates a *range* of character positions. Thus, using a *single number*:

unicode-range: U+20A7
: no wild cards - it indicates a single character position (the Spanish peseta currency symbol)

unicode-range: U+215?
: one wild card, covers the range 2150 to 215F (the fractions)

unicode-range: U+00??
: two wild cards, covers the range 0000 to 00FF (Latin-1)

unicode-range: U+E??
: two wild cards, covers 0E00 to 0EFF (the Lao script)

A *pair of numbers* in this format can be combined with the dash character to indicate larger ranges. For example

unicode-range: U+AC00-D7FF
: the range is AC00 to D7FF (the Hangul Syllables area)

Multiple, discontinuous ranges can be specified, separated by a comma. As with other comma-separated lists in CSS, any whitespace [p. 31] before or after the comma is skipped. [p. 31]

For example:

unicode-range: U+370-3FF, U+1F??
: This covers the range 0370 to 03FF (Modern Greek) plus 1F00 to 1FFF (Ancient polytonic Greek).

unicode-range: U+3000-303F, U+3100-312F, U+32??, U+33??, U+4E00-9FFF, U+F9000-FAFF, U+FE30-FE4F
: Something of a worst case in terms of verbosity, this very precisely indicates

that this (extremely large) font contains only Chinese characters from [ISO10646] [p. 267] , without including any characters that are uniquely Japanese or Korean. The range is 3000 to 303F (CJK symbols and punctuation) plus 3100 to 312F (Bopomofo) plus 3200 to 32FF (enclosed CJK letters and months) plus 3300 to 33FF (CJK compatibility zone) plus 4E00 to 9FFF (CJK unified Ideographs) plus F900 to FAFF (CJK compatibility ideographs) plus FE30 to FE4F (CJK compatibility forms).

A more likely representation for a typical Chinese font would be:

unicode-range: U+3000-33FF, U+4E00-9FFF

unicode-range: U+11E00-121FF

This font covers a proposed registration for Aztec pictograms, covering the range 1E00 to 21FF in plane 1.

unicode-range: U+1A00-1A1F

This font covers a proposed registration for Irish Ogham covering the range 1A00 to 1A1F

14.3.4 Descriptor for Numeric Values: ’units-per-em’

The following descriptor is optional within a font definition, but is required if there are any numeric values in the ’em’ space in which glyphs are defined.

’units-per-em’ (Descriptor)

Descriptor name: ’units-per-em’
Value: <number>
Initial: undefined
Applies to media: visual

This is the descriptor for the number of the co-ordinate units on the em square [p. 175] , the size of the design grid on which glyph representations are laid out.

14.3.5 Descriptor for Referencing: ’src’

This descriptor is required for referencing actual font data, whether downloadable or locally installed.

’src’ (Descriptor)

Descriptor name: ’src’
Value: [<uri> [format [,format]*]? | <font-face-name>] [, <uri> [format [,format]*]?] | <font-face-name>]*
Initial: undefined
Applies to media: visual

This is a prioritized list of URIs and/or locally installed font face names. The URI points to the font data itself. This is required if the WebFont is to be retrieved. The font resource may be a subset of the source font. The URI may be partial, in which case it is resolved relative to the location of the style sheet containing the @font-face.

The URI may have optional hints regarding the format of font resource to be found at that URI, and this information should be used by clients in format negotiation with the server. As with any hypertext reference, there may be other formats available, or the resource may have been moved; but the client has a better idea of what is likely to be there, in a more robust way than trying to parse filename extensions in URIs.

The <font-face-name> is the adorned font name of a locally installed font. The *adorned font name* is the name of the font as reported by the operating system and is the name most likely to be used in reader style sheets, or author style sheets on an intranet. Adornments such as bold, italic, underline are often used to select the appropriate font within a font family. For more information about adorned font names [p. 174] please consult the notes about fonts.

Examples:

`src: url(http://foo/bar)`
: a full URI and no information about the font format(s) available there

`src: local(BT Century 751 No. 2 Semi Bold Italic)`
: references a particular face of a locally installed font

`src: url(../fonts/bar) format(truedoc)`
: a partial URI which has a font available in TrueDoc format

`src: url(http://cgi-bin/bar?stuff) format(opentype, intellifont)`
: a full URI, in this case to a script, which can generate two different formats - OpenType and Intellifont

`src: local(T-26 Typeka Mix), url(http://site/magda-extra) format(type1)`
: two alternatives are given, firstly a locally installed font and secondly a downloadable font available in Type 1 format.

Access to locally installed fonts is via the <font-face-name>. The font face name is not truly unique, nor is it truly platform or font format independent, but at the moment it is the best way to identify font data. The use of the font face name can be made more accurate by providing an indication of the glyph complement required. This may be done by indicating the range of [ISO10646] [p. 267] character positions for which the font provides some glyph representations (see 'unicode-range').

14.3.6 Descriptors for Matching: 'panose-1', 'stemv', 'stemh', 'slope', 'cap-height', 'x-height', 'ascent', and 'descent'

These descriptors are optional for a CSS2 definition, but may be used if intelligent font matching is desired by the author.

'panose-1' (Descriptor)

Descriptor name: 'panose-1'
Value: [<number>]{10}
Initial: 0 0 0 0 0 0 0 0 0 0
Applies to media: visual

This is the descriptor for the Panose-1 number [p. 178] and consists of ten decimal numbers, separated by whitespace [p. 31] . A comma separated list is not permitted for this descriptor, because the Panose-1 system can indicate that a range of values are matched. The initial value is zero for each PANOSE digit, which means "any"; all fonts will match the Panose number if this value is used.

'stemv' (Descriptor)

Descriptor name: 'stemv'
Value: <number>
Initial: undefined
Applies to media: visual

This is the descriptor for the vertical stem width [p. 180] of the font. If the value is undefined, the descriptor is not used for matching. If this descriptor is used, the 'units-per-em' [p. 168] descriptor must also be used.

'stemh' (Descriptor)

Descriptor name: 'stemh'
Value: <number>
Initial: undefined
Applies to media: visual

This is the descriptor for the horizontal stem width [p. 176] of the font. If the value is undefined, the descriptor is not used for matching. If this descriptor is used, the 'units-per-em' [p. 168] descriptor must also be used.

'slope' (Descriptor)

Descriptor name: 'slope'
Value: <number>
Initial: 0
Applies to media: visual

This is the descriptor for the vertical stroke angle [p. 180] of the font.

'cap-height' (Descriptor)

Descriptor name: 'cap-height'
Value: <number>
Initial: undefined
Applies to media: visual

This is the descriptor for the number of the height of capital glyph representations [p. 176] of the font. If the value is undefined, the descriptor is not used for matching. If this descriptor is used, the 'units-per-em' [p. 168] descriptor must also be used.

'x-height' (Descriptor)

Descriptor name: 'x-height'
Value: <number>
Initial: undefined
Applies to media: visual

This is the descriptor for the height of lowercase glyph representations [p. 177] of the font. If the value is undefined, the descriptor is not used for matching. If this descriptor is used, the 'units-per-em' [p. 168] descriptor must also be used.

'ascent' (Descriptor)

Descriptor name: 'ascent'
Value: <number>
Initial: undefined
Applies to media: visual

This is the descriptor for the maximum unaccented height [p. 178] of the font. If the value is undefined, the descriptor is not used for matching. If this descriptor is used, the 'units-per-em' [p. 168] descriptor must also be used.

'descent' (Descriptor)

Descriptor name: 'descent'
Value: <number>
Initial: undefined
Applies to media: visual

This is the descriptor for the Maximum unaccented depth [p. 178] of the font. If the value is undefined, the descriptor is not used for matching. If this descriptor is used, the 'units-per-em' [p. 168] descriptor must also be used.

14.3.7 Descriptors for Synthesis: 'widths' and 'definition-src'

Synthesizing a font means, at minimum, matching the width metrics of the specified font. Therefore, for synthesis, this metric information must be available. Similarly, progressive rendering requires width metrics in order to avoid reflow of the content when the actual font has been loaded. Although the following descriptors are optional for a CSS2 definition, some are required if synthesizing (and progressive rendering) is desired by the author. Should the actual font become available, the substitution should be replaced by the actual font. Any of these descriptors which are present will be used to provide a better or faster approximation of the intended font.

Of these descriptors, the most important are the 'widths' descriptor and `bbox` which are used to prevent text reflow should the actual font become available. In addition, the descriptors in the set of descriptors required for matching [p. 169] can be used to provide a better synthesis of the actual font appearance.

'widths' (Descriptor)

Descriptor name: 'widths'
Value: [<urange>]? [<number>]+ [,[<urange>]? <number>]+]
Initial: undefined
Applies to media: visual

This is the descriptor for the number of the glyph representation widths [p. 176] . The value is a (comma separated list of) <urange> values followed by one or more glyph representation widths. If this descriptor is used, the 'units-per-em'

[p. 168] descriptor must also be used.

For example:

```
widths: U+4E00-4E1F 1736 1874 1692
```

In this instance a range of 32 characters is given, from 4E00 to 4E1F. The glyph corresponding to the first character (4E00) has a width of 1736, the second has a width of 1874 and the third, 1692. Because not enough widths have been provided, the last width replicates to cover the rest of the specified range. If too many widths are provided, the extras are skipped. [p. 31]

If the <urange> is omitted, a range of U+0-7FFFFFFF is assumed which covers all characters and their glyph representations

This descriptor cannot describe multiple glyphs corresponding to a single character, or ligatures of multiple characters. Thus, this descriptor can *only* be used for scripts which do not have contextual forms or mandatory ligatures. It is nevertheless useful in those situations. Scripts which require a one-to-many or many-to-many mapping of characters to glyphs cannot at present use this descriptor to enable font synthesis although they can still use font downloading or intelligent matching.

'definition-src' (Descriptor)

Descriptor name: 'definition-src'
Value: <uri>
Initial: undefined
Applies to media: visual

The font descriptors may either be within the font definition in the style sheet, or may be provided within a separate *font definition resource* identified by a URI. The latter approach can reduce network traffic when multiple style sheets reference the same fonts.

Having the font descriptors separate from the font data has a benefit beyond being able to do font selection and/or substitution. The data protection and replication restrictions on the font descriptors may be much weaker than on the full font data. Thus, it may be possible to locally install the font definition, or at least to have it in a local cache. This allows the abbreviated form of font definition within documents, but would not require accessing the full font definition over the Web more than once per named font.

14.3.8 Descriptors for Alignment: 'baseline', 'centerline', 'mathline', and 'topline'

These optional descriptors are used to align runs of different scripts with one another.

'baseline' (Descriptor)

Descriptor name: 'baseline'
Value: <number>
Initial: 0
Applies to media: visual

This is the descriptor for the lower baseline [p. 177] of a font. If this descriptor is given a non-default (non-zero) value, the 'units-per-em' [p. 168] descriptor must also be used.

'centerline' (Descriptor)

Descriptor name: 'centerline'
Value: <number>
Initial: undefined
Applies to media: visual

This is the descriptor for the central baseline [p. 175] of a font. If the value is undefined, the UA may employ various heuristics such as the midpoint of the ascent and descent values. If this descriptor is used, the 'units-per-em' [p. 168] descriptor must also be used.

'mathline' (Descriptor)

Descriptor name: 'mathline'
Value: <number>
Initial: undefined
Applies to media: visual

This is the descriptor for the mathematical baseline [p. 178] of a font. If undefined, the UA may use the center baseline. If this descriptor is used, the 'units-per-em' [p. 168] descriptor must also be used.

'topline' (Descriptor)

Descriptor name: 'topline'
Value: <number>
Initial: undefined
Applies to media: visual

This is the descriptor for the top baseline [p. 180] of a font. If undefined, the UA may use an approximate value such as the ascent. If this descriptor is used, the 'units-per-em' [p. 168] descriptor must also be used.

14.4 Font Characteristics

14.4.1 Introducing Font Characteristics

In this section are listed the font characteristics that have been found useful for client-side font matching, synthesis, and download for heterogeneous platforms accessing the Web. The data may be useful for any medium which needs to use fonts on the Web by some other means than physical embedding of the font data inside the medium.

These characteristics are used to characterize fonts. They are not specific to CSS or to style sheets. In CSS, each characteristic is described by a font descriptor. These definitions could also be mapped onto VRML nodes, or CGM Application Structures, or a Java API, or alternative style sheet languages. Fonts retrieved by one medium and stored in a proxy cache could be re-used by another medium, saving download time and network bandwidth.

A non-exhaustive list examples of such media includes:

- 2-D vector formats
  - Computer Graphics Metafile
  - Simple Vector Format
- 3-D graphics formats
  - VRML
  - 3DMF
- Object embedding technologies
  - Java
  - Active-X
  - Obliq

14.4.2 Adorned font name

This is the full name of a particular face of a font family. It typically includes a variety of non-standardized textual qualifiers or *adornments* appended to the font family name. It may also include a foundry name or abbreviation, often prepended to the font family name. It is only used in the 'src' descriptor, to refer to locally installed fonts, because the format of the adorned name can vary from platform to platform.

The name of the font definition is important because it is the link to any locally installed fonts. It is important that the name be robust, both with respect to platform and application independence. For this reason, the name should be one which is not application or language specific.

The ideal solution would be to have a name which uniquely identifies each collection of font data. This name does not exist in current practice for font data. Fonts with the same face name can vary over of number of descriptors. Some of these descriptors, such as different complements of glyphs in the font may be insignificant if the needed glyphs are in the font. Other descriptors, such as different width metrics, make fonts with the same name incompatible. It does not seem possible to define a rule that will always identify incompatibilities, but will not prevent the use of a perfectly suitable local copy of the font data with a given name. Therefore, only the range of [ISO10646] [p. 267] characters will be used to qualify matches for the font face name.

Since a prime goal of the font face name in the font definition is allow a user agent to determine when there is a local copy of the specified font data, the font face name must be a name which will be in all legitimate copies of the font data. Otherwise, unnecessary Web traffic may be generated due to missed matches for the local copy.

For TrueType and OpenType fonts, this value may be obtained from the `full font name` from the `name` table.

For Type 1 fonts, this value may be obtained from the PostScript language name; the name which, in a PostScript language program, is used as an operand of the findfont operator. It is the name associated with the font by a definefont operation. This is usually the value of the FontName entry in the font dictionary. For more information, see Section 5.2 of the PostScript Language Reference Manual, Second Edition.

Multiple Master Type 1 fonts allow specifying various design dimensions (e.g., weight, such as light to extra-bold, and width, such as condensed to expanded). Coordinates along these design dimensions are specified by numbers, and are appended as a suffix to the base font name. To specify the appearance of the font, numeric values must be supplied for each design dimension of the multiple master font. A completely specified multiple master font is referred to as an instance of the multiple master font.

The PostScript language name used for a Multiple Master Type 1 is the name of the instance. If the name contains spaces (such as "MinionMM 366 465 11"), these spaces are replaced with underscores. For example, the base font name here is TektonMM and the 2 dimensions specified have values of 200 and 300:

```
TektonMM_200_300
```

The full font name of the TrueType font and the PostScript Language name may differ by spacing and punctuation. For example, spaces are not allow in a PostScript Language name, but are common in full font names. The TrueType name table can also contain the PostScript name, which has no spaces.

14.4.3 Central Baseline

This gives the position in the em square of the central baseline. The central baseline is used by ideographic scripts for alignment, just as the bottom baseline is used for Latin, Greek and Cyrillic scripts.

For TrueType GX fonts, this value may be obtained from the [TRUETYPEGX] [p. 269] `bsln` table. Within this table, the `ideographic centered baseline` may be used for stretches of predominantly ideographic characters and the `ideographic low baseline` is more suitable for ideographic characters in a run of predominantly Latin, Greek or Cyrillic characters.

14.4.4 Co-ordinate units on the em square

Certain values, such as width metrics, are expressed in units that are relative to an abstract square whose height is the intended distance between lines of type in the same type size. This square is called the EM square. The value of this descriptor specifies how many units the EM square is divided into. The valid range is 16 to 16384 units per EM square. Common values are 250 (Intellifont), 1000 (Type 1) and 2048 (TrueType).

If this value is not specified, it becomes impossible to know what any font metrics mean. For example, one font has lowercase glyph representations of height 450; another has smaller ones of height 890! The numbers are actually fractions; the first font has 450/1000 and the second has 890/2048 which is indeed smaller.

For Type 1 fonts, this value may be obtained from the `FontMatrix` entry in the font dictionary. For TrueType fonts, this value may be obtained from the `unitsPerEm` entry in the `head` table. For Intellifont fonts, this value is contained in the font attribute file.

14.4.5 Font encoding tables

Either explicitly or implicitly, each font has a table associated with it, the *font encoding table* , that tells for each glyph what character it is a representation for. In "Type 1 fonts", the table is referred to as an *encoding vector*.

In fact, many fonts contain several glyphs for the same character. Which of those glyphs should be used depends either on the rules of the language, or on the preference of the designer.

In Arabic, for example, all letters have four (or two) different shapes, depending on whether the letter is used at the start of a word, in the middle, at the end, or in isolation. It is the same character in all cases, and thus there is only one character in the HTML document, but when printed, it looks differently each time.

There are also fonts that leave it to the graphic designer to choose from among various alternative shapes provided. Unfortunately, CSS2 doesn’t yet provide the means to select those alternatives. Currently, it is always the default shape that is chosen from such fonts.

14.4.6 Font family name

Specifies the family name portion of the font face name. For example, the family name for Helvetica-Bold is Helvetica and the family name of ITC Stone Serif Semibold Italic is ITC Stone Serif. Some systems treat adornments relating to condensed or expanded faces as if they were part of the family name.

For Type 1 fonts, this value may be obtained from the `FamilyName` entry in the FontInfo dictionary. For TrueType and OpenTypefonts, it may be obtained from the `name` table.

14.4.7 Glyph Representation widths

For Type 1 fonts, this value may be obtained from the @@???. For TrueType fonts, the values are in the `hmtx` table.

14.4.8 Horizontal stem width

For Type 1 fonts, this value may be obtained from the `StdHW` entry, in the Private dictionary or the AFM file.

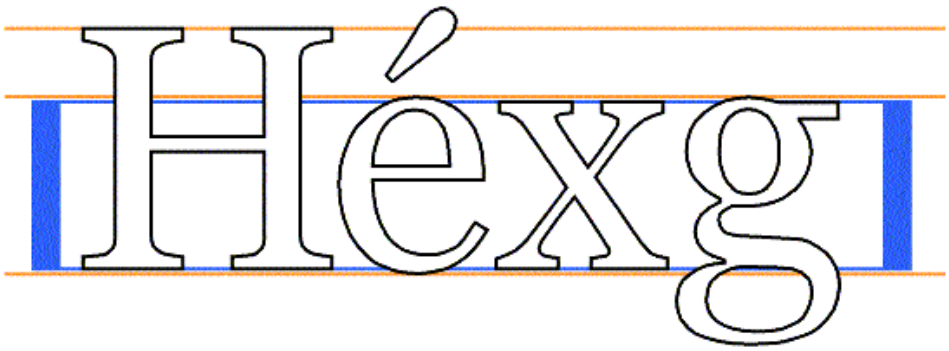
14.4.9 Height of capital glyph representations

The y-coordinate of the top of flat capital letters in Latin, Greek and Cyrillic scripts, measured from the baseline. This descriptor is not useful for fonts that do not contain any glyph representations from these scripts.

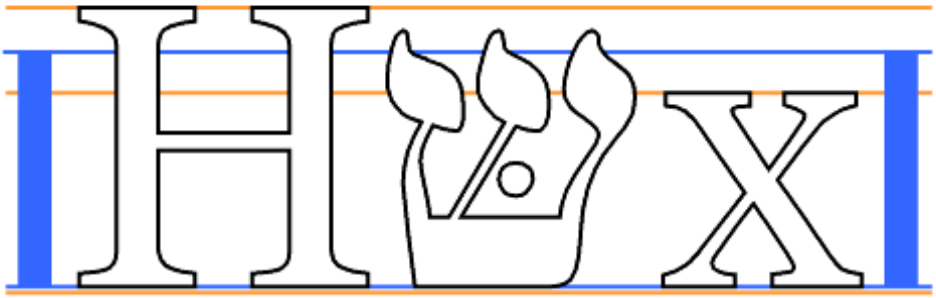
For Type 1 fonts, this value may be obtained from the `CapHeight` entry in the AFM file or from the `Bluevalues` entry in the Private dictionary

14.4.10 Height of lowercase glyph representations

The y-coordinate of the top of unaccented, non-ascending lowercase letters in Latin, Greek and Cyrillic scripts, measured from the baseline. Flat-topped letters are used, ignoring any optical correction zone. Usually used as a ratio of lowercase to uppercase heights, as a means of comparison between font families. The terms large-eye, small-eye are also used to indicate the height of lowercase glyph representations relative to the height of uppercase.



This descriptor is not useful for fonts that do not contain any glyph representations from these scripts. Since the heights of lowercase and uppercase letters are often formed into a ratio for comparing different fonts, it may be useful to set both the lowercase and uppercase heights to the same value for unicameral scripts such as Hebrew, where for mixed Latin and Hebrew text the Hebrew characters are typically set at a height midway between the capital and lowercase heights of the Latin font.



For Type 1 fonts, this value may be obtained from the `Bluevalues` entry in the Private dictionary.

14.4.11 Lower Baseline

This gives the position in the em square of the lower baseline. The lower baseline is used by Latin, Greek and Cyrillic scripts for alignment, just as the upper baseline is used for Sanscrit-derived scripts.

14.4.12 Mathematical Baseline

This gives the position in the em square of the mathematical baseline. The mathematical baseline is used by ideographic scripts for alignment, just as the lower baseline is used for Latin, Greek and Cyrillic scripts.

For TrueType GX fonts, this value may be obtained from the [TRUETYPEGX] [p. 269] `bsln` table.

14.4.13 Maximal bounding box

For Type 1 fonts, this value may be obtained from the FontBBox entry in the font dictionary. For TrueType fonts, the four values are in the `'xMin'`, `'xMax'`, `'yMin'` and `'yMax'` entries of the `'head'` table.

14.4.14 Maximum unaccented height

For Type 1 fonts, this value may be obtained from the `'Ascender'` value in the AFM file. For TrueType and OpenType fonts, this value may be obtained from the `'Ascender'` entry in the [OPENTYPE] [p. 269] `'hhea'` table or (preferably) from the `'sTypoAscender'` value in the [OPENTYPE] [p. 269] `'OS/2'` table.

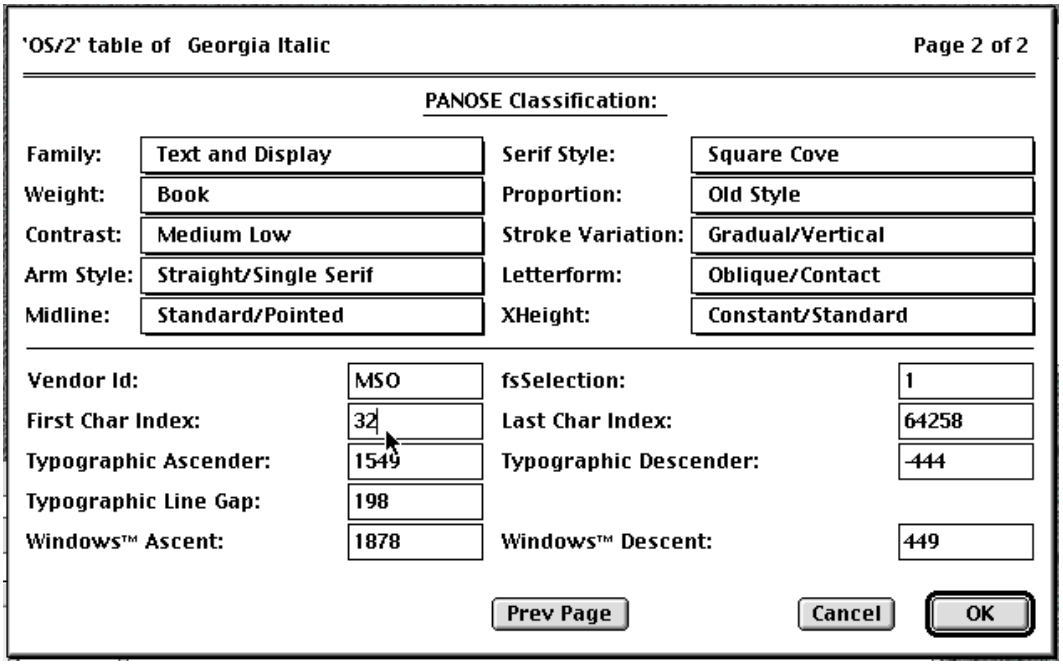
For TrueType GX fonts, the `'horizontalBefore'` entry in the [TRUETYPEGX] [p. 269] `'fmtx'` table is used, overriding Ascender values in the `'hhea'` table.

14.4.15 Maximum unaccented depth

For Type 1 fonts, this value may be obtained from `'descender'` value in the AFM file.

14.4.16 Panose-1 number

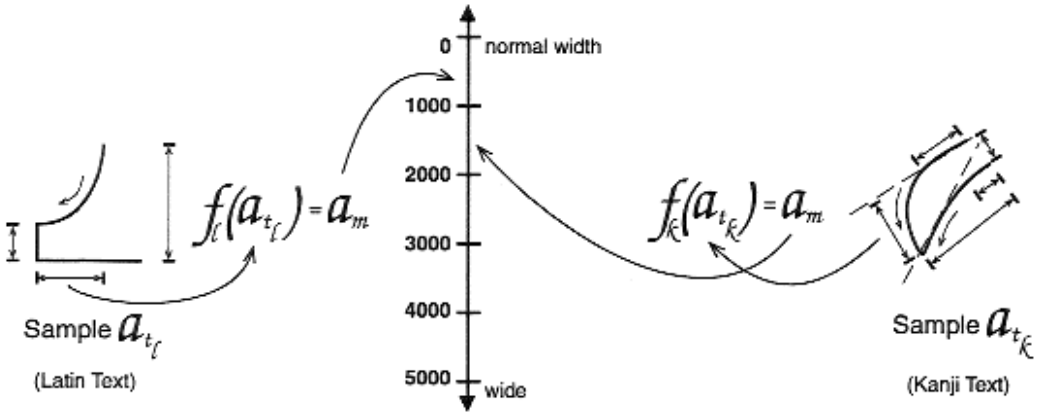
Panose-1 is an industry standard TrueType font classification and matching technology. The PANOSE system consists of a set of ten numbers that categorize the key attributes of a Latin typeface, a classification procedure for creating those numbers, and Mapper software that determines the closest possible font match given a set of typefaces. The system *could*, with modification, also be used for Greek and Cyrillic, but is not suitable for unicameral and ideographic scripts (Hebrew, Armenian, Arabic, Chinese/Japanese/Korean). Panose-1 technology was originally developed by Elseware Corporation and is now owned by Hewlett Packard.



The Family, Serif Style and Proportion numbers are used by Windows95 for font selection and matching.

The meaning of the ten numbers and the allowable values (given in parentheses) are given in Appendix C [p. 256] for the most common case, where the "family" digit is `2, Text and Display`. (If the first digit has a different value, the remaining nine digits have different meanings).

Panose-2 (see [PANOSE2] [p. 269]) is a specification for a more comprehensive font classification and matching technology which is not limited to Latin typefaces. For example, the serif characteristics of a Latin face may be compared with the stroke terminations of a Kanji face.



The Panose-2 value is not stored inside any known font formats, but may be measured.

14.4.17 Range of ISO10646characters

This indicated the glyph repertoire of the font, relative to the Basic Multilingual Plane of [ISO10646] [p. 267] , and is used to eliminate unsuitable fonts (ones that will not have the required glyphs). It does not indicate that the font definitely has the required glyphs, only that it is worth downloading and looking at the font. See

[ISO10646] [p. 267] for information about useful documents.

Font formats that do not include this information, explicitly or indirectly, may still use this descriptor, but the value must be supplied by the document or style sheet author, perhaps being obtained by inspection.

For Type 1 fonts, this value may be obtained from the CMap file).

For TrueType and Opentype fonts with an `OS/2` table, see Appendix C [p. 259] .

There are other classifications into scripts, such as the [MONOTYPE] [p. 269] system and a proposed ISO script system.

Because of this, classification of glyph repertoires by the range of [ISO10646] [p. 267] characters that may be represented with a particular font is suggested in this specification.

14.4.18 Top Baseline

This gives the position in the em square of the top baseline. The top baseline is used by Sanscrit-derived scripts for alignment, just as the bottom baseline is used for Latin, Greek and Cyrillic scripts.

For TrueType GX fonts, this value may be obtained from the [TRUETYPEGX] [p. 269] `bsln` table.

14.4.19 Vertical stem width

The width of vertical (or near-vertical) stems of glyph representations. This information is often tied to hinting, and may not be directly accessible in some font formats. For Type 1 fonts, this may be obtained from the `/StdVW` entry in the Private dictionary or the AFM file. For TrueType fonts, this may be obtained from the `cvt` table.

14.4.20 Vertical stroke angle

Angle, in degrees counterclockwise from the vertical, of the dominant vertical strokes of the font. The value is negative for fonts that slope to the right, as almost all italic fonts do. This descriptor may also be specified for oblique fonts, slanted fonts, script fonts, and in general for any font whose vertical strokes are not precisely vertical. A non-zero value does not of itself indicate an italic font.

14.5 Font matching algorithm

This specification extends the algorithm given in the CSS1 specification. This algorithm reduces down to the algorithm in the CSS1 specification when the author and reader style sheets do not contain any @font-face rules.

Matching of descriptors to font faces must be done carefully. The descriptors are matched in a well-defined order to insure that the results of this matching process are as consistent as possible across UAs (assuming that the same library of font faces and font descriptions is presented to each of them). This algorithm may be optimized, provided that an implementation behaves as if the algorithm had been followed exactly.

1. The user agent makes (or accesses) a database of relevant font-face descriptors of all the fonts of which the UA is aware. If there are two fonts with exactly the same descriptors, one of them is skipped. [p. 31] The UA may be aware of a font because:
    - it has been installed locally
    - it is declared using an @font-face rule in one of the style sheets linked to or contained in the current document
    - it is used in the UA default style sheet, which conceptually exists in all UAs and is considered to have full @font-face rules for all fonts which the UA will use for default presentation, plus @font-face rules for the five special generic font families (see 'font-family') defined in CSS2
2. At a given element and for each character in that element, the UA assembles the font properties applicable to that element. Using the complete set of properties, the UA uses the 'font-family' descriptor to choose a tentative font family. Thus, matching on a family name will succeed before matching on some other descriptor. The remaining properties are tested against the family according to the matching criteria described with each descriptor. If there are matches for all the remaining properties, then that is the matching font face for the given element.
3. If there is no matching font face within the 'font-family' being processed by step 2, *UAs which implement intelligent matching* may proceed to examine other descriptors such as x-height, glyph representation widths, and panose-1 to identify a different tentative font family. If there are matches for all the remaining descriptors, then that is the matching font face for the given element. The font-family descriptor which is reflected into the CSS2 properties is the font family that was requested, not whatever name the intelligently matched font may have. UAs which do not implement intelligent matching are considered to fail at this step.
4. If there is no matching font face within the 'font-family' being processed by step 3, *UAs which implement font downloading* may proceed to examine the src descriptor of the tentative font face identified in step 3 or 4 to identify a network resource which is available, and of the correct format. If there are matches for all the remaining descriptors, then that is the matching font face for the given element and the UA may attempt to download this font resource. The UA may choose to block on this download or may choose to proceed to the next step while the font downloads. UAs which do not implement font download, or are not connected to a network, or where the user preferences have disabled font download, or where the requested resource is unavailable for whatever reason, or where the downloaded font cannot be used for whatever reason, are considered to fail at this step.
5. If there is no matching font face within the 'font-family' being processed by step 3, *UAs which implement font synthesis* may proceed to examine other descriptors such as x-height, glyph representation widths, and panose-1 to identify a different tentative font family for synthesis. If there are matches for all the remaining descriptors, then that is the matching font face for the given element and synthesis of the faux font may begin. UAs which do not implement font synthesis are considered to fail at this step.
6. If all of steps 3, 4 and 5 fail, and if there is a next alternative 'font-family' in the font set, then repeat from step 2 with the next alternative 'font-family'.
7. If there is a matching font face, but it doesn't contain a glyph representation

for the current character, and if there is a next alternative 'font-family' in the font sets, then repeat from step 2 with the next alternative 'font-family'. The 'unicode-range' descriptor may be used to rapidly eliminate from consideration those font faces which do not have the correct glyph representations. If the 'unicode-range' descriptor indicates that a font contains some glyph representations in the correct range, it may be examined by the UA to see if it has that particular one.

8. If there is no font within the family selected in 2, then use a UA-dependent default 'font-family' and repeat from step 2, using the best match that can be obtained within the default font. If a particular character cannot be displayed using the default font, the UA should indicate that a character is not being displayed (for example, using the 'missing character' glyph).
9. UAs which implement progressive rendering and have pending font downloads may, once download is successful, use the downloaded font as a font family. If the downloaded font is missing some glyph representations that the temporary progressive font did contain, the downloaded font is not used for that character and the temporary font continues to be used.

Note. *The above algorithm can be optimized to avoid having to revisit the CSS2 properties for each character.*

The per-descriptor matching rules from (2) above are as follows:

1. 'font-style' is tried first. 'italic' will be satisfied if there is either a face in the UA's font database labeled with the CSS keyword 'italic' (preferred) or 'oblique'. Otherwise the values must be matched exactly or font-style will fail.
2. 'font-variant' is tried next. 'normal' matches a font not labeled as 'small-caps'; 'small-caps' matches (1) a font labeled as 'small-caps', (2) a font in which the small caps are synthesized, or (3) a font where all lowercase letters are replaced by upper case letters. A small-caps font may be synthesized by electronically scaling uppercase letters from a normal font.
3. 'font-weight' is matched next, it will never fail. (See 'font-weight' below.)
4. 'font-size' must be matched within a UA-dependent margin of tolerance. (Typically, sizes for scalable fonts are rounded to the nearest whole pixel, while the tolerance for bitmapped fonts could be as large as 20%.) Further computations, e.g. by 'em' values in other properties, are based on the 'font-size' value that is used, not the one that is specified.

14.5.1 Examples of font matching

The following example defines a specific font face, Alabama Italic. A panose font description and source URI for retrieving a truetype server font are also provided. Font-weight, and font-style descriptors are provided to describe the font. The declaration says that the weight will also match any request in the range 300 to 500). The font family is Alabama and the adorned font name is Alabama Italic.

```
@font-face {
  src: local(Alabama Italic),
       url(http://www.fonts.org/A/alabama-italic) format(truetype);
  panose-1: 2 4 5 2 5 4 5 9 3 3;
  font-family: Alabama, serif;
  font-weight:   300, 400, 500;
  font-style:  italic, oblique;
}
```

The next example defines a family of fonts. A single URI is provided for retrieving the font data. This data file will contain multiple styles and weights of the named font. Once one of these @font-face definitions has been dereferenced, the data will be in the UA cache for other faces that use the same URI.

```
@font-face {
  src: local(Helvetica Medium),
       url(http://www.fonts.org/sans/Helvetica_family) format(truedoc);
  font-family: "Helvetica";
  font-style: normal
}
@font-face {
  src: local(Helvetica Oblique),
       url(http://www.fonts.org/sans/Helvetica_family) format(truedoc);
  font-family: "Helvetica";
  font-style: oblique;
  slope: -18
}
```

The following example groups three physical fonts into one virtual font with extended coverage. In each case, the adorned font name is given in the src descriptor to allow locally installed versions to be preferentially used if available. A fourth rule points to a font with the same coverage, but contained in a single resource.

```
@font-face {
font-family: Excelsior;
src: local(Excelsior Roman), url(http://site/er) format(intellifont);
unicode-range: U+?? /* Latin-1 */
}
@font-face {
font-family: Excelsior;
src: local(Excelsior EastA Roman), url(http://site/ear) format(intellifont);
unicode-range: U+100-220 /* Latin Extended A and B */
}
@font-face {
font-family: Excelsior;
src: local(Excelsior Cyrillic Upright), url(http://site/ecr) format(intellifont);
unicode-range: U+4?? /* Cyrillic */
}
@font-face {
font-family: Excelsior;
src: url(http://site/excels) format(truedoc);
unicode-range: U+??,U+100-220,U+4??;
}
```

This next example might be found in a UA’s default style sheet. It implements the CSS2 generic font family, `serif` by mapping it to a wide variety of serif fonts that might exist on various platforms. No metrics are given since these vary between the possible alternatives.

```
@font-face {
  src: local(Palatino),
          local(Times New Roman),
          local(New York),
          local(Utopia),
          url(http://somewhere/free/font);
  font-family: serif;
  font-weight: 100, 200, 300, 400, 500;
  font-style: normal;
  font-variant: normal;
  font-size: all
}
```

15 Text

Contents

1. Indentation [p. 185] : the ’text-indent’ property
2. Alignment [p. 186] the ’text-align’ property
3. Decoration [p. 186]
    1. Underlining, over lining, striking, and blinking [p. 186] : the ’text-decoration’ property
    2. Text shadows [p. 187] : the ’text-shadow’ property
4. Letter and word spacing [p. 189] : the ’letter-spacing’ and ’word-spacing’ properties
5. Case [p. 190]
    1. Capitalization [p. 190] : the ’text-transform’ property
    2. Special first letter/first line [p. 191]
6. White space [p. 191] : the ’white-space’ property
7. Text in HTML [p. 192]
    1. Forcing a line break [p. 192]

The properties defined in the following sections affect the visual presentation of characters, spaces, words, and paragraphs.

15.1 Indentation: the ’text-indent’ property

’text-indent’

Property name:	’text-indent’
Value:	<length> | <percentage> | inherit
Initial:	0
Applies to:	block-level elements
Inherited:	yes
Percentage values:	refer to parent element’s width
Media groups:	visual [p. 65]

The property, which only applies to block-level [p. 69] elements that contain inline [p. 69] elements, specifies the indentation of the first line of text in the block. More precisely, it specifies the indentation of the first box that flows into the block’s first line box [p. 79] . The box is indented with respect to the left (or right, for right-to-left layout) edge of the line box. User agents should render this indentation as blank space.

Values have the following meanings:

<length>
: The indentation is a fixed length.

<percentage>
: The indentation is a percentage of the containing block width.

The value of ’text-indent’ may be negative, but there may be implementation-specific limits. A line is not indented if the previous line was broken explicitly (e.g., the BR element in HTML).

The following example causes a 3em text indent.

```
P { text-indent: 3em }
```

15.2 Alignment the ’text-align’ property

’text-align’

Property name:	’text-align’
Value:	left | right | center | justify | inherit
Initial:	depends on user agent
Applies to:	block-level elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

This property describes how a paragraph of text is aligned. More precisely, it specifies how boxes in each line box [p. 79] of a block align width respect to the line box. (Note that alignment is not with respect to the viewport [p. 68] but the current containing block [p. 67] .)

In this example, note that since ’text-align’ inherits, all block-level elements inside the DIV element with ’class=center’ will be centered.

```
DIV.center { text-align: center }
```

Note. *The actual justification algorithm used is user-agent and human-language dependent.*

Conforming HTML user agents [p. 27] may interpret the value ’justify’ as ’left’ or ’right’, depending on whether the element’s default writing direction is left-to-right or right-to-left, respectively.

15.3 Decoration

15.3.1 Underlining, over lining, striking, and blinking: the ’text-decoration’ property

’text-decoration’

Property name:	’text-decoration’
Value:	none | [underline || overline || line-through || blink] | inherit
Initial:	none
Applies to:	all elements
Inherited:	no (see prose)
Percentage values:	N/A
Media groups:	visual [p. 65]

This property describes decorations that are added to the text of an element. If the property is specified for a block-level [p. 69] element, it affects all inline children. If it is specified for (or affects) an inline [p. 69] element, it affects all boxes generated by the element. If the element has no content or no text content (e.g., the IMG element in HTML), user agents must skip [p. 31] this property.

Values have the following meanings:

none
: Produces no text decoration.

underline
: Each line of text is underlined.

overline
: Each line of text has a line above it.

line-through
: Each line of text has a line through the middle

blink
: Text blinks (alternates between visible and invisible). Conforming user agents [p. 27] are not required to support this value.

The color(s) required for the text decoration should be derived from the 'color' property value.

This property is not inherited, but descendant boxes of a block-level box should be rendered with the same decoration (e.g., they should all be underlined). The color of decorations should remain the same even if descendant elements have different 'color' values.

In the following example for HTML, the text content of all A elements acting as hyperlinks will be underlined:

```
A:link, A:visited, A:active { text-decoration: underline }
```

15.3.2 Text shadows: the 'text-shadow' property

'text-shadow'

Property name:	'text-shadow'
Value:	none | [<color> || <length> <length> <length>? ,]* [<color> || <length> <length> <length>?] | inherit
Initial:	none
Applies to:	all
Inherited:	no (see prose)
Percentage values:	N/A
Media groups:	visual [p. 65]

The 'text-shadow' property accepts a comma-separated list of shadow effects to be applied to the text of the element. The shadow effects are applied in the order specified and may thus overlay each other, but they will never overlay the text itself. Shadow effects do not alter the size of the element, but may extend beyond its boundaries. The Z-order [p. 96] of the shadow effects is the same as for the element itself.

Each shadow effect must specify a shadow offset and may optionally specify a blur radius and a shadow color.

A shadow offset is specified with two <length> values that indicate the distance from the text. The first length value specifies the horizontal distance to the right of the text. A negative horizontal length value places the shadow to the left of the text. The second length value specifies the vertical distance below the text. A negative vertical length value places the shadow above the text.

A blur radius may optionally be specified after the shadow offset. The blur radius is a length value that indicates the boundaries of the blur effect. The exact algorithm for computing the blur effect is not specified.

A color value may optionally be specified before or after the length values of the shadow effect. The color value will be used as the basis for the shadow effect. If no color is specified, the value of the 'color' property will be used instead.

The example below will set a text shadow to the right and below the element's text. Since no color has been specified, the shadow will have the same color as the element itself, and since no blur radius is specified, the text shadow will not be blurred:

```
H1 { text-shadow: 0.2em 0.2em }
```

The next example will place a shadow to the right and below the element's text. The shadow will have a 5px blur radius and will be red.

```
H2 { text-shadow: 3px 3px 5px red }
```

The next example specifies a list of shadow effects. The first shadow will be to the right and below the element's text and will be red with no blurring. The second shadow will overlay the first shadow effect, and it will be yellow, blurred, and placed to the left and below the text. The third shadow effect will be placed to the right and above the text. Since no shadow color is specified for the third shadow effect, the value of the element's 'color' property will be used:

```
H2 { text-shadow: 3px 3px red, yellow -3px 3px 2px, 3px -3px }
```

Consider this example:

```
SPAN.glow {
  background: white;
  color: white;
  text-shadow: black 0px 0px 5px;
}
```

Here, the 'background' and 'color' properties have the same value and the 'text-shadow' property is used to create a "solar eclipse" effect:

Note. *This property is not defined in CSS1. Some shadow effects may render text invisible in UAs that only support CSS1.*

15.4 Letter and word spacing: the ’letter-spacing’ and ’word-spacing’ properties

’letter-spacing’

Property name:	’letter-spacing’
Value:	normal | <length> | auto | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

This property specifies spacing behavior between text characters. Values have the following meanings:

normal
: This value allows the user agent to alter the space between characters in order to justify text.

<length>
: This value indicates inter-character space in addition to the default space between characters. Values may be negative, but there may be implementation-specific limits.

auto
: This value tells the user agent to adjust the spacing between characters so that the entire text of an element fits on one line. This value should only be used with special elements (e.g., headlines). See also the ’font-size’ property for related ’auto’ behavior.

Character spacing algorithms are user agent-dependent. Character spacing may also be influenced by justification (see the ’text-align’ property).

In this example, the space between characters in BLOCKQUOTE elements is increased by ’0.1em’.

```
BLOCKQUOTE { letter-spacing: 0.1em }
```

This will not happen if ’letter-spacing’ is explicitly set to a <length> value, as in:

```
BLOCKQUOTE { letter-spacing: 0 }
BLOCKQUOTE { letter-spacing: 0cm }
```

When the resultant space between two characters is not the same as the default space, user agents should not use ligatures

Conforming HTML user agents [p. 27] may consider the value of the ’letter-spacing’ property to be ’normal’.

’word-spacing’

Property name: 'word-spacing'
Value: normal | <length> | inherit
Initial: normal
Applies to: all elements
Inherited: yes
Percentage values: N/A
Media groups: visual [p. 65]

This property specifies spacing behavior between words. Values have the following meanings:

normal
This value allows the user agent to alter the space between words in order to justify text.

<length>
This value indicates inter-word space in addition to the default space between words. Values may be negative, but there may be implementation-specific limits.

Word spacing algorithms are user agent-dependent. Word spacing may also be influenced by justification (see the 'text-align' property).

In this example, the word-spacing between each word in H1 elements is increased by '1em'.

```
H1 { word-spacing: 1em }
```

Conforming HTML user agents [p. 27] may consider the value of the 'word-spacing' property to be 'normal'.

15.5 Case

15.5.1 Capitalization: the 'text-transform' property

'text-transform'

Property name: 'text-transform'
Value: capitalize | uppercase | lowercase | none | inherit
Initial: none
Applies to: all elements
Inherited: yes
Percentage values: N/A
Media groups: visual [p. 65]

This property controls capitalization effects of an element's text. Values have the following meanings:

capitalize
Puts the first character of each word in uppercase.

uppercase
Puts all characters of each word in uppercase.

lowercase
Puts all characters of each word in lowercase.
none
No capitalization effects.

The actual transformation in each case is human language- dependent. See [RFC2070] [p. 267] for ways to find the language of an element.

Conforming HTML user agents [p. 27] may consider the value of 'text-transform' to be 'none' for characters that are not from the Latin-1 repertoire and for elements in languages for which the transformation is different from that specified by the case-conversion tables of [UNICODE] [p. 268] .

In this example, all text in an H1 element is transformed to uppercase text.

```
H1 { text-transform: uppercase }
```

15.5.2 Special first letter/first line

Please consult the sections on first line [p. 52] and first letter [p. 52] for information on specially formatting the first letter or line of a paragraph.

15.6 White space: the 'white-space' property

'white-space'

Property name:	'white-space'
Value:	normal | pre | nowrap | inherit
Initial:	normal
Applies to:	block-level elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

This property declares how whitespace [p. 31] inside the element is handled. Values have the following meanings:

normal
This value allows user agents to collapse sequences of white space. Line breaks may be created by characters or elements (e.g., the BR element in HTML).
pre
This value prevents user agents from collapsing sequences of white space.
nowrap
This value suppresses line breaks within text except for those created by other elements (e.g., the BR element in HTML).

The following examples show what whitespace [p. 31] behavior is expected from the PRE and P elements in HTML.

```
PRE { white-space: pre }
P   { white-space: normal }
```

Conforming HTML user agents [p. 27] may skip [p. 31] the 'white-space' property in author and user style sheets but must specify a value for it in the default style sheet.

15.7 Text in HTML

15.7.1 Forcing a line break

The current CSS2 properties and values cannot describe the behavior of the BR element; the BR element specifies a line break between words. In effect, the element is replaced by a line break. Future versions of CSS may handle added and replaced content, but CSS2-based formatters must treat BR specially.

16 Lists

Contents

1. Visual formatting of lists [p. 193]
    1. List properties [p. 195] : 'list-style-type', 'list-style-image', 'list-style-position', and 'list-style'

16.1 Visual formatting of lists

CSS allows authors to control the visual presentation of lists in a number of ways:

- Authors may specify a marker that appears before each list item.
- Markers may be placed outside or inside the list item's content.
- Markers may be represented by predefined shapes (bullets, circles, squares), numerals (arabic, roman, letters, etc.), or images.
- With descendant selectors [p. 45] and child selectors [p. 46] , it's possible to specify different marker types depending on the depth of embedded lists.

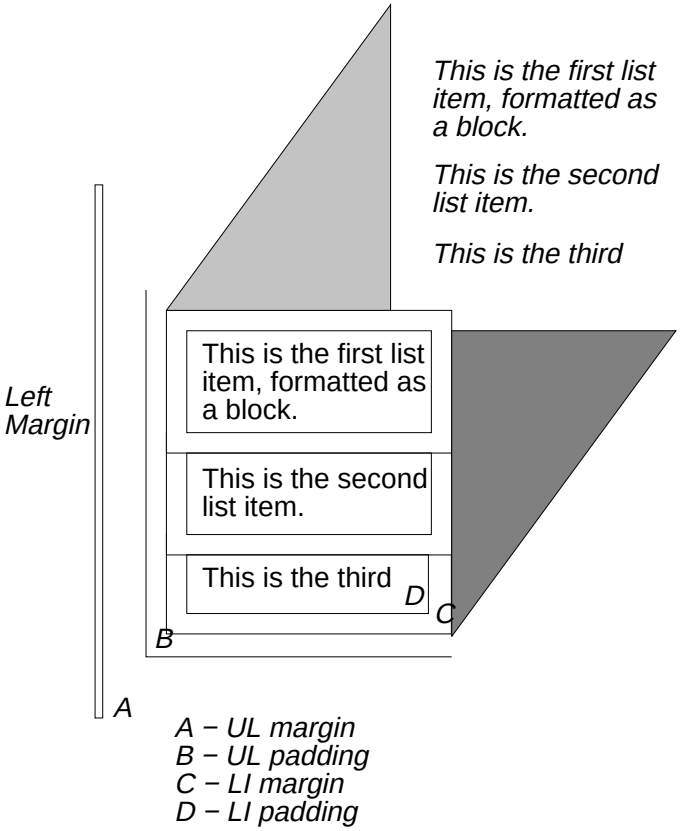
An element with a 'display' property value of 'list-item' generates two boxes: a block-level [p. 69] box that contains its content and an inline box that contains a marker. Only the content box is involved with positioning (e.g, in the normal flow [p. 79]). The position of the marker box (see the 'list-style-position' property) does not affect the position of the content box. CSS properties allow authors to specify the marker type (image, glyph, or number) and its position with respect to the content box (outside the content or at the beginning of content).

The background properties [p. 142] apply to the content box only; the marker box is transparent.

The declaration in the following HTML program causes LI elements to generated 'list-item' boxes. The 'list-style' value 'none' suppresses preceding markers.

```
<HTML>
<HEAD>
   <STYLE type="text/css">
      LI { display: list-item; list-style: none }
   </STYLE>
</HEAD>
<BODY>
   <UL>
      <LI> This is the first list item, formatted as a block.
      <LI> This is the second list item.
      <LI> This is the third.
   </UL>
</BODY>
</HTML>
```

The list might be formatted as follows:

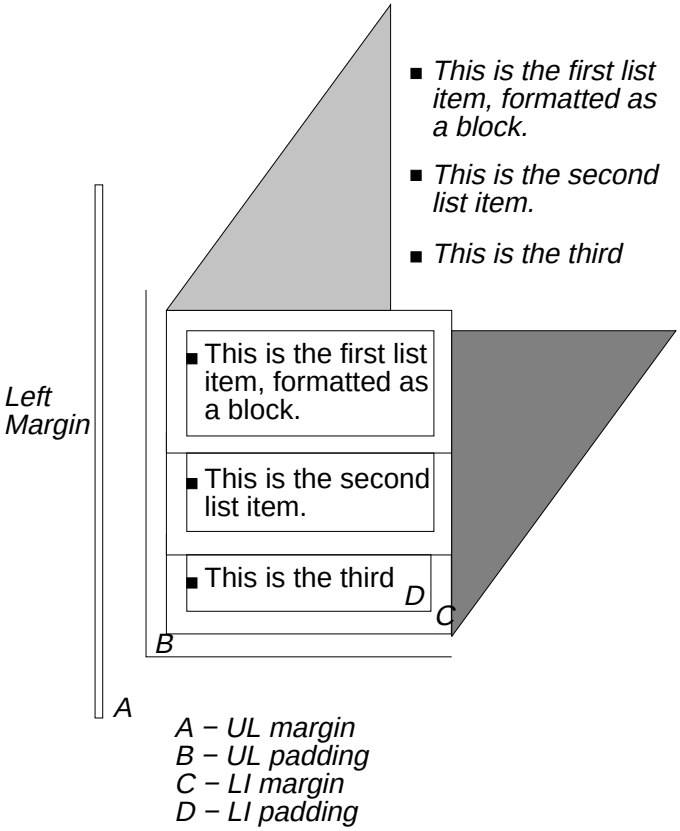


The illustration shows the relationship between the current left margin and the margins and padding of the list (UL) box and the list item (LI) boxes. (The lines delimiting the margins and padding are not rendered).

If we change the 'list-style' to "square":

```
LI { display: list-item; list-style: square }
```

each list item will be preceded by a small square. However, the placement of the square does not affect the block formatting [p. 79] of the list item box:



Note.

- *CSS2 does not include a property to adjust the separation between a list marker and the content of its list item.*
- *There is no "list" presentation for other types of list structures (e.g., "definition lists" declared by DL, DT, and DD in HTML). Each part of a definition list is simply a block element.*

16.1.1 List properties: 'list-style-type', 'list-style-image', 'list-style-position', and 'list-style'

'list-style-type'

Property name:	'list-style-type'
Value:	disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none | inherit
Initial:	disc
Applies to:	elements with the 'display' property set to 'list-item'
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

This property specifies appearance of the list item marker if 'list-style-image' has the value 'none' or if the image pointed to by the URI cannot be displayed.

Values have the following meanings:

disc
: A disc (exact presentation is UA-dependent)

circle
: A circle (exact presentation is UA-dependent)

square
: A square (exact presentation is UA-dependent)

decimal
: Decimal numbers, beginning with 1.

lower-roman
: Lower case roman numerals (i, ii, iii, iv, v, etc.)

upper-roman
: Upper case roman numerals (I, II, III, IV, V, etc.)

lower-alpha
: Lower case ascii letters (a, b, c, ... z)

upper-alpha
: Upper case ascii letters (A, B, C, ... Z)

none
: No marker

For example, the following HTML document:

```
<HTML>
   <HEAD>
   <STYLE type="text/css">
        OL { list-style-type: lower-roman }
   </STYLE>
</HEAD>
<BODY>
   <OL>
      <LI> This is the first item.
      <LI> This is the second item.
      <LI> This is the third item.
   </OL>
</BODY>
</HTML>
```

might produce something like this:

```
  i This is the first item.
 ii This is the second item.
iii This is the third item.
```

'list-style-image'

Property name:	'list-style-image'
Value:	<uri> | none | inherit
Initial:	none
Applies to:	elements with the 'display' property set to 'list-item'
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

This property sets the image that will be used as the list item marker. When the image is available, it will replace the marker set with the ’list-style-type’ marker.

The following example sets the marker at the beginning of each list item to be the image "ellipse.png".

```
UL { list-style-image: url(http://png.com/ellipse.png) }
```

’list-style-position’

Property name:	’list-style-position’
Value:	inside | outside | inherit
Initial:	outside
Applies to:	elements with the ’display’ property set to ’list-item’
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

This property specifies the position of the marker box with respect to line item content box. Values have the following meanings:

outside
: The list item marker is outside the box that is generated for the list item.

inside
: The list item marker is the first inline box generated for the list item. The list item’s contents flow after the marker box.

For example:

```
<HTML>
<HEAD>
<STYLE type="text/css">
  UL         { list-style: outside }
  UL.compact { list-style: inside }
</STYLE>
</HEAD>
<BODY>
<UL>
  <LI>first list item comes first
  <LI>second list item comes second
</UL>

<UL class="compact">
  <LI>first list item comes first
  <LI>second list item comes second
</UL>
</BODY>
</HTML>
```

The above example may be formatted as:

- first list item comes first
- second list item comes second

- first list item comes first
- second list item comes second

The left sides of the list item boxes are not affected by marker placement

In right-to-left text, the markers would have been on the right side of the box.

'list-style'

Property name:	'list-style'
Value:	[<'list-style-type'> || <'list-style-position'> || <'list-style-image'>] | inherit
Initial:	not defined for shorthand properties
Applies to:	elements with the 'display' property set to 'list-item'
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

The 'list-style' property is a shorthand notation for setting the three properties 'list-style-type', 'list-style-image', and 'list-style-position' at the same place in the style sheet.

```
UL { list-style: upper-roman inside }  /* Any UL*/
UL + UL { list-style: circle outside } /* Any UL child of a UL*/
```

Although authors may specify 'list-style' information directly on list item elements (e.g., LI in HTML), they should do so with care. The following rules look similar, but the first declares a descendant selector [p. 45] and the second a (more specific) child selector. [p. 46]

```
OL.alpha LI  { list-style: lower-alpha } /* Any LI descendant of an OL */
OL.alpha + LI  { list-style: lower-alpha } /* Any LI child of an OL */
```

Authors who only use the descendant selector [p. 45] may not achieve the results they expect. Consider the following rules:

```
<HTML>
<HEAD>
<STYLE type="text/css">
  OL.alpha LI  { list-style: lower-alpha }
  UL LI        { list-style: disc }
</STYLE>
</HEAD>
```

```
<BODY>
  <OL class="alpha">
    <LI>level 1
    <UL>
       <LI>level 2
    </UL>
  </OL>
</BODY>
</HTML>
```

The desired rendering would have level 1 list items with ’lower-alpha’ labels and level 2 items with ’disc’ labels. However, the cascading order [p. 60] will cause the first style rule (which includes specific class information) to mask the second. The following rules solve the problem by employing a child selector [p. 46] instead:

```
OL.alpha + LI  { list-style: lower-alpha }
UL LI   { list-style: disc }
```

Another solution would be to specify ’list-style’ information only on the list type elements:

```
OL.alpha  { list-style: lower-alpha }
UL        { list-style: disc }
```

Inheritance will transfer the ’list-style’ values from OL and UL elements to LI elements. This is the recommended way to specify list style information.

A URI value may be combined with any other value, as in:

```
UL { list-style: url(http://png.com/ellipse.png) disc }
```

In the example above, the ’disc’ will be used when the image is unavailable.

17 Tables

Contents

1. Building visual tables [p. 202]
    1. The ’display’ property [p. 202]
    2. Cell spanning properties: [p. 204] ’column-span’, and ’row-span’
2. Layout model of elements inside tables [p. 204]
    1. Margins on rows, columns and cells [p. 205]
    2. Interactions between rows and columns [p. 205]
        1. Labeled diagram: [p. 206]
    3. The ’border-collapse’ property [p. 207]
    4. The border styles [p. 209]
3. Computing widths and heights [p. 210]
    1. The ’table-layout’ property [p. 210]
    2. The ’collapse’ value for the ’visibility’ property [p. 211]
4. ’Vertical-align’ on table cells [p. 211]
5. Table elements in selectors [p. 212]
    1. Columns and cell selectors [p. 212]
    2. Row, column and cell pseudo-classes [p. 213]
        1. Row pseudo-classes: [p. 213]
        2. Column pseudo-classes: [p. 213]
6. HTML 4.0 default table stylesheet [p. 213]
7. UNDER CONSTRUCTION! [p. 214]
    1. Table layout [p. 214]
    2. Computing widths and heights [p. 216]
    3. Placement of the borders: [p. 216] ’cell-spacing’
    4. Conflict resolution for borders [p. 218]
    5. Properties for columns and rows [p. 220]
    6. Vertical alignment of cells in a row [p. 220]
    7. Horizontal alignment of cells in a column [p. 222]
    8. Table captions [p. 222] : the ’caption-side’ property
    9. Generating speech [p. 222] : the ’speak-header’ property
    10. Table implementation notes [p. 225]

Tables are used to show the relations between pieces of data, by arranging them into visual rows, columns, and cells. CSS2 can create and control the presentation of these visual elements, but is not intended to allow the arrangement of the structure itself.

Most CSS properties apply to table elements in the same manner as they apply to block-level elements. However, because table columns in the HTML model do not contain strictly, some properties behave differently for tables. A few properties apply *only* to tables.

17.1 Building visual tables

A visual table is composed of a single element designated as the "table", which contains any number of table columns, column groups, rows and row groups. This will allow XML data, for example, to be visually presented as a table through CSS, without requiring any other semantics to be attached to the XML element types in the document.

17.1.1 The ’display’ property

There are ten values for the ’display’ property that create tables and parts of tables:

- *’table’* - a ’table’ is the outer container for all tables. It defines a rectangular block element that contains any number of row groups, rows, column groups, columns, and/or captions, all arranged in an irregular grid pattern. Any rendering objects other than these should be displayed outside the rectangular grid. (HTML analog: TABLE)
- *’inline-table’* - an ’inline-table’ is identical to a ’table’ element, except that it is treated as an inline replaced element in the context surrounding the ’inline-table’ element.
- *’table-column-group’* - a ’table-column-group’ is a container for a number of table columns. This allows the designer to set presentation properties on a group of columns, for example by placing borders around a group of columns. (HTML analog: COLGROUP)
- *’table-column’* - a ’table-column’ is a grouping of all cells in a particular vertical column. (HTML analog: COL)
- *’table-row-group’* - a ’table-column-group’ is a container for a number of table rows. This allows the designer to set presentation properties on a group of rows, for example by placing borders around a group of rows. (HTML analog: TBODY)
- *’table-header-group’* - A ’table-header-group’ is a special type of ’table-row-group’ that is always displayed at the top of the table, above all other rows and rowgroups, but below any captions. (HTML analog: THEAD)
- *’table-footer-group’* - A ’table-footer-group’ is a special type of ’table-row-group’ that is always displayed at the bottom of the table, below all other rows and rowgroups, but above any captions. (HTML analog: TFOOT)
- *’table-row’* - a ’table-row’ groups all cells in a particular horizontal row. (HTML analog: TR)
- *’table-cell’* - a ’table-cell’ is the rectangular object to be arranged in the table grid. Table cells are only allowed inside rows. (HTML analog: TD)
- *’table-caption’* - a ’table-caption’ is a special type of table cell that appears as a row or column of its own in a table. (HTML analog: CAPTION) *[Need better description; caption is not really a row or column]*

Elements designated as ’inline-table’ are considered to be replaced elements; the others are to be treated as block-level elements for the purposes of classification of interaction with other CSS properties - that is, other CSS properties that are defined as applying to block-level elements will apply to these

elements. Some of the semantics of these other properties may change, as defined below.

Throughout this specification, "table elements" refers to any element designated as one of these ten types; "'table' elements" refers to only those designated as display type 'table'. Similarly, unless single-quoted, "row groups" refers to elements of type 'table-row-group', 'table-header-group' or 'table-footer-group', "row elements" refers to row group elements or elements of type 'table-row', and 'columns' refers to elements of type 'table-column' or 'table-column-group'.

Tables are row-primary; that is, a branch of a structured document is displayed as a visual table by designating one element as a 'table', with one or more of its child elements designated as 'rows', and one or more elements inside each row designated as 'cells'. 'Row-groups' are optional structures used to group rows, and potentially create specific margin, border or padding space around them. 'Columns' and 'column-groups' are optional structures that allow borders to be placed around specific rows and allow CSS selectors to select cells and their children based on the columns they appear in. The visual interactions of columns and rows are described below in the table layout model.

"Missing" elements are inserted as anonymous elements [p. 79] . In other words: any table element will automatically generate a whole table tree around itself, consisting of at least a 'table'/'inline-table', a 'table-row' and a 'table-cell'.

In particular, if the parent a 'table-cell' is not a 'table-row', an anonymous 'table-row' element will be assumed as its parent, and this anonymous element will span all consecutive 'table-cell' siblings.

Similarly, if the parent of a 'table-row' is not a 'table'/'inline-table' nor a row group element, an anonymous 'table' will be assumed as its parent, spanning all consecutive siblings that also need an anonymous 'table' parent.

Analogously for a row group element.

In this XML example, an anonymous 'table' is inserted to contain the HBOX element:

```
<HBOX>
  <VBOX>George</VBOX>
  <VBOX>4287</VBOX>
  <VBOX>1998</VBOX>
</HBOX>
```

The style sheet is:

```
HBOX {display: table-row}
VBOX {display: table-cell}
```

Anonymous 'table-cell' elements are inserted as children of 'table-rows', to contain consecutive children that are not 'table-cell' elements.

In this example, three anonymous 'table-cell' elements are inserted to contain the text in the ROWs. Note that the text is further encapsulated in anonymous inline boxes, as explained in "Visual rendering model." [p. 79] :

```
<STACK>
  <ROW>This is the <D>top</D> row.</ROW>
  <ROW>This is the <D>middle</D> row.</ROW>
  <ROW>This is the <D>bottom</D> row.</ROW>
</STACK>
```

The style sheet is:

```
STACK {display: inline-table}
ROW {display: table-row}
D {display: inline; font-weight: bolder}
```

Elements with ’display’ set to ’table-column’ or ’table-column-group’ are not rendered (exactly as if they had ’display: none’), but they are useful, because they may have attributes which induce a certain style for the columns they represent.

17.1.2 Cell spanning properties: ’column-span’, and ’row-span’

’row-span’

Property name:	’row-span’
Value:	<integer> | inherit
Initial:	1
Applies to:	cell elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

This property specifies the number of rows spanned by a cell. A cell box occupies a rectangle of ’column-span’ by ’row-span’ grid cells in a table.

’column-span’

Property name:	’column-span’
Value:	<integer> | inherit
Initial:	1
Applies to:	cell, column, and column-group elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

This property specifies the number of grid columns spanned by a cell, column, or column group. A cell box occupies a rectangle of ’column-span’ by ’row-span’ grid cells in a table.

[Need description of how to determine the row/column coordinates of cells and how to count column elements; or refer to HTML 4.0?]

17.2 Layout model of elements inside tables

The layout of tables is a fundamentally different algorithm than that used to lay out regular block-level elements. Cells are laid out on a two-dimensional grid-based pattern, unlike block elements, which are laid on in a one-dimensional flow. This is further complicated by the interactions of rows and columns.

17.2.1 Margins on rows, columns and cells

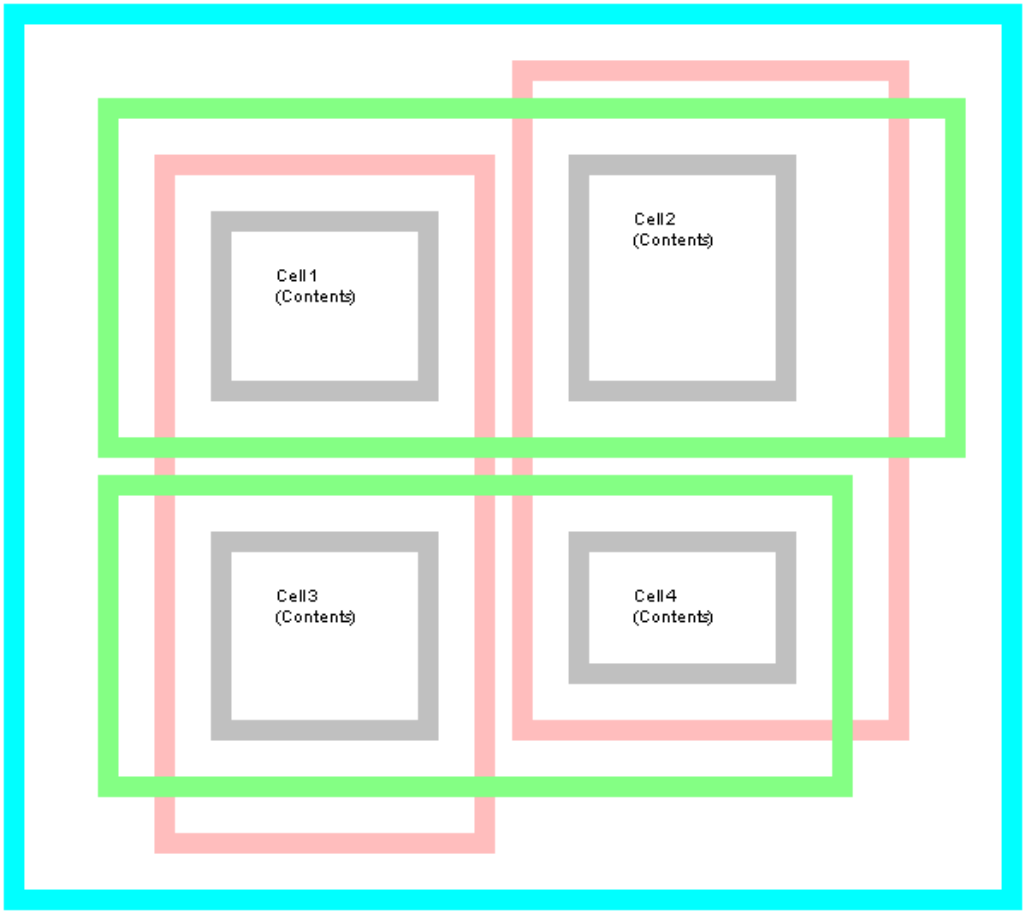
The most important difference between the layout model of tables in CSS and the layout model of other elements (e.g., inline and block-level elements) is that margins collapse horizontally as well as vertically on elements inside tables (margins on the table element itself function as expected on a replaced element). This collapsing uses the same algorithm as vertical collapsing on regular block elements in CSS. This is designed to allow easy duplication of the HTML "cell spacing" effect - that is, to easily design evenly spaced table cells in a table.

[Can this work? Maybe it is better to say that table elements (other than 'table' and 'inline-table') have no margins at all, and instead use a single 'cell-spacing' property for the whole table, or a single horizontal and a single vertical spacing (see below)]

17.2.2 Interactions between rows and columns

Row and Column elements (and row-groups and column-groups) inside tables essentially exist in two different layout planes. They both constrain and are constrained by the layout of the cells they contain, but rows and columns are not considered to interact with each other directly when laying out the table. This means that widths on columns may change the layout of the cell (due to word-breaking occurring with different constraints), which may affect the height of the cell (and hence the row), but the columns do not directly affect the row height (and vice versa).

Example of interaction:



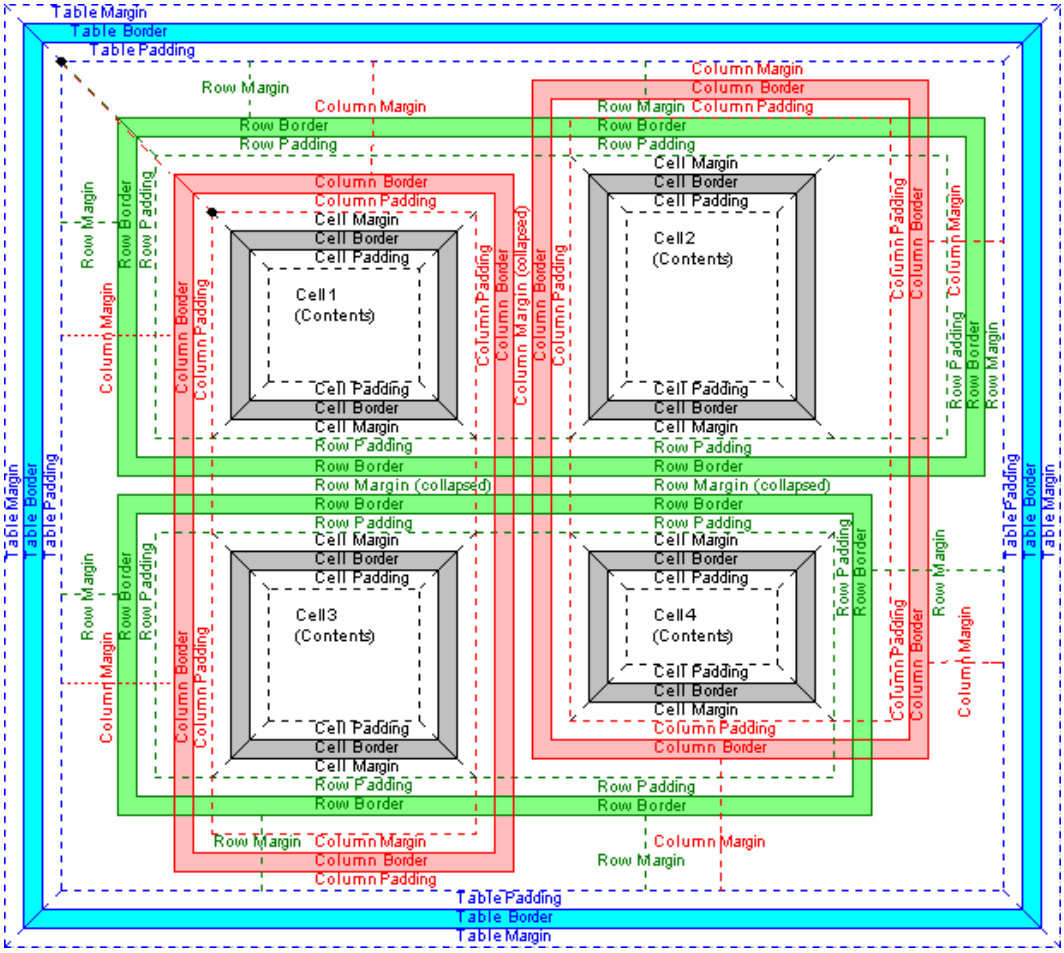
CSS rules:

```
#tbl1 {
        margin: 1em;
        border: 1em solid cyan;
        padding: 1em;
}
```

HTML source:

```
<P>
<TABLE id="tbl1">
<COL id="col1"><COL id="col2">
<TR id="row1">
    <TD id="cell1">Cell1<BR>(Contents)</TD>
    <TD id="cell2">Cell2<BR>(Contents)</TD>
</TR><TR id="row2">
    <TD id="cell3">Cell3<BR>(Contents)</TD>
    <TD id="cell4">Cell4<BR>(Contents)</TD>
</TR>
</TABLE>
```

Labeled diagram:



17.2.3 The ’border-collapse’ property

’border-collapse’

Property name:	’border-collapse’
Value:	collapse | separate
Initial:	separate
Applies to:	’table’ elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

This property selects between two different models of border behavior. The HTML table behavior is that borders are drawn separately for each cell, and cell spacing separates the borders. Another common model in document publishing (and used in most popular word processors) is a collapsing border model. In this model, adjacent borders are collapsed into one border when drawn. If the border is collapsed, any margin or column/row padding separating the two borders is ignored.

The question of which border to draw is determined by the following algorithm. If any border side on an element is set to ’none’, it is considered to have the lowest possible precedence - in other words, any other non-’none’ values that

may apply to that edge from other coincident border sides will apply. (This does not apply to borders with style of ’hidden’, which has highest precedence - see the following section on border style values for more information.)

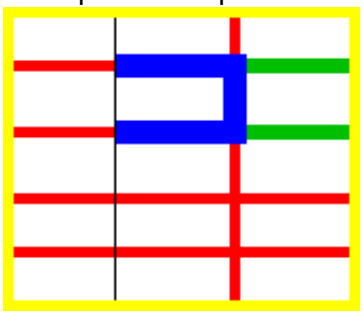
1. Borders on ’table’ elements take precedence over borders on any other table elements.
2. Borders on ’row-groups’ take precedence over borders on ’rows’, and likewise borders on ’column-groups’ take precedence over borders on ’columns’.
3. Borders on any other type of table element take precedence over ’table-cell’ elements.
4. If the adjacent elements are of the same type, the wider of the two borders is used. "Wider" takes into account the border-style of ’none’, so a "1px solid" border will take precedence over a "20px none" border.
5. If the borders are of the same width, the border on the element occurring first is used.

[A possible other rule, that tries to make sure thinner borders never interrupt thicker ones: if *any* of the coincident border is ’hidden’, use that; otherwise, if *all* of the coincident borders are ’none’, use that; otherwise take the widest border among the non-’none’ styles; if there is more than one, prefer ’double’ over ’solid’ over ’dashed’ over ’dotted ’ over ’inset’ over ’outset’ over ’ridge over ’groove’. If there is still a conflict (which must be in the color), the result is undefined.]

The chosen border is drawn using its normal width, style and color. Other coincident border edges are **not** drawn underneath, in the case of partially transparent border styles (e.g. ’double’ borders, which have a transparent gap).

Note. [If the margins around table elements are dropped in favor of a single ’cell-spacing’ property, as suggested above, the cell spacing and ’border-collapse’ can be combined according to ’cell-spacing’]

Example of collapsed borders:



CSS source:

```
TABLE {         border: 5px solid yellow; }
#col1 {         border: 1px solid black; }
TD {            border: 5px solid red; }
TD.foo {        border: 11px solid blue; }
TD.bar {        border: 7px solid green; }
}
```

HTML source:

```
<P>
<TABLE>
<COL id="col1"><COL id="col2"><COL id="col3">
<TR id="row1">
    <TD> </TD>
    <TD> </TD>
    <TD> </TD>
</TR>
<TR id="row2">
    <TD> </TD>
    <TD class="foo"> </TD>
    <TD class="bar"> </TD>
</TR>
<TR id="row3">
    <TD> </TD>
    <TD> </TD>
    <TD> </TD>
</TR>
<TR id="row4">
    <TD> </TD>
    <TD> </TD>
    <TD> </TD>
</TR>
<TR id="row5">
    <TD> </TD>
    <TD> </TD>
    <TD> </TD>
</TR>
</TABLE>
```

17.2.4 The border styles

Some of the border styles are drawn differently in tables than elsewhere (cf. "Border style" [p. 107]):

none
: No border.

hidden
: Same as ’none’, but if borders are drawn collapsed, also inhibits any other border (see above [p. 207]).

dotted
: The border is a series of dots.

dashed
: The border is a series of short line segments.

solid
: The border is a single line segment.

double
: The border is two solid lines. The sum of the two lines and the space between them equals the value of ’border-width’.

groove
: The border looks as though it were carved into the canvas.

ridge
: The opposite of ’grove’: the border looks as though it were coming out of the canvas.

inset
: If borders are drawn non-collapsed: the border makes the entire box look as though it were embedded in the canvas. If borders are collapsed: same as ’groove’.

outset
: The opposite of ’inset’. If borders are drawn non-collapsed: the border makes the entire box look as though it were coming out of the canvas. If borders are collapsed: same as ’ridge’.

Example of hidden collapsed borders:

HTML source:

```
<TABLE style="border-collapse: collapse; border: solid;">
<TR><TD style="border: hidden">foo</TD>
    <TD style="border: solid">bar</TD></TR>
<TR><TD style="border: none">foo</TD>
    <TD style="border: solid">bar</TD></TR>
</TABLE>
```

17.3 Computing widths and heights

17.3.1 The ’table-layout’ property

’table-layout’

Property name:	’table-layout’
Value:	auto | fixed
Initial:	auto
Applies to:	’table’ elements
Inherited:	no
Percentage values:	N/A
Media groups:	visual [p. 65]

The ’table-layout’ property controls the algorithm used to lay out the table (that is, to determine the widths and heights of the table and its individual cells, rows and columns). A value of ’auto’ denotes the existing-practice HTML layout algorithm, which requires multiple passes through the data and determines the width of a column of cells based on the largest unbreakable element in a cell in the column. This algorithm can be inefficient - not only does it require multiple passes, it also requires the layout engine to have access to all the content in the table before determining the final layout.

The ’fixed’ layout algorithm is a much faster algorithm. The horizontal layout of the table is not dependent on the content contained in the cells, it is dependent entirely on the widths set on the table and its columns and first row of cells. If the table has a ’width’ of ’auto’, the entire available space is used. The cell’s width is determined by any width value set on the column, overridden by any value other than ’auto’ set on the cell itself. All cells that have a ’width’ of ’auto’ in the row are collected by the layout engine, and when the end of the row is reached, all

remaining available space is evenly distributed among them.

In this manner, the engine lays out the first row of the table immediately upon receiving the content. All subsequent rows have the same widths of columns of cells - nothing in subsequent rows can affect the widths of the columns. Any cell that has content that overflows its dimensions uses the 'overflow' property to determine how to handle the overflow content. Using the 'overflow' value of 'visible' will not grow the cell dimensions, but will slop the content into the next cell. The height of the row of cells may be computed (the 'auto' value), or may be set explicitly, in which case overflow is handled in the same manner as with the width dimension.

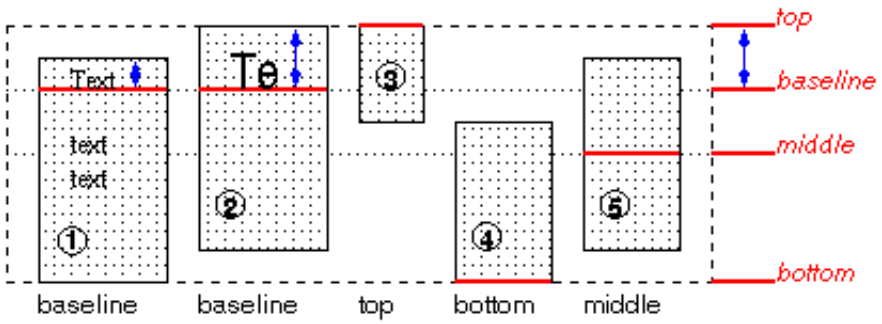
17.3.2 The 'collapse' value for the 'visibility' property

The 'visibility' property should take an additional value -- 'collapse'. This value works identically to the 'hidden' value, except when it is applied to a 'table-row', 'table-row-group', 'table-column' or 'table-column-group'. When applied to an entire table row or column, the 'collapse' property will cause the entire row or column to be removed from the display, and the space normally taken up by the row or column will be available for other content - in this context only, it is similar to setting the 'display' property to 'none', except the cells that are no longer displayed will continue to affect the horizontal layout of the table columns (if a row is being collapsed) or vertical layout of table rows (if a column is being collapsed). This allows dynamic effects to remove table rows or columns without forcing a re-layout of the table in order to account for the potential change in column constraints.

17.4 'Vertical-align' on table cells

The CSS 'vertical-align' property also applies to 'table-cell' elements. Only the 'top', 'middle', 'bottom', 'baseline' and percentage values are applicable - all other values function as 'middle'. The property sets the vertical alignment of the content inside the table cell, in the same manner as the HTML VALIGN attribute on table cells (TD elements) does.

The baseline of a cell is the baseline of the first line of text in the cell. If there is no text, the baseline is the baseline of whatever object is displayed in the cell, or, if it has none, the bottom of the cell. The maximum distance between the top of the cell and the baseline over all cells that have 'vertical-align:baseline' is used to set the baseline of the row. Here is an example:



Cells 1 and 2 are aligned at their baselines. Cell 2 has the largest height above the baseline, so that determines the baseline of the row. Note that if there is no cell aligned at its baseline, the row will not have (not need) a baseline.

To avoid ambiguous situations, the alignment of cells proceeds in a certain order. First the cells that are aligned on their baseline are positioned. This will establish the baseline of the row. Next the cells with alignment 'top' are positioned.

The row now has a top, possibly a baseline, and a provisional height, which is the distance from the top to the lowest bottom of the cells positioned so far. (See conditions on the cell padding below.)

If any of the remaining cells, those aligned at the bottom or the middle, have a height that is larger than the current height of the row, the height of the row will be increased to the maximum of those cells, by lowering the bottom.

Finally the remaining cells are positioned.

The area between the cell content and the border is part of the cell's padding. The padding at the top and bottom of each cell after positioning must be at least as large as the 'padding' property specifies. The height of the row must be as small as possible without violating this rule.

17.5 Table elements in selectors

Due to the uniqueness of the table structure, there are several additional semantics and pseudo-classes for table cells.

[The implementation cost of this feature may be high. It requires two passes over the style sheet: first to determine the 'display', 'float' and 'position' properties using the normal element hierarchy, than again searching for selectors that match cell element inside column elements. In the second kind of rules, 'display', 'position' and 'float' may not appear.]

17.5.1 Columns and cell selectors

Due to their two-dimensional visual structure, cells have a unique feature in CSS selectors - they have a dual inheritance chain for contextual selectors, both from columns and from rows. This means a stylesheet author can easily write a CSS rule that applies to all the cells "contained" in a column, as well as all the cells contained in a row. The columns are treated as a second parent to the cell, not intermingled with the row inheritance chain.

CSS source:

```
TR.foo TD { color: red }
COL.bar TD { color: blue }
COL.foo TR.bar TD { color: green }
TR.bar COL.bar TD { color: yellow }
```

HTML source:

```
<P>
<TABLE>
<COLGROUP><COL class=foo><COL class=bar></COLGROUP>
<TR class=foo><TD>one</TD><TD>two</TD></TR>
<TR class=bar><TD>three</TD><TD>four</TD></TR>
</TABLE>
```

Cell "one" will be red, cell "two" will be blue. Since the row and column inheritance chains are not intermingled, the second two rules in the stylesheet will not apply to any of the content, and therefore cell "three" will be default color, and cell "four" will be blue.

If a cell appears in multiple columns (e.g. the cell has a column-span of more than 1), all column parents will apply in the selector.

17.5.2 Row, column and cell pseudo-classes

There are several new row and column pseudo-classes available for table formatting.

[The implementation cost of this feature is probably even higher than the one in the previous section. Not only does it require two passes, it also precludes the use of hash tables for quick access to rules.]

Row pseudo-classes:

- *:row[n], :row[%n], :row[%n+m]* - This pseudo-class allows the author to specify "the third row" (":row[3]"), "every third row" (":row[%3]"), or "every third row, starting with the second row" (":row[%3 + 1") (rows are zero-indexed).
- *:row-even* - This is a shortcut for a pseudo-class of ":row[%2]".
- *:row-odd* - This is a shortcut for a pseudo-class of ":row[%2+1.

Column pseudo-classes:

- *:column[n], :column[%n], :column[%n+m]* - This pseudo-class allows the author to specify (for example) "the third column" (":column[3]"), "every third column" (":column[%3]"), or "every third column, starting with the second column" (":column[%3 + 1") (columns are zero-indexed).
- *:column-even* - This is a shortcut for a pseudo-class of ":column[%2]".
- *:column-odd* - This is a shortcut for a pseudo-class of ":column[%2+1]".

17.6 HTML 4.0 default table stylesheet

The following CSS2 stylesheet attempts to codify HTML 4.0 table behavior.

```
TABLE {
        display: table;
        border-collapse: separate;
        table-layout: auto;
        }
COLGROUP { display: table-column-group; column-span: attr(colspan); }
COL { display: table-column; column-span: attr(span); }
THEAD { display: table-header-group; row-span: attr(rows) }
TBODY { display: table-footer-group; row-span: attr(rows) }
TFOOT { display: table-row-group; row-span: attr(rows) }
TR { display: table-row }
TD { display: table-cell; column-span: attr(colspan); row-span: attr(rowspan) }
CAPTION { display: table-caption }
```

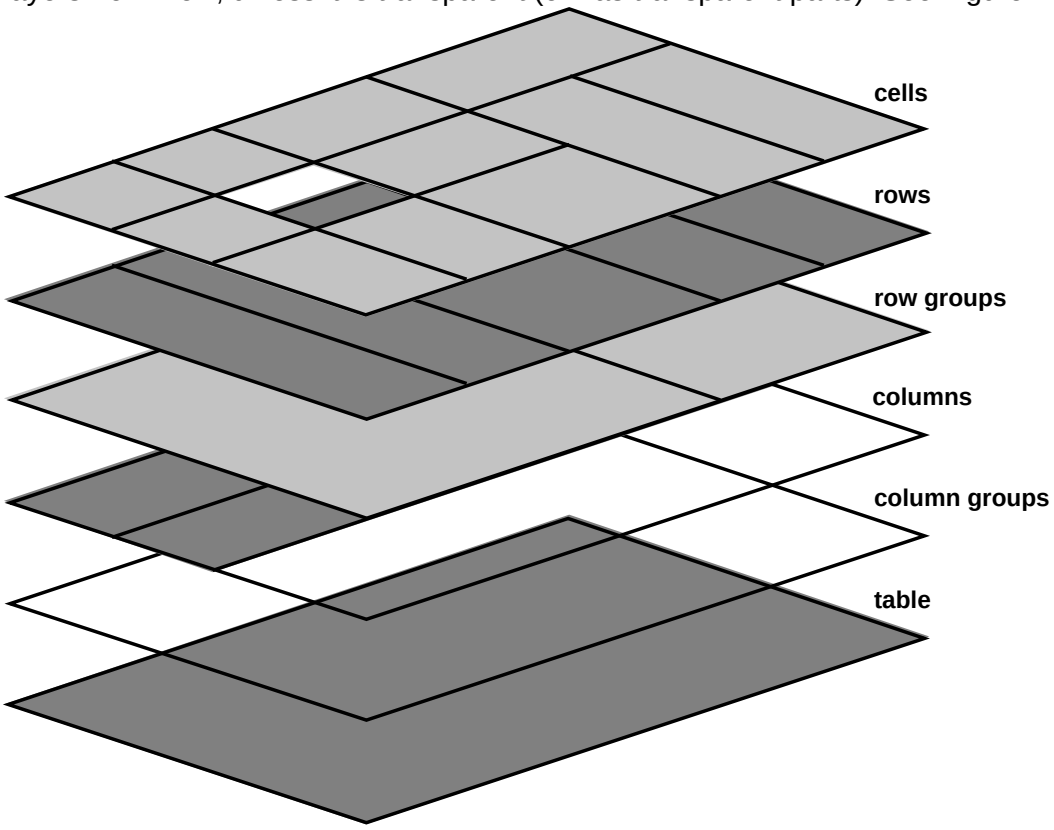
17.7 UNDER CONSTRUCTION!

The rest of this chapter is material that may merge with the preceding sections, or replace some of them.

17.7.1 Table layout

A table is made up of one table element, several columns possibly grouped into column groups, and several rowgroups, containing rows, which in turn contain cells. (For speech style sheets, the cells are further subdivided into header and data cells.) The spatial layout is governed by a grid. All boxes that make up the table have to align with the grid.

One can think of a table as built from six layers. Each layer hides the lower layers from view, unless it is transparent (or has transparent parts). See Figure 1.



1. The lowest layer is a single plane, representing the table box itself. (Note that like all boxes, it may be transparent).
2. The next layer contains the column groups. The columns groups are as tall as the table, but they need not cover the whole table horizontally.
3. On top of the column groups are the areas representing the column boxes. Like column groups, columns are as tall as the table, but need not cover the whole table horizontally.
4. Next is the layer containing the row groups. Each row group is as wide as the table. Together, the row groups completely cover the table from top to bottom.

5. The last but one layer contains the rows. The rows also cover the whole table.
6. The topmost layer contains the cells themselves, and the borders in between them. As the figure shows, the cells don’t have to cover the whole table, but may leave "holes."

To position the table elements, we assume a hypothetical grid, consisting of an infinite number of columns and rows of "grid cells." All table elements (table box, row boxes, cell boxes, etc.) are rectangular and are aligned with the grid: they occupy a whole number of grid cells, determined according to the following rules.

Columns are placed next to each other in the order they occur. Each one occupies the number of grid columns given by its ’column-span’ property. A column group occupies the same columns as the columns contained in it. The first column may be either on the left or on the right, depending on the value of the ’direction’ property of the table.

Each row box occupies one row of grid cells. Together, the row boxes fill the table from top to bottom in the order they occur in the source document, or, stated differently: the table occupies exactly as many grid rows as there are row elements.

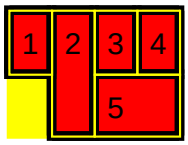
A row group occupies the same grid cells as the rows inside the row group together.

Each cell occupies a rectangle of ’column-span’ grid cells wide and ’row-span’ grid cells high. The top row of this rectangle of grid cells must be in the row occupied by the cell’s parent. The rectangle must be as far to the left as possible, but may not overlap with any other cell, and must be to the right of all cells in the same row that are earlier in the source document. (If the ’direction’ of the table is ’right-to-left’, interchange "left" and "right" in the previous sentence.)

Cells are ’row-span’ high only if there are enough rows: a cell cannot extend below the last row box; it is made shorter until it fits.

Note that there may be "holes" left between the cells. These holes are transparent, and the lower layers of the table are visible through them. Example:

```
<HTML>
<HEAD>
<STYLE type="text/css">
  TABLE {background: #ff0}
  TD {background: red; border: double black}
</STYLE>
</HEAD>
<BODY>
<P>
<TABLE>
  <TR>
    <TD> 1
    <TD rowspan="2"> 2
    <TD> 3
    <TD> 4
  </TR>
  <TR>
    <TD>
    <TD colspan=2> 5
  </TR>
</TABLE>
</BODY>
</HTML>
```



17.7.2 Computing widths and heights

The principle for determining the width of each column is as follows:

1. The width is determined by the ’width’ property of the column box.
2. However, if there is no column box, or its ’width’ is ’auto’, the width is given by the width requirements of the cells in the column.
3. If the value of ’width’ for the first cell in the column is ’auto’, the UA finds the "optimal" width of the column, based on some heuristics.

More details are given below.

The width of the table is given by its ’width’ property. If that is ’auto’, the width is the sum of the column widths. More precisely: the sum of the columns and the borders between them. See "Placement of the borders" [p. 216] below.

Finding the optimal width is complicated. In many cases, what is optimal is a matter of taste. CSS therefore doesn’t define what the optimal width of each column is; a UA is free to use whatever heuristics is has, and is also free to prefer speed over precision. There are a few implementation hints in chapter [???].

The width computation is complicated by cells that span columns and by widths that are specified as percentages. The problem of finding the widths can be regarded as a constraint resolution system, that may be over- or under-constrained.

A percentage is relative to the table width. If the table’s width is ’auto’, a percentage represents a constraint on the column’s width, which a UA should try to satisfy. (Obviously, this is not always possible: if the column’s width is ’110%’, the constraint cannot be satisfied inside a table whose ’width’ is ’auto’.)

A cell that spans columns, provides a constraint on the sum of the widths of the columns it spans.

If a cell’s content doesn’t "fit" the width of the column, the ’overflow’ property determines what happens to it. Similarly, if the ’width’ of the table is not ’auto’, and the sum of the columns is not equal to the table’s width, the ’overflow’ property of the table determines what happens.

17.7.3 Placement of the borders: ’cell-spacing’

For block-level and inline elements, the position of the border relative to the content of the element is determined by the margin and the padding. But in a table, the positions of the borders are constrained by the fact that they have to line up from one row to the next and from one column to the next.

There are two distinct models for setting borders on table cells. One is most suitable for so-called "2½D" borders (ridge, groove, inset, and outset) around individual cells, the other is suitable for borders that are continuous from one and of the table to the other. Many border styles can be achieved with either model, so it is often a matter of taste which model is used.

The property ’cell-spacing’ selects the model:
’cell-spacing’

Property name:	’cell-spacing’
Value:	none | <length> <length>? | inherit
Initial:	none
Applies to:	table
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

The ’cell-spacing’ property only applies to elements with ’display: table’.

If the value is a length, that amount of space is kept open between the borders of all cells. It may not be negative. The space is filled with the background of the table element.

If there are two length values, the first one is the horizontal spacing, and the second one is the vertical spacing. If there is just one length value, it gives both the horizontal and vertical spacing.

Each cell has its own borders, and the overall width of the table is the sum of the cells, plus the cell-spacing between all borders (see figure 3).

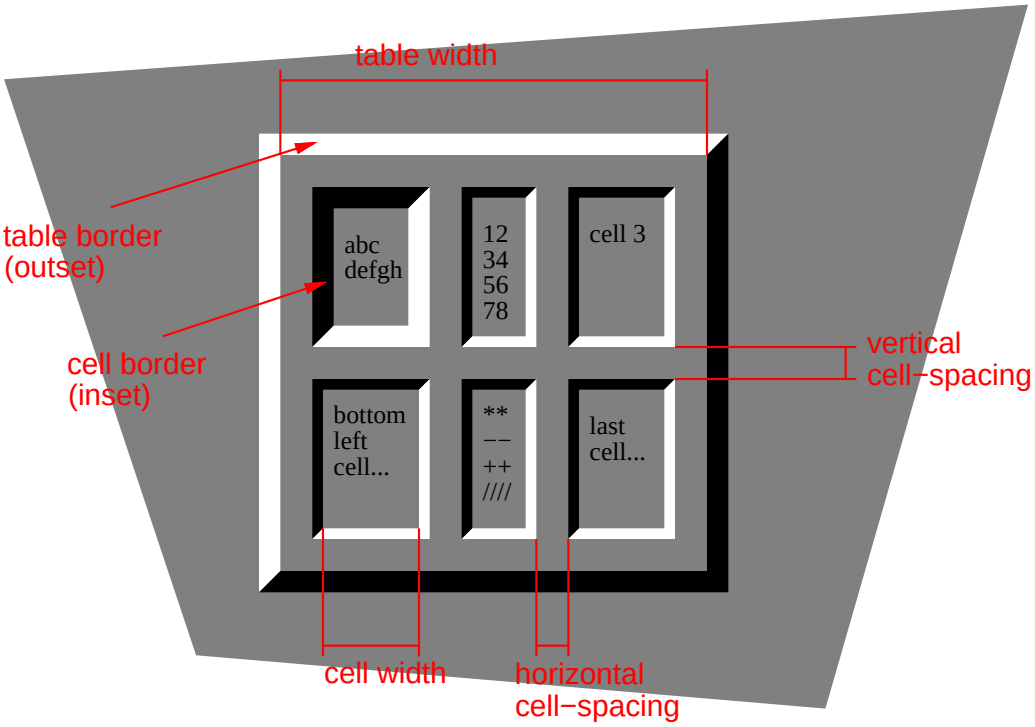


Figure 3. A table with ’cell-spacing’ set to a length value. Note that each cell has its own border, and the table has a separate border as well.

In this mode, rows, columns, row-groups and column groups cannot have borders (i.e., user agents must skip [p. 31] the border properties for those elements.

The table in figure 3 could be the result of a style sheet like this:

```
TABLE {border: outset 10pt; cell-spacing: 15pt}
TD {border: inset 5pt}
TD.special {border: inset 10pt}  /* The top-left cell */
```

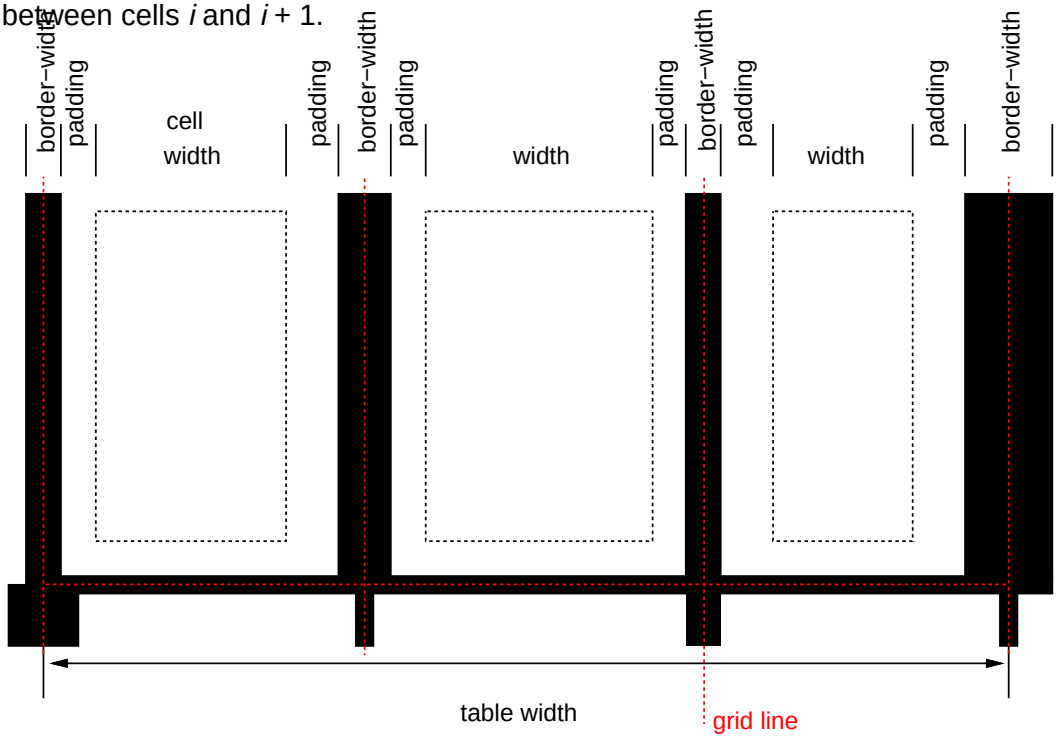
If the value of ’cell-spacing’ is ’none’, the model is more complicated. In this mode it is possible to set borders on rows and row-groups that extend all the way from one end of the table to the other, and it is possible, e.g., to create tables with rules between the cells, and no rules on the outside of the table.

The borders are centered on the grid lines between the cells. A renderer has to find a consistent rule for rounding off in the case of an odd number of discrete units (screen pixels, printer dots).

The diagram below shows how the width of the table, the widths of the borders, the padding and the cell width interact. Their relation is given by the following equation, which holds for every row of the table:

$$\textit{table-width} = \textit{border-width}_0 + \textit{padding-left}_1 + \textit{width}_1 + \textit{padding-right}_1 + \textit{border-width}_1 + \textit{padding-left}_2 + ... + \textit{padding-right}_n + \textit{border-width}_n$$

Here n is the number of cells in the row, and $\textit{border-width}_i$ refers to the border between cells i and $i + 1$.



Note that for a table element, using ’cell-spacing’ ’none’, the width of the table includes half the border, and that a table doesn’t have a padding. It does have a margin, however.

17.7.4 Conflict resolution for borders

If ’cell-spacing’ is ’none’, the style of the borders between the cells is found by comparing the border properties of all the boxes (cells, columns, the table itself, etc.) that meet at that border. Columns and rows can also have borders, but they are only drawn when they coincide with a cell border.

To find the border style at each side of a grid cell, the following properties have to be compared:

1. Those of the one or two cells that have an edge here. Less than two can occur at the edge of the table, but also at the edges of "holes" (unoccupied grid cells).
2. Those of the columns that have an edge here.
3. Those of the column groups that have an edge here.
4. Those of the rows that have an edge here.
5. Those of the row groups that have an edge here.
6. Those of the table, if this is the edge of the table.

This will give between 0 and 8 ’border’ values. Each value is made up of a ’border-width’, ’border-color’ and ’border-style’. The border with the largest width will be drawn. If there are two or more with the same width, but different style, then the one with a style near the start of the following list will be drawn:

’blank’, ’double’, ’solid’, ’dashed’, ’dotted’, ’ridge’, ’groove’, ’none’

If the style is ’outset’, it will be drawn as ’ridge’ instead, and ’inset’ will be drawn as ’groove’.

If the borders only differ in color, a color different from the ’color’ property of the two cells on either side will be preferred over a color that only differs from one of the cells, which in turn will be chosen over a border that doesn’t differ in color from the cells.

If none of these rules determine the color of the border, the UA is free to choose one of the colors.

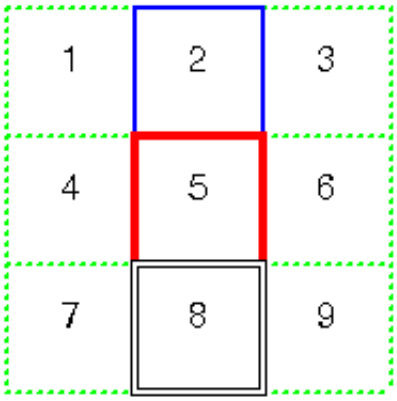
Here is an example:

```
TD.blue {border: medium solid blue} TD.thick {border: thick solid red}
TD.double {border: thick double black} TR {border: medium dotted
green}
```

with this document:

```
<P>
<TABLE>
<TR><TD>1<TD class="blue">2<TD>2
<TR><TD>4<TD class="thick">5<TD>6
<TR><TD>7<TD class="double">8<TD>9
</TABLE>
```

This will be the result:



Here is a table with horizontal rules between the rows. The top border of the table is set to 'blank' to suppress the top border of the first row.

```
TR {border-top: solid}
TABLE {border-top: blank}
```

a	b	c
3	4	5
5	12	13

In this case the same effect can also be achieved without setting a 'blank' border on TABLE, by addressing the first row separately. Which method is preferred is a matter of taste.

```
TR:first-child {border-top: none}
TR {border-top: solid}
```

17.7.5 Properties for columns and rows

Only four properties apply to a column box or column-group box: 'border', 'background', 'width', and 'column-span'. The first two are actually shorthand properties, so all the border properties and all the background properties apply.

Only 'border' and 'background' apply to a row or row-group. But note that you can set inherited properties on rows and row-groups, and they will be inherited by the cells.

17.7.6 Vertical alignment of cells in a row

The cells in a row are aligned somewhat like letters on a line. Each cell, or rather each cell's content, has a baseline, a top, a middle and a bottom, and so does the row itself. The value of the 'vertical-align' property of the cells determines on which of these lines they are aligned:

baseline
: the baseline of the cell is put at the same height as the baseline of the row (see below for the definition of baselines of cells and rows)

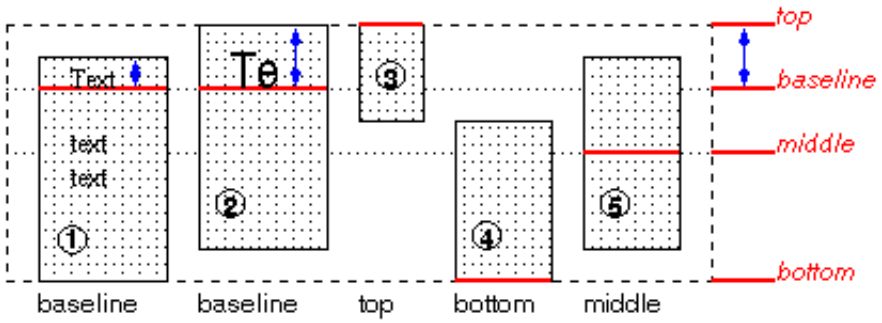
top
: the top of the cell is aligned with the top of the row

bottom
: the bottom of the cell is aligned with the bottom of the row

middle
: the center of the cell is aligned with the center of the row

sub, super, text-top, text-bottom
: these values do not apply to cells; the cell is aligned at the baseline instead

The baseline of a cell is the baseline of the first line of text in the cell. If there is no text, the baseline is the baseline of whatever object is displayed in the cell, or, if it has none, the bottom of the cell. The maximum distance between the top of the cell and the baseline over all cells that have 'vertical-align:baseline' is used to set the baseline of the row. Here is an example:



Cells 1 and 2 are aligned at their baselines. Cell 2 has the largest height above the baseline, so that determines the baseline of the row. Note that if there is no cell aligned at its baseline, the row will not have (not need) a baseline.

To avoid ambiguous situations, the alignment of cells proceeds in a certain order. First the cells that are aligned on their baseline are positioned. This will establish the baseline of the row. Next the cells with alignment 'top' are positioned.

The row now has a top, possibly a baseline, and a provisional height, which is the distance from the top to the lowest bottom of the cells positioned so far. (See conditions on the cell padding below.)

If any of the remaining cells, those aligned at the bottom or the middle, have a height that is larger than the current height of the row, the height of the row will be increased to the maximum of those cells, by lowering the bottom.

Finally the remaining cells are positioned.

The area between the cell content and the border is part of the cell's padding. The padding at the top and bottom of each cell after positioning must be at least as large as the 'padding' property specifies. The height of the row must be as small as possible without violating this rule.

17.7.7 Horizontal alignment of cells in a column

A cell is similar to a block in the way its contents are rendered, that means, in particular, that 'text-align' applies to it. However, tables also allow a way of aligning text that does not apply to other blocks, and that is aligning the contents of several cells so that they all align on, e.g., a decimal point (".")

More precisely, if the value of 'alignment' for a certain cell is a string, that cell has an *alignment point*, which is the start of that string. The alignment point must be straight above or below the alignment points of all other cells in the same column that have an alignment point. (Note that the other cells do not need to have the same value for 'text-align'; as long as they are aligned on a string, they have an alignment point.)

Aligning text in this way is only useful if the text is short enough not to be broken over several lines. The result is undefined if the text *is* broken.

If the string occurs more than once in the cell's content, the alignment point is the start of the first occurrence.

If the string doesn't occur, the alignment point is the end of the content.

17.7.8 Table captions: the 'caption-side' property

'caption-side'

Property name:	'caption-side'
Value:	top | bottom | inherit
Initial:	top
Applies to:	caption elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

[Values top-left, bottom-left, top-right and bottom-right also proposed. They would make the caption into something similar to a float.] ['top' means caption is a block above the table, 'bottom' means it is a block after the table]

17.7.9 Generating speech: the 'speak-header' property

'speak-header'

Property name:	'speak-header'
Value:	once | always | inherit
Initial:	once
Applies to:	header cells
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

[Does 'speak-header' apply to TH or to TD?]

When a table is spoken by a speech generator, the relation between the data cells and the header cells must be expressed in a different way than by horizontal and vertical alignment. Some speech browsers may allow a user to

move around in the 2-dimensional space, thus giving them the opportunity to map out the spatially represented relations. When that is not possible, the style sheet must specify at which points the headers are spoken.

CSS supports two possibilities: the headers are spoken before every cell, or only before a cell when that cell is associated with a different header than the previous cell.

[Add speak:header-cell|data-cell, and some way to mirror the axis/headers attributes? BB]

It is assumed that a speech UA analyzes the table as specified in the HTML 4.0 specification, to find for each data cell the header cells with which it is associated. In summary, the algorithm is to go up in the column and find all header cells, and to go towards the start of the row to find all header cells there. If a data cell is found above a header cell, then the search for header cells in the column stops there. Similarly, if a data cell is found in front of a header cell, the search in that row stops.

Since sometimes header cells are not put in the column or row to which they apply (see e.g., the cells "San Jose" and "Seattle" in the example below), an explicit association using the "axis" and "headers" attributes must be made. The example below shows the required mark-up

Travel Expense Report

	Meals	Hotels	Transport	subtotals
San Jose				
25-Aug-97	37.74	112.00	45.00	
26-Aug-97	27.28	112.00	45.00	
subtotals	65.02	224.00	90.00	379.02
Seattle				
27-Aug-97	96.25	109.00	36.00	
28-Aug-97	35.00	109.00	36.00	
subtotals	131.25	218.00	72.00	421.25
Totals	196.27	442.00	162.00	**800.27**

This presents the money spent on meals, hotels and transport in two locations (San Jose and Seattle) for successive days. Conceptually, you can think of the table in terms of a n-dimensional space. The headers of this space are: location, day, category and subtotal. Some cells define marks along an axis while others give money spent at points within this space. The HTML markup for this table is:

```
<TABLE>
<CAPTION>Travel Expense Report</CAPTION>
<TR>
  <TH></TH>
  <TH>Meals</TH>
  <TH>Hotels</TH>
  <TH>Transport</TH>
  <TH>subtotal</TH>
</TR>
<TR>
  <TH id="san-jose" axis="san-jose">San Jose</TH>
</TR>
<TR>
  <TH headers="san-jose">25-Aug-97</TH>
  <TD>37.74</TD>
  <TD>112.00</TD>
```

```
  <TD>45.00</TD>
  <TD></TD>
</TR>
<TR>
  <TH headers="san-jose">26-Aug-97</TH>
  <TD>27.28</TD>
  <TD>112.00</TD>
  <TD>45.00</TD>
  <TD></TD>
</TR>
<TR>
  <TH headers="san-jose">subtotal</TH>
  <TD>65.02</TD>
  <TD>224.00</TD>
  <TD>90.00</TD>
  <TD>379.02</TD>
</TR>
<TR>
  <TH id="seattle" axis="seattle">Seattle</TH>
</TR>
<TR>
  <TH headers="seattle">27-Aug-97</TH>
  <TD>96.25</TD>
  <TD>109.00</TD>
  <TD>36.00</TD>
  <TD></TD>
</TR>
<TR>
  <TH headers="seattle">28-Aug-97</TH>
  <TD>35.00</TD>
  <TD>109.00</TD>
  <TD>36.00</TD>
  <TD></TD>
</TR>
<TR>
  <TH headers="seattle">subtotal</TH>
  <TD>131.25</TD>
  <TD>218.00</TD>
  <TD>72.00</TD>
  <TD>421.25</TD>
</TR>
<TR>
  <TH>Totals</TH>
  <TD>196.27</TD>
  <TD>442.00</TD>
  <TD>162.00</TD>
  <TD>800.27</TD>
</TR>
</TABLE>
```

By providing the data model in this way, authors make it possible for speech enabled-browsers to explore the table in rich ways, e.g. each cell could be spoken as a list, repeating the applicable headers before each data cell:

```
  San Jose, 25-Aug-97, Meals:  37.74
  San Jose, 25-Aug-97, Hotels:  112.00
  San Jose, 25-Aug-97, Transport:  45.00
 ...
```

The browser could also speak the headers only when they change:

```
San Jose, 25-Aug-97, Meals: 37.74
    Hotels: 112.00
    Transport: 45.00
  26-Aug-97, Meals: 27.28
    Hotels: 112.00
...
```

The ’speak-header’ property of a header cell determines when it is spoken: before every data cell, or only when the previous cell spoken wasn’t associated with this header.

17.7.10 Table implementation notes

[Move to appendix]

[Minimum/maximum]

18 User interface

Contents

1. Cursors: [p. 227] the ’cursor’ property
2. User preferences for colors [p. 228]
3. User preferences for fonts [p. 230]
4. Other rendering issues that depend on user agents [p. 230]
    1. Magnification [p. 230]

18.1 Cursors: the ’cursor’ property

’cursor’

Property name:	’cursor’
Value:	[[auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help] || <uri>?] | inherit
Initial:	auto
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	visual [p. 65]

This property specifies the type of cursor to be displayed for the pointing device. Values have the following meanings:

auto
: The UA determines the cursor to display based on the current context.

crosshair
: A simple crosshair (e.g., short line segments resembling a "+" sign).

default
: The platform-dependent default cursor. Often rendered as an arrow.

pointer
: The cursor is a pointer that indicates a link.

move
: Indicates something is to be moved

***-resize**
: Indicates that the edge is to be moved.

text
: Indicates text that may be selected. Often rendered as an I-bar.

wait
: A cursor to indicate that the program is busy and the user should wait. Often rendered as a watch or hourglass.

help
: Help is available for the object under the cursor. Often rendered as a

question mark or a balloon.

<uri>

The user agent should retrieve the cursor from the resource designated by the URI. It is an error if the resource is not a proper cursor. User agents may handle this error condition in different ways.

The following example makes the cursor stored in the resource `mything.cur` the "text" cursor.

```
P { cursor : text url(mything.cur) }
```

CSS2 does not allow users to specify animated cursors.

18.2 User preferences for colors

In addition to being able to assign pre-defined color values to text, backgrounds, etc., CSS2 allows authors to specify colors in a manner that integrates them into the user's graphic environment. For instance, color-blind users may have their environment configured to avoid specific colors. Style rules that take into account user preferences thus offer the following advantages:

1. They produce pages that fit the user's defined look and feel.
2. They produce pages that may be more accessible as the current user settings may be related to a disability.

The set of values defined for system colors is intended to be exhaustive. For systems that do not have a corresponding value, the specified value should be mapped to the nearest system attribute, or to a default color.

The following lists additional values for color related CSS attributes and their general meaning. Any color property (e.g., 'color' or 'background-color') can take one of the following names:

activeborder

Active window border.

activecaption

Active window caption.

appworkspace

Background color of multiple document interface.

background

Desktop background.

buttonface

Face color for three-dimensional display elements.

buttonhighlight

Dark shadow for three-dimensional display elements (for edges facing away from the light source).

Shadow color for three-dimensional display elements.

buttontext

Text on push buttons.

captiontext

Text in caption, size box, and scroll bar arrow box.

graytext
Grayed (disabled) text. This color is set to #000 if the current display driver does not support a solid gray color.
highlight
Item(s) selected in a control.
highlighttext
Text of item(s) selected in a control.
inactiveborder
Inactive window border.
inactivecaption
Inactive window caption.
inactivecaptiontext
Color of text in an inactive caption.
infobackground
Background color for tooltip controls.
infotext
Text color for tooltip controls.
menu
Menu background.
menutext
Text in menus.
scrollbar
Scroll bar gray area.
threeddarkshadow
Dark shadow for three-dimensional display elements.
threedface
Face color for three-dimensional display elements.
threedhighlight
Highlight color for three-dimensional display elements.
threedlightshadow
Light color for three-dimensional display elements (for edges facing the light source).
threedshadow
Dark shadow for three-dimensional display elements.
window
Window background.
windowframe
Window frame.
windowtext
Text in windows.

For example, to set the foreground and background colors of a paragraph to the same foreground and background colors of the user's window, write the following:

```
P { color: windowtext; background-color: window }
```

18.3 User preferences for fonts

As for colors, authors may specify fonts in a way that makes use of a user's system resources. Please consult the 'font' property for details.

18.4 Other rendering issues that depend on user agents

18.4.1 Magnification

The CSS working group considers that the magnification of a document or portions of a document should not be specified through style sheets. User agents may support such magnification in different ways (e.g., larger images, louder sounds, etc.)

When magnifying a page, UAs should preserve the relationships between positioned elements. For example, a comic strip may be composed of images with overlaid text elements. When magnifying this page, a user agent should keep the text within the comic strip balloon.

19 Aural style sheets

Contents

1. Introduction to aural style sheets [p. 231]
2. Volume properties [p. 231] : ’volume’
3. Speaking properties [p. 232] : ’speak’
4. Pause properties [p. 233] : ’pause-before’, ’pause-after’, and ’pause’
5. Cue properties [p. 234] : ’cue-before’, ’cue-after’, and ’cue’
6. Mixing properties [p. 236] : ’play-during’
7. Spatial properties [p. 237] : ’azimuth’ and ’elevation’
8. Voice characteristic properties [p. 239] : ’speech-rate’, ’voice-family’, ’pitch’, ’pitch-range’, ’stress’, and ’richness’
9. Speech properties [p. 242] : ’speak-punctuation’, ’speak-date’, ’speak-numeral’, and ’speak-time’

19.1 Introduction to aural style sheets

The aural rendering of a document, already commonly used by the blind and print-impaired communities, combines speech synthesis and "audio icons" Often such aural presentation occurs by converting the document to plain text and feeding this to a *screen reader* -- software or hardware that simply reads all the characters on the screen. This results in less effective presentation than would be the case if the document structure were retained. Style Sheet properties for aural presentation may be used together with visual properties (mixed media) or as an aural alternative to visual presentation.

Besides the obvious accessibility advantages, there are other large markets for aural presentation, including in-car use, industrial and medical documentation systems (intranets), home entertainment, and to help illiterate users.

19.2 Volume properties: ’volume’

’volume’

Property name:	’volume’
Value:	<number> | <percentage> | silent | x-soft | soft | medium | loud | x-loud | inherit
Initial:	medium
Applies to:	all elements
Inherited:	yes
Percentage values:	relative to inherited value
Media groups:	aural [p. 65]

Volume refers to the median volume of the waveform. In other words, a highly inflected voice at a volume of 50 might peak well above that. The overall values are likely to be human adjustable for comfort, for example with a physical volume control (which would increase both the 0 and 100 values proportionately); what this property does is adjust the dynamic range.

Values have the following meanings:

<number>
: Any number between ’0’ and ’100’. ’0’ represents the *minimum audible* volume level and 100 corresponds to the *maximum comfortable* level.

<percentage>
: Percentage values are calculated relative to the inherited value, and are then clipped to the range ’0’ to ’100’.

silent
: No sound at all. **Note.** The value ’0’ does not mean the same as ’silent’.

x-soft
: Same as ’0’.

soft
: Same as ’25’.

medium
: Same as ’50’.

loud
: Same as ’75’.

x-loud
: Same as ’100’.

User agents should allow the values corresponding to ’0’ and ’100’ to be set by the listener. No one setting is universally applicable; suitable values depend on the equipment in use (speakers, headphones), the environment (in car, home theater, library) and personal preferences. Some examples:

- A browser for in-car use has a setting for when there is lots of background noise. ’0’ would map to a fairly high level and ’100’ to a quite high level. The speech is easily audible over the road noise but the overall dynamic range is compressed. Cars with better insulation might allow a wider dynamic range.
- Another speech browser is being used in an apartment, late at night, or in a shared study room. ’0’ is set to a very quiet level and ’100’ to a fairly quiet level, too. As with the first example, there is a low slope; the dynamic range is reduced. The actual volumes are low here, whereas they were high in the first example.
- In a quiet and isolated house, an expensive hi-fi home theater setup. ’0’ is set fairly low and ’100’ to quite high; there is wide dynamic range.

The same author style sheet could be used in all cases, simply by mapping the ’0’ and ’100’ points suitably at the client side.

19.3 Speaking properties: ’speak’

’speak’

Property name: ’speak’
Value: normal | none | spell-out | inherit
Initial: normal
Applies to: all elements
Inherited: yes
Percentage values: N/A
Media groups: aural [p. 65]

This property specifies whether text will be rendered aurally and if so, in what manner (somewhat analogous to the ’display’ property). The possibles values are:

none
Suppresses aural rendering so that, unless overridden recursively, the element and its children require no time to render.
normal
Uses regular language-dependent pronunciation rules for rendering an element and its children.
spell-out
Spells the text one letter at a time (useful for acronyms and abbreviations).

Note the difference between an element whose ’volume’ property has a value of ’silent’ and an element whose ’speak’ property has the value ’none’. The former takes up the same time as if it had been spoken, including any pause before and after the element, but no sound is generated. This may be used in language teaching applications, for example. A pause is generated for the pupil to speak the element themselves. Note that since the value of this property is inherited, child elements will also be silent. Child elements may however set the volume to a non-silent value and will then be spoken. On the other hand, elements for which the ’speak’ property has the value ’none’ are not spoken and take no time. Child elements may however override this value and may be spoken normally.

19.4 Pause properties: ’pause-before’, ’pause-after’, and ’pause’

’pause-before’

Property name: ’pause-before’
Value: <time> | <percentage> | inherit
Initial: depends on user agent
Applies to: all elements
Inherited: no
Percentage values: see prose
Media groups: aural [p. 65]

’pause-after’

Property name: ’pause-after’
Value: <time> | <percentage> | inherit
Initial: depends on user agent
Applies to: all elements
Inherited: no
Percentage values: see prose
Media groups: aural [p. 65]

These properties specify a pause to be observed before (or after) speaking an element’s content. Values have the following meanings:

<time>
Expresses the pause in absolute time units (seconds and milliseconds).
<percentage>
Refers to the inverse of the value of the ’speech-rate’ property. For example, if the speech-rate is 120 words per minute (i.e., a word takes half a second, or 500ms) then a ’pause-before’ of 100% means a pause of 500 ms and a ’pause-before’ of 20% means 100ms.

Authors should use relative units to create more robust style sheets in the face of large changes in speech-rate.

’pause’

Property name: ’pause’
Value: [[<time> | <percentage>]{1,2}] | inherit
Initial: depends on user agent
Applies to: all elements
Inherited: no
Percentage values: see descriptions of ’pause-before’ and ’pause-after’
Media groups: aural [p. 65]

The ’pause’ property is a shorthand for setting ’pause-before’ and ’pause-after’. If two values are given, the first value is ’pause-before’ and the second is ’pause-after’. If only one value is given, it applies to both properties.

Examples:

```
H1 { pause: 20ms } /* pause-before: 20ms; pause-after: 20ms */
H2 { pause: 30ms 40ms } /* pause-before: 30ms; pause-after: 40ms */
H3 { pause-after: 10ms } /* pause-before: ?; pause-after: 10ms */
```

19.5 Cue properties: ’cue-before’, ’cue-after’, and ’cue’

’cue-before’

Property name: ’cue-before’
Value: <uri> | none | inherit
Initial: none
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: aural [p. 65]

’cue-after’

Property name: ’cue-after’
Value: <uri> | none | inherit
Initial: none
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: aural [p. 65]

Auditory icons are another way to distinguish semantic elements. Sounds may be played before, and/or after the element to delimit it. Values have the following meanings:

<uri>
The URI designates an audio icon resource.
none
No audio icon is specified.

For example:

```
A {cue-before: url(bell.aiff); cue-after: url(dong.wav) }
H1 {cue-before: url(pop.au); cue-after: url(pop.au) }
```

’cue’

Property name: ’cue’
Value: [<’cue-before’> || <’cue-after’>] | inherit
Initial: not defined for shorthand properties
Applies to: all elements
Inherited: no
Percentage values: N/A
Media groups: aural [p. 65]

The ’cue’ property is a shorthand for setting ’cue-before’ and ’cue-after’. If two values are given, the first value is ’cue-before’ and the second is ’cue-after’. If only one value is given, it applies to both properties.
The following two rules are equivalent:

```
H1 {cue-before: url(pop.au); cue-after: url(pop.au) }
H1 {cue: url(pop.au) }
```

19.6 Mixing properties: ’play-during’

’play-during’

Property name:	’play-during’
Value:	<uri> | mix? repeat? | auto | none | inherit
Initial:	auto
Applies to:	all elements
Inherited:	no
Percentage values:	N/A
Media groups:	aural [p. 65]

Similar to the ’cue-before’ and ’cue-after’ properties, this property specifies a sound to be played as a background while an element’s content is spoken. Values have the following meanings:

<uri>
: The sound designated by this <uri> is played as a background while the element’s content is spoken.

mix
: When present, this keyword means that the sound inherited from the parent element’s ’play-during’ property continues to play and the sound designated by the <uri> is mixed with it. If ’mix’ is not specified, the sound replaces the sound of the parent element.

repeat
: When present, this keyword means that the sound will repeat if it is too short to fill the entire duration of the element. Otherwise, the sound plays once and then stops. This is similar to the background repeat properties in CSS2. If the sound is too long for the element, it is clipped once the element is spoken.

auto
: The sound of the parent element continues to play (it is not restarted, which would have been the case if this property had been inherited).

none
: Means that there is silence - the sound of the parent element (if any) is silent during the current element and continues after the current element.

Examples:

```
BLOCKQUOTE.sad {play-during: url(violins.aiff) }
BLOCKQUOTE Q {play-during: url(harp.wav) mix}
SPAN.quiet {play-during: none }
```

If a stereo icon is dereferenced, the central point of the stereo pair should be placed at the azimuth for that element and the left and right channels should be placed to either side of this position.

19.7 Spatial properties: ’azimuth’ and ’elevation’

Spatial audio is an important stylistic property for aural presentation. It provides a natural way to tell several voices apart, as in real life (people rarely all stand in the same spot in a room). Stereo speakers produce a lateral sound stage. Binaural headphones or the increasingly popular 5-speaker home theater setups can generate full surround sound, and multi-speaker setups can create a true three-dimensional sound stage. VRML 2.0 also includes spatial audio, which implies that in time consumer-priced spatial audio hardware will become more widely available.

’azimuth’

Property name:	’azimuth’
Value:	<angle> | [[left-side | far-left | left | center-left | center | center-right | right | far-right | right-side] || behind] | leftwards | rightwards | inherit
Initial:	center
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	aural [p. 65]

Values have the following meanings:

<angle>
Position is described in terms of degrees, within the range ’-360deg’ to ’360deg’. The value ’0deg’ means directly ahead in the center of the sound stage. ’90deg’ is to the right, ’180deg’ behind, and ’270deg’ (or, equivalently and more conveniently, ’-90deg’) to the left.

left-side
Same as ’270deg’. With ’behind’, ’270deg’.

far-left
Same as ’300deg’. With ’behind’, ’240deg’.

left
Same as ’320deg’. With ’behind’, ’220deg’.

center-left
Same as ’340deg’. With ’behind’, ’200deg’.

center
Same as ’0deg’. With ’behind’, ’180deg’.

center-right
Same as ’20deg’. With ’behind’, ’160deg’.

right
Same as ’40deg’. With ’behind’, ’140deg’.

far-right
Same as ’60deg’. With ’behind’, ’120deg’.

right-side
Same as ’90deg’. With ’behind’, ’90deg’.
leftwards
Moves the sound to the left, relative to the current angle. More precisely, subtracts 20 degrees. Arithmetic is carried out modulo 360 degrees. Note that ’leftwards’ is more accurately described as "turned counter-clockwise," since it *always* subtracts 20 degrees, even if the inherited azimuth is already behind the listener (in which case the sound actually appears to move to the right).
rightwards
Moves the sound to the right, relative to the current angle. More precisely, adds 20 degrees. See ’leftwards’ for arithmetic.

This property is most likely to be implemented by mixing the same signal into different channels at differing volumes. It might also use phase shifting, digital delay, and other such techniques to provide the illusion of a sound stage. The precise means used to achieve this effect and the number of speakers used to do so are user agent-dependent; this property merely identifies the desired end result.
Examples:

```
H1   { azimuth: 30deg }
TD.a { azimuth: far-right }          /*  60deg */
#12  { azimuth: behind far-right }   /* 120deg */
P.comment { azimuth: behind }        /* 180deg */
```

If spatial-azimuth is specified and the output device cannot produce sounds *behind* the listening position, user agents should convert values in the rearwards hemisphere to forwards hemisphere values. One method is as follows:

- if 90deg < x <= 180deg then x := 180deg - x
- if 180deg < x <= 270deg then x := 540deg - x

’elevation’

Property name: ’elevation’
Value: <angle> | below | level | above | higher | lower | inherit
Initial: level
Applies to: all elements
Inherited: yes
Percentage values: N/A
Media groups: aural [p. 65]

Values of this property have the following meanings:

<angle>
Specifies the elevation as an angle, between ’-90deg’ and ’90deg’. ’0deg’ means on the forward horizon, which loosely means level with the listener. ’90deg’ means directly overhead and ’-90deg’ means directly below.
below
Same as ’-90deg’.

level
Same as ’-0deg’.
above
Same as ’90deg’.
higher
Adds 10 degrees to the current elevation.
lower
Subtracts 10 degrees from the current elevation.

The precise means used to achieve this effect and the number of speakers used to do so are undefined. This property merely identifies the desired end result.

Examples:

```
H1   { elevation: above }
TR.a { elevation: 60deg }
TR.b { elevation: 30deg }
TR.c { elevation: level }
```

19.8 Voice characteristic properties: ’speech-rate’, ’voice-family’, ’pitch’, ’pitch-range’, ’stress’, and ’richness’

’speech-rate’

Property name:	’speech-rate’
Value:	<number> | x-slow | slow | medium | fast | x-fast | faster | slower | inherit
Initial:	medium
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	aural [p. 65]

This property specifies the speaking rate. Note that both absolute and relative keyword values are allowed (compare with ’font-weight’). Values have the following meanings:

<number>
Specifies the speaking rate in words per minute, a quantity that varies somewhat by language but is nevertheless widely supported by speech synthesizers.
x-slow
Same as ?
slow
Same as ?

medium
Same as ? Refers to the user's preferred speech-rate setting.
fast
Same as ?
x-fast
Same as ?
faster
Adds ? to current speech rate.
slower
Subtracts ? to current speech rate.

'voice-family'

Property name:	'voice-family'
Value:	[[<specific-voice> | <generic-voice>],]* [<specific-voice> | <generic-voice>] | inherit
Initial:	depends on user agent
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	aural [p. 65]

The value is a comma-separated, prioritized list of voice family names (compare with 'font-family'). Values have the following meanings:

<generic-voice>
Values are voice families (e.g., male, female, child).
<specific-voice>
Values are specific instances (e.g., comedian, trinoids, carlos, lani).

Examples:

```
H1 { voice-family: announcer, male }
P.part.romeo  { voice-family: romeo, male }
P.part.juliet { voice-family: juliet, female }
```

'pitch'

Property name:	'pitch'
Value:	<frequency> | x-low | low | medium | high | x-high | inherit
Initial:	medium
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	aural [p. 65]

Specifies the average pitch of the speaking voice. Values have the following meanings:

<frequency>
Specifies the average pitch of the speaking voice in hertz (Hz).
x-low
Same as ?
low
Same as ?
medium
Same as ?
high
Same as ?
x-high
Same as ?

’pitch-range’

Property name: ’pitch-range’
Value: <number> | inherit
Initial: 50
Applies to: all elements
Inherited: yes
Percentage values: N/A
Media groups: aural [p. 65]

Specifies variation in average pitch. Values have the following meanings:

<number>
A pitch range of 0 produces a flat, monotonic voice. A pitch range of 50 produces normal inflection. Pitch ranges greater than 50 produce animated voices.

’stress’

Property name: ’stress’
Value: <number> | inherit
Initial: 50
Applies to: all elements
Inherited: yes
Percentage values: N/A
Media groups: aural [p. 65]

Specifies the level of stress (assertiveness or emphasis) of the speaking voice. English is a **stressed** language, and different parts of a sentence are assigned primary, secondary or tertiary stress. The value of ’stress’ controls the amount of inflection that results from these stress markers. Values have the following meanings:

<number>
Increasing the value of this property results in the speech being more strongly inflected. It is, in a sense, a companion to the ’pitch-range’ property and is provided to allow developers to exploit higher-end auditory displays.

’richness’

Property name:	’richness’
Value:	<number> | inherit
Initial:	50
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	aural [p. 65]

Specifies the richness (brightness) of the speaking voice. Values have the following meanings:

<number>
The effect of increasing richness is to produce a voice that *carries*. Reducing richness produces a soft, mellifluous voice.

19.9 Speech properties: ’speak-punctuation’, ’speak-date’, ’speak-numeral’, and ’speak-time’

Note. *The following four properties are preliminary and discussion on them is invited.*

An additional speech property, speak-header [p. 222] , is described in the chapter on tables [p. 201]

’speak-punctuation’

Property name:	’speak-punctuation’
Value:	code | none | inherit
Initial:	none
Applies to:	all elements
Inherited:	yes
Percentage values:	N/A
Media groups:	aural [p. 65]

This property specifies how punctuation is spoken. Values have the following meanings:

code
Punctuation such as semicolons, braces, and so on are to be spoken literally.

none
Punctuation is not to be spoken, but instead rendered naturally as various pauses.

’speak-date’

Property name: ’speak-date’
Value: mdy | dmy | ymd | inherit
Initial: depends on user agent
Applies to: all elements
Inherited: yes
Percentage values: N/A
Media groups: aural [p. 65]

This property controls how dates are spoken. Values have the following meanings:

mdy
Month-Day-Year (common in the United States).
dmy
Day-Month-Year (common in Europe).
ymd
Year-Month-Day.

This property would be useful, for example, when combined with an XML element used to identify dates, such as:

```
<PARA>The campaign started on <DATE value="1874-10-21"/>
 and finished <DATE value="1874-10-28/"></PARA>
```

’speak-numeral’

Property name: ’speak-numeral’
Value: digits | continuous | none | inherit
Initial: none
Applies to: all elements
Inherited: yes
Percentage values: N/A
Media groups: aural [p. 65]

This property controls how numerals are spoken. Values have the following meanings:

digits
Speak the numeral as individual digits. Thus, "237" is spoken "Two Three Seven".
continuous
Speak the numeral as a full number. Thus, "237" is spoken "Two hundred thirty seven". Word representations are language-dependent.
none
[What does this mean?]

’speak-time’

Property name: ’speak-time’
Value: 24 | 12 | none | inherit
Initial: none
Applies to: all elements
Inherited: yes
Percentage values: N/A
Media groups: aural [p. 65]

This property controls how times are spoken. Values have the following meanings:

24
Use the 24-hour time system.

12
Use the 12-hour am/pm time system.

none
[What does this mean?]

When used in combination with the ’speak-date’ property, this allows elements with an attribute containing an ISO 8601 format date/time attribute to be presented in a flexible manner.

Appendix A: A sample style sheet for HTML 4.0

This appendix is informative, not normative.

The Base Style Sheet describes the typical rendering of all HTML 4.0 [HTML40] [p. 268]) elements visual UAs. The style sheet is based on extensive research on how current UAs render HTML, and developers are encouraged to use it as a default style sheet in their implementations.

The full presentation of some HTML elements cannot be expressed in CSS2, including replaced elements (IMG, OBJECT), scripting elements (SCRIPT, APPLET), form control elements, frame elements, and BR.

```
/* rendered CSS1-addressable elements and all applicable non-inherited
properties set to initial values and default display types */

A, ABBR, ACRONYM, ADDRESS, BDO, BLOCKQUOTE, BODY, BUTTON, CITE, CODE,
DD, DEL, DFN, DIV, DL, DT, EM, FIELDSET, FORM, H1, H2, H3, H4, H5, H6,
IFRAME, IMG, INS, KBD, LABEL, LI, OBJECT, OL, P, Q,
SAMP, SMALL, SPAN, STRONG, SUB, SUP, UL, VAR, APPLET, BASEFONT,
B, BIG, CENTER, DIR, FONT, HR, I, MENU, PRE, S, STRIKE, TT, U {
        background: transparent;
        width: auto;
        height: auto;
        text-decoration: none;
        margin: 0;
        padding: 0;
        border: 0;
        float: none;
        clear: none;
        vertical-align: baseline;
        list-style-image: none;
        list-style-type: disc;
        list-style-position: outside;
        }

ADDRESS, BLOCKQUOTE, BODY, DD, DIV, DL, DT, FIELDSET, FORM, FRAME, FRAMESET
H1, H2, H3, H4, H5, H6, IFRAME, NOSCRIPT, NOFRAMES, OBJECT,
OL, P, UL, APPLET, CENTER, DIR, HR, MENU, PRE {
        display: block;
        }

A, ABBR, ACRONYM, BDO, BUTTON, CITE, CODE, DEL, DFN, EM, IMG, INPUT, INS,
ISINDEX, KBD, LABEL, MAP, Q, SAMP, SELECT, SMALL, SPAN, STRONG, SUB, SUP,
TEXTAREA, VAR, B, BASEFONT, BIG, FONT, I, S, STRIKE, TT, U {
        display: inline;
        }

LI      {
        display: list-item;
        }

/* Don't display HEAD elements */
HEAD, LINK, META, PARAM, STYLE, TITLE  {
        display: none;
        }

/* Begin tree of inherited properties and cascades. */

BODY    {
        font-size: 1em;
        line-height: 1.33em;
        margin: 8px;
        /* background flush with initial containing block edge */
        background-position: -8px -8px;
        word-spacing: normal;
        letter-spacing: normal;
        text-transform: none;
        text-align: left;
        text-indent: 0;
```

```
        white-space: normal;
        }

H1      {
        font-size: 2em;
        margin: .67em 0;
        }

H2      {
        font-size: 1.5em;
        margin: .83em 0;
        }

H3      {
        font-size: 1.17em;
        line-height: 1.17em;
        margin: 1em 0;
        }

H4, P, BLOCKQUOTE, UL, OL, DL, DIR, MENU        {
        margin: 1.33em 0;
        }

H5      {
        font-size: .83em;
        line-height: 1.17em;
        margin: 1.67em 0;
        }

H6      {
        font-size: .67em;
        margin: 2.33em 0;
        }

H1, H2, H3, H4, H5, H6, B, STRONG       {
        font-weight: bolder;
        }

BLOCKQUOTE      {
        margin-left: 40px;
        margin-right: 40px;
        }

I, CITE, EM, VAR, ADDRESS       {
        font-style: italic;
        }

PRE, TT, CODE, KBD, SAMP        {
        font-family: monospace;
        }

PRE     {
        white-space: pre;
        }

BIG     {
        font-size: 1.17em;
        }

SMALL, SUB, SUP {
        font-size: .83em;
        }

SUB     {
        vertical-align: sub;
        }

SUP     {
        vertical-align: super;
        }

S, STRIKE, DEL  {
        text-decoration: line-through;
        }

HR      {
        border: 1px inset;
```

```
        }

OL, UL, DIR, MENU, DD   {
        margin-left: 40px;
        }

OL LI   {
        list-style-type: decimal;
        }

OL UL   {
        margin-top: 0;
        margin-bottom: 0;
        }

UL OL   {
        margin-top: 0;
        margin-bottom: 0;
        }

UL UL   {
        margin-top: 0;
        margin-bottom: 0;
        }

OL OL   {
        margin-top: 0;
        margin-bottom: 0; /* how far to carry such contextual declarations? Exhaustive list
                could be very long. */
        }

U, INS  {
        text-decoration: underline;
        }

CENTER  {
        text-align: center;
        }



/* Table element rendering behavior cannot be described completely in CSS1,
yet the following declarations appear to apply. This section is likely to become
obsolete upon the deployment of a more comprehensive style sheet specification
for tables. */

CAPTION, COL, COLGROUP, LEGEND, TABLE, TBODY, TD, TFOOT, TH, THEAD, TR  {
        background: transparent;
        text-decoration: none;
        margin: 1px;
        padding: 1px;
        border: none;
        float: none;
        clear: none;
        }

TABLE, TBODY, TFOOT, THEAD, TR  {
        display: block;
        background-position: top left;
        width: auto;
        height: auto;
        }

CAPTION, LEGEND, TD, TH {
        display: inline;
        vertical-align: baseline;
        font-size: 1em;
        line-height: 1.33em;
        color: black;
        word-spacing: normal;
        letter-spacing: normal;
        text-transform: none;
        text-align: left;
        text-indent: 0;
        white-space: normal;
        }
```

```
TH      {
        font-weight: bolder;
        text-align: center;
        }

CAPTION {
        text-align: center;
        }


/* proposed default for HTML 4.0's new ABBR/ACRONYM elements */

ABBR, ACRONYM {
        font-variant: small-caps;
        letter-spacing: 0.1em;
           /* This is almost facetious. Should they not have any
           default rendering? Uppercase transform? Not all languages
           distinguish between simple abbreviations and acronyms, and
           not all abbrev. should be capped. */ }

/* not part of the legacy browser default sheet, but an obvious enhancement */

OL OL LI        {
        list-style-type: lower-alpha;
        }

OL OL OL LI     {
        list-style-type: lower-roman
        }
```

Appendix B: Changes from CSS1

Contents

1. New functionality [p. 249]
2. Updated descriptions [p. 249]
3. Changes [p. 249]

CSS2 builds on [CSS1] [p. 267] and all valid CSS1 style sheets are valid CSS2 style sheets. The changes between the CSS1 specification and this specification fall into three groups: new functionality, updated descriptions of CSS1 functionality, and changes to CSS1.

2.1 New functionality

In addition to the functionality of CSS1, CSS2 supports:

- The concept of media types [p. 63] .
- Paged media [p. 131]
- Aural style sheets [p. 231]
- An extended font selection [p. 162] mechanism, including intelligent matching, synthesis, and downloadable fonts. Also, the concept of system fonts has been is introduced, and a new property, 'font-size-adjust', has been added.
- Tables [p. 201] , including new values on 'display' and 'vertical-align'.
- Relative [p. 82] and Absolute positioning [p. 85] , including fixed positioning [p. 85] .
- An extended selector [p. 43] mechanism, including child selectors, adjacent selectors and attribute selectors.
- Generated content and automatic numbering [p. 129]
- Text shadows, through the new
- 'text-shadow' property.
- A new anchor pseudo-class [p. 54] , :hover.
- System colors [p. 228]

2.2 Updated descriptions

The CSS1 specification was short and concise. This specification is much more voluminous and more readable. Much of the additional content describes new functionality, but the description of CSS1 features has also been extended. Except in a few cases described below, the rewritten descriptions do not indicate a change in syntax nor semantics.

2.3 Changes

- In CSS2 color values [p. 39] are clipped with regard to the device gamut, not with regard to the sRGB gamut as in CSS1.
- [strength of constraints based on writing direction]

- In CSS1, several properties (e.g. ’padding’) had values referring to the width of the parent element. This was an error; the value should always refer to the width of a block-level element and this specification reflects this.
- Initial value of ’display’ is ’inline’, not ’block’ as in CSS1.
- [the ’clear’ property now only applies to block-level elements]

Appendix C: Implementation and performance notes

Contents

1. Order of property value calculation [p. 251]
2. Colors [p. 251]
    1. Gamma Correction [p. 251]
3. Fonts [p. 252]
    1. Glossary of font terms [p. 252]
    2. Font retrieval [p. 256]
    3. Meaning of the Panose Digits [p. 256]
    4. Deducing Unicode Ranges for TrueType [p. 259]

3.1 Order of property value calculation

This section is under construction

Due to dependencies among properties, user agents must compute [p. 57] some property values in a specific order.

- 'font-size' must be computed before any property that may take a relative length [p. 36] units.
- 'display', 'position', and 'float' must be computed before other visual flow [p. 67] properties.
- 'line-height' must be computed before 'vertical-align'.
- An element's width and height must be computed before 'background-position'.

3.2 Colors

3.2.1 Gamma Correction

The following information is informative, not normative. See the Gamma Tutorial in the PNG specification [PNG10] [p. 267] if you aren't familiar with gamma issues.

In the computation, UAs displaying on a CRT may assume an ideal CRT and ignore any effects on apparent gamma caused by dithering. That means the minimal handling they need to do on current platforms is:

PC using MS-Windows
: none

Unix using X11
: none

Mac using QuickDraw
: apply gamma 1.39 [ICC32] [p. 267] (ColorSync-savvy applications may simply pass the sRGB ICC profile to ColorSync to perform correct color correction)

SGI using X
: apply the gamma value from `/etc/config/system.glGammaVal` (the default value being 1.70; applications running on Irix 6.2 or above may simply pass the sRGB ICC profile to the color management system)

NeXT using NeXTStep
: apply gamma 2.22

"Applying gamma" means that each of the three R, G and B must be converted to $R'=R^{gamma}$, $G'=G^{gamma}$, $B'=B^{gamma}$, before handing to the OS.

This may rapidly be done by building a 256-element lookup table once per browser invocation thus:

```
for i := 0 to 255 do
  raw := i / 255;
  corr := pow (raw, gamma);
  table[i] := trunc (0.5 + corr * 255.0)
end
```

which then avoids any need to do transcendental math per color attribute, far less per pixel.

3.3 Fonts

3.3.1 Glossary of font terms

DocLock™
: Bitstream's *DocLock™* technology ensures that TrueDoc PFRs can only be used with the site they are published for. A TrueDoc PFR moved to a different site or referenced from another site will not work.

Digital Signature
: Part of a trust management technology, used to provide signed assertions about a resource.

Font Caching
: *Font caching* allows for a temporary copy of fonts on the client system. They are often stored on disk with other cached items such as graphics specifically for the UA.

Font Face
: A "handle" that refers to a specific face of a font, excluding the font size (? size may be needed for non-scalable fonts)

Font Matching
: *Font matching* is a process of selecting a similar font based on using one or more attributes of the primary font. Common attribute include serif, sans serif, weight, cap height, x height, spacing, language, and posture. Font matching is dependent on the algorithm and the variety of candidate fonts.

Glyph Representation Sub-setting
: *Glyph Representation sub-setting* is the process by which unwanted glyph representations, (together with their side bearings and kerning information) are removed from a primary font to produce a smaller subset font that covers a particular document or set of documents. This is a particular win for documents that use ideographic scripts, where the glyph complement of the base font can be very large. Glyph representation sub-setting for documents

using scripts that require ligatures, such as Arabic, is difficult without knowing the ligature formation rules of the final display system.

Intellifont

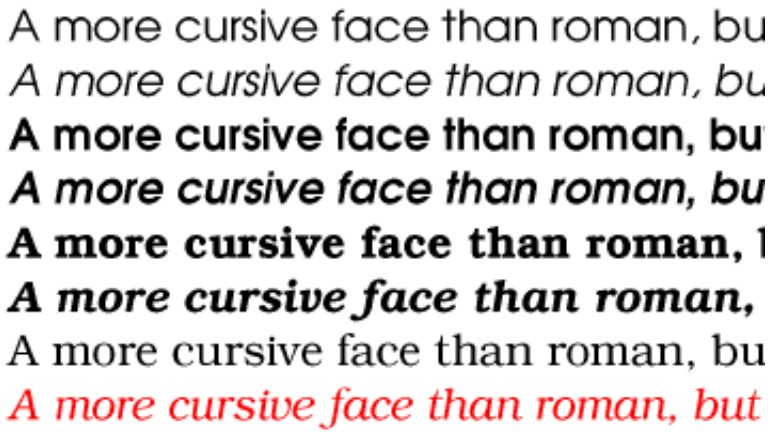
Intellifont technology was developed by Agfa and is the native format for Hewlett-Packard and other printers that use the PCL5 language. It is also the native font format on the Amiga computers.

Infinifont

A font synthesis technique which, given a Panose-1 number (and, optionally, additional font description data) can generate a faux font without extrapolating from a single master outline or interpolating between two or more outlines. See [INFINIFONT] [p. 268] .

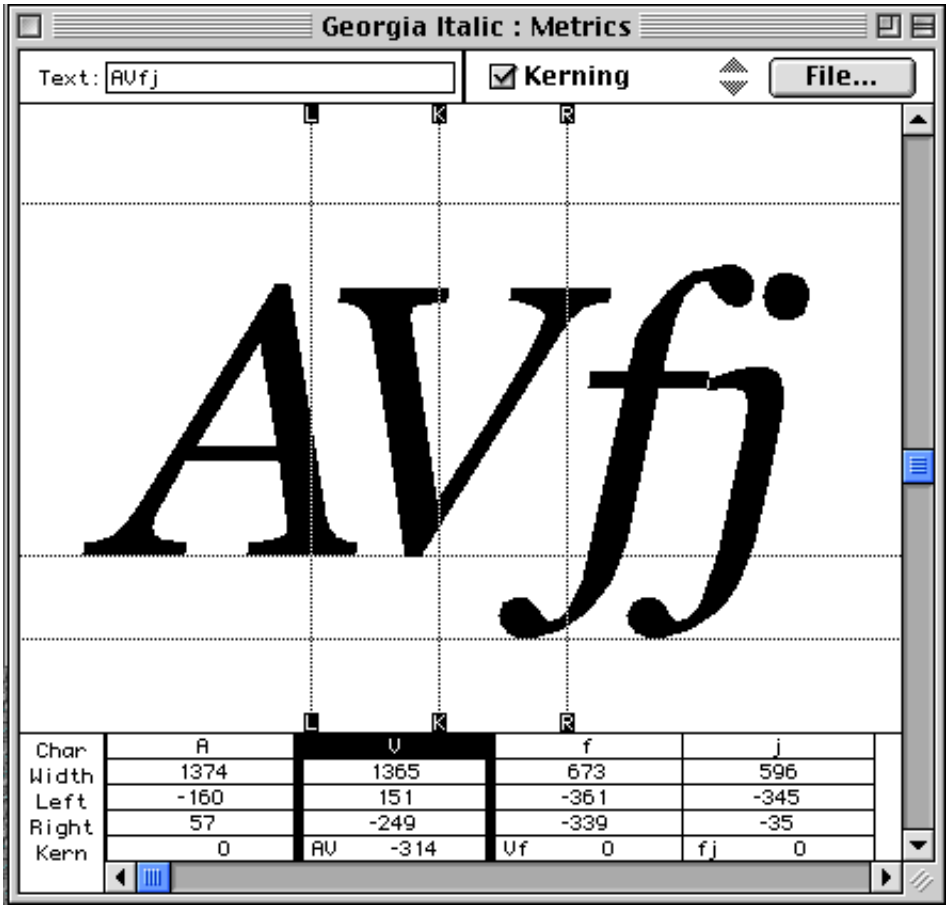
Italic

A class of letter forms for Latin scripts, that are more cursive than roman letter forms but less so than script forms. Often, a pair of fonts are designed to be used together; one is a serifed roman and one is italic. Other terms to describe this class of letter forms include cursive and, for Cyrillic scripts, kursiv. For sans-serif faces, the companion face is often a slanted or oblique variant rather than a different class of letter form.



Kerning

Altering the spacing between selected glyph representations, which would otherwise appear to be too close or too far apart, to obtain a more even typographical color.



Multiple Master Font

A *Multiple Master Font* contain two primary fonts that are used with special rendering software to provide an interpolated result. Adobe Systems provides a mechanism that allows for parameters to be used to control the output or the interpolated output font. These parameters generally describe the characteristics of an original font and the multiple master result is referred to as a "synthesized font."

Open Type

Open Type is an extension to the TrueType font format which contains additional information that extends the capabilities of the fonts to support high-quality international typography. Open Type can associate a single character with multiple glyph representations, and combinations of characters with a single glyph representation (ligature formation). Open Type includes two-dimensional information to support features for complex positioning and glyph attachment. TrueType Open contains explicit script and language information, so a text-processing application can adjust its behavior accordingly. See [OPENTYPE] [p. 269] .

Server Font

A *Server Font* is a font resource located on the web server that is referenced by the WebFont definition. The user agent may use this resource for rendering the page.

Speedo

Speedo font technology was developed by Bitstream and is the native font format on the Atari ST and Falcon computers,. It is also used by computers running X.

TrueDoc

TrueDoc technology was developed by Bitstream for the creation, transport, and imaging of platform independent scalable font objects on the web. Creation of font objects is done by the TrueDoc character shape recorder (CSR) and the rendering of the font objects is done by TrueDoc's character shape player (CSP). The technology is intended to be used on the web for viewing and printing.

TrueDoc Portable Font Resource

A *TrueDoc Portable for resource* (or **PFR**) is a platform independent scalable font object which is produce by a character shape player. Input may be either TrueType or Type 1 of any flavor on either Windows, Mac, or Unix. TrueDoc Portable Font Resources provide good compression ratios, are platform independent, and because they are not in an native font format (TrueType or Type 1) they can not be easily installed.

TrueType

TrueType is a font format developed by Apple and licensed to Microsoft. TrueType is the native operating system font format for Windows and Macintosh. TrueType contains a hierarchical set of tables and glyph representations. Characters can be hinted on a per character and point size basis yielding excellent quality at screen resolutions. TrueType fonts for Windows and Mac have few differences, though they can be different enough to prevent cross platform usage. Font foundries provide TrueType fonts for each platform and generally include a license preventing electronic manipulation to achieve cross platform transparency.

TrueType Collection

A *TrueType Collection* (or **TTC**) is an extension to the TrueType format that includes tables that allow for multiple TrueType fonts to be contained within a single TrueType font file. TrueType collection files are relatively rare at this time.

TrueType GX Fonts

TrueType GX Fonts contain extensions to the standard TrueType format that allow for mutable fonts, similar to Multiple Master fonts. There may be several mutation axis such as weight, height, and slant. The axis can be defined to obtain almost any effect. TrueType GX can also supports alternate glyph representation substitution for ligatures, contextual forms, fractions, etc. To date, TrueType GX is available only on the Mac. See [TRUETYPEGX] [p. 269] .

Type 1 font

Type 1 fonts, developed by Adobe Systems, were one of first scalable formats available. Type 1 fonts generally contain 228 characters with the glyph representations described using third degree bezier curves. Mac, Windows, and X have similar but separate formats; Adobe provides Adobe Type Manager for all three platforms. Type1c is a more recent losslessly-compressed storage form for Type 1 glyph representations.

URI Binding

A process of locking a particular font resource to a given Web site by embedding an encrypted URI or a digitally signed usage assertion into the font resource.

3.3.2 Font retrieval

There are many different font formats in use by many different platforms. To select a preferred font format, transparent content negotiation is used (see [NEGOT] [p. 269]). It is always possible to tell when a font is being dereferenced, because the URI is inside a font description. Thus, only the relevant Accept headers need be sent (not headers related to images, HTML, etc).

3.3.3 Meaning of the Panose Digits

For further details on Panose-1, see [PANOSE] [p. 269] .

Family

- Any (0)
- No Fit (1)
- **Latin Text and Display** [p. 269] *(2)*
- **Latin Script** [p. 269] (3)
- **Latin Decorative** [p. 269] (4)
- **Latin Pictorial** [p. 269] (5)

Serif Style

- Any (0)
- No Fit (1)
- Cove (2)
- Obtuse Cove (3)
- Square Cove (4)
- Obtuse Square Cove (5)
- Square (6)
- Thin (7)
- Bone (8)
- Exaggerated (9)
- Triangle (10)
- Normal Sans (11)
- Obtuse Sans (12)
- Perp Sans (13)
- Flared (14)
- Rounded (15)

Weight

- Any (0)
- No Fit (1)
- Very Light (2)[100]
- Light (3) [200]
- Thin (4) [300]
- Book (5) [400] *same as CSS1 'normal'*
- Medium (6) [500]
- Demi (7) [600]
- Bold (8) [700] *same as CSS1 'bold'*
- Heavy (9) [800]

- Black (10) [900]
- Extra Black / Nord (11) [900] *force mapping to CSS1 100-900 scale*

Proportion

- Any (0)
- No Fit (1)
- Old Style (2)
- Modern (3)
- Even Width (4)
- Expanded (5)
- Condensed (6)
- Very Expanded (7)
- Very Condensed (8)
- Monospaced (9)

Contrast

- Any (0)
- No Fit (1)
- None (2)
- Very Low (3)
- Low (4)
- Medium Low (5)
- Medium (6)
- Medium High (7)
- High (8)
- Very High (9)

Stroke Variation

- Any (0)
- No Fit (1)
- No Variation (2)
- Gradual/Diagonal (3)
- Gradual/Transitional (4)
- Gradual/Vertical (5)
- Gradual/Horizontal (6)
- Rapid/Vertical (7)
- Rapid/Horizontal (8)
- Instant/Horizontal (9)
- Instant/Vertical (10)

Arm Style

- Any (0)
- No Fit (1)
- Straight Arms/Horizontal (2)
- Straight Arms/Wedge (3)
- Straight Arms/Vertical (4)
- Straight Arms/Single Serif (5)
- Straight Arms/Double Serif (6)
- Non-Straight Arms/Horizontal (7)
- Non-Straight Arms/Wedge (8)
- Non-Straight Arms/Vertical 90)

- Non-Straight Arms/Single Serif (10)
- Non-Straight Arms/Double Serif (11)

Letterform

- Any (0)
- No Fit (1)
- Normal/Contact (2)
- Normal/Weighted (3)
- Normal/Boxed (4)
- Normal/Flattened (5)
- Normal/Rounded (6)
- Normal/Off Center (7)
- Normal/Square (8)
- Oblique/Contact (9)
- Oblique/Weighted (10)
- Oblique/Boxed (11)
- Oblique/Flattened (12)
- Oblique/Rounded (13)
- Oblique/Off Center (14)
- Oblique/Square (15)

Midline

- Any (0)
- No Fit (1)
- Standard/Trimmed (2)
- Standard/Pointed (3)
- Standard/Serifed (4)
- High/Trimmed (5)
- High/Pointed (6)
- High/Serifed (7)
- Constant/Trimmed (8)
- Constant/Pointed (9)
- Constant/Serifed (10)
- Low/Trimmed (11)
- Low/Pointed (12)
- Low/Serifed (13)

XHeight

- Any (0)
- No Fit (1)
- Constant/Small (2)
- Constant/Standard (3)
- Constant/Large (4)
- Ducking/Small (5)
- Ducking/Standard (6)
- Ducking/Large (7)

3.3.4 Deducing Unicode Ranges for TrueType

This information is available in the font by looking at the 'ulUnicodeRange' bits in the 'OS/2' table (if it has one), which holds a bitfield representation of the set. This table is defined in revision 1.66 of the TrueType specification, from Microsoft. Considering this information as a set, each element corresponds to a Unicode 1.1 character block, and the presence of that element in the set indicates that the font has one or more glyph representations to represent at least one character in that block. The set has 128 elements as described below. The order generally follows that in the Unicode 1.1 standard. This table may be used to convert the information in a TrueType font into a CSS 'unicode-range' descriptor.

Block	Add	Block name	Unicode range
0	1	Basic Latin	U+0-7F
1	2	Latin-1 Supplement	U+80-FF
2	4	Latin-1 Extended-A	U+100-17F
3	8	Latin Extended-B	U+180-24F
4	1	IPA Extensions	U+250-2AF
5	2	Spacing Modifier Letters	U+2B0-2FF
6	4	Combining Diacritical Marks	U+300-36F
7	8	Greek	U+370-3CF
8	1	*Greek Symbols and Coptic*	U+3D0-3EF
9	2	Cyrillic	U+400-4FF
10	4	Armenian	U+530-58F
11	8	Hebrew	U+590-5FF
12	1	*Hebrew Extended-A* *Hebrew Extended-B*	?? what ranges ??
13	2	Arabic	U+600-69F
14	4	*Arabic Extended*	U+670-6FF
15	8	Devanagari	U+900-97F
16	1	Bengali	U+980-9FF
17	2	Gurmukhi	U+A00-A7F
18	4	Gujarati	U+A80-AFF

19	8	Oriya	U+B00-B7F
20	1	Tamil	U+B80-BFF
21	2	Telugu	U+C00-C7F
22	4	Kannada	U+C80-CFF
23	8	Malayalam	U+D00-D7F
24	1	Thai	U+E00-E7F
25	2	Lao	U+E80-EFF
26	4	Georgian	U+10A0-10EF
27	8	*Georgian Extended*	U+10F0-10FF ??
28	1	Hangul Jamo	U+1100-11FF
29	2	Latin Extended Additional	-
30	4	Greek Extended	U+1F00-1FFF
31	8	General Punctuation	U+2000-206F
32	1	Superscripts and Subscripts	-
33	2	Currency Symbols	U+20A0-20CF
34	4	Combining Marks for Symbols	U+20D0-20FF
35	8	Letterlike Symbols	U+2100-214F
36	1	Number Forms	U+2150-218F
37	2	Arrows	U+2190-21FF
38	4	Mathematical Operators	U+2200-22FF
39	8	Miscellaneous Technical	U+2300-23FF
40	1	Control Pictures	U+2400-243F
41	2	Optical Character Recognition	U+2440-245F
42	4	Enclosed Alphanumerics	U+2460-24FF
43	8	Box Drawing	U+2500-257F
44	1	Block Elements	U+2580-259F
45	2	Geometric Shapes	U+25A0-25FF
46	4	Miscellaneous Symbols	U+2600-26FF

47	8	Dingbats	U+2700-27BF
48	1	CJK Symbols and Punctuation	U+3000-303F
49	2	Hiragana	U+3040-309F
50	4	Katakana	U+30A0-30FF
51	8	Bopomofo	U+3100-312F
52	1	Hangul Compatibility Jamo	U+3130-318F
53	2	CJK Miscellaneous	??
54	4	Enclosed CJK Letters and Months	U+3200-32FF
55	8	CJK compatibility	U+3300-33FF
56	1	Hangul	U+AC00-D7FF
59	8	CJK Unified Ideographs	U+4E00-9FFF
60	1	Private Use Area	U+E000-F8FF
61	2	CJK Compatibility Ideographs	U+F900-FAFF
62	4	Alphabetic Presentation Forms	U+FB00-FB4F
63	8	Arabic Presentation Forms-A	U+FB50-FDFF
64	1	Combining Half Marks	U+FE20-FE2F
65	2	CJK compatibility Forms	U+FE30-FE4F
66	4	Small Form Variants	U+FE50-FE6F
67	8	Arabic Presentation Forms-B	U+FE70-FEFF
68	1	Halfwidth and Fullwidth Forms	U+FF00-FFEF
69	2	Specials	U+FFF0-FFFD

The TrueType bitfield system has the problem that it is tied to Unicode 1.1 and is unable to cope with Unicode expansion - it is unable to represent Tibetan for example.

Appendix D: The grammar of CSS2

This appendix is normative.

The grammar below defines the syntax of CSS2. It is in some sense, however, a superset of CSS2 as this specification imposes additional semantic constraints not expressed in this grammar. A conforming UA must also adhere to the forward-compatible parsing rules [p. 29] , the property and value notation [p. 14] , and the unit notation. In addition, the document language may impose restrictions, e.g. HTML imposes restrictions on the possible values of the "class" attribute.

4.1 Grammar

The grammar below is LL(1) (but note that most UA's should not use it directly, since it doesn't express the parsing conventions [p. 35] , only the CSS2 syntax). The format of the productions is optimized for human consumption and some shorthand notation beyond [YACC] [p. 268] is used:

- *****: 0 or more
- **+**: 1 or more
- **?**: 0 or 1
- **|**: separates alternatives
- **[]**: grouping

The productions are:

```
stylesheet
  : [S|CDO|CDC]* [ import [S|CDO|CDC]* ]*
    [ [ ruleset | media | page | font_face ] [S|CDO|CDC]* ]*
  ;
import
  : IMPORT_SYM S*
    [STRING|URI] S* [ medium [ ',' S* medium]* ]? ';' S*
  ;
media
  : MEDIA_SYM S* medium [ ',' S* medium ]* '{' S* ruleset* '}' S*
  ;
medium
  : IDENT S*
  ;
page
  : PAGE_SYM S* pseudo_page?
    '{' S* declaration [ ';' S* declaration ]* '}' S*
  ;
pseudo_page
  : ':' IDENT S*
  ;
font_face
  : FONT_FACE_SYM S*
    '{' S* declaration [ ';' S* declaration ]* '}' S*
  ;
operator
  : '/' S* | ',' S* | /* empty */
  ;
combinator
  : '+' S* | '>' S* | /* empty */
```

```
  ;
unary_operator
  : '-' | '+'
  ;
property
  : IDENT S*
  ;
ruleset
  : selector [ ',' S* selector ]*
    '{' S* declaration [ ';' S* declaration ]* '}' S*
  ;
selector
  : simple_selector [ combinator simple_selector ]*
  ;
/*
 * simple selector cannot start with attrib selector
 */
simple_selector
  : element_name [ HASH | class | attrib | pseudo ]* S*
  | HASH [ class | attrib | pseudo ]* S*
  | pseudo [ HASH | class | attrib | pseudo ]* S*
  | class [ HASH | class | attrib | pseudo ]* S*
  ;
class
  : '.' IDENT
  ;
element_name
  : IDENT | '*'
  ;
attrib
  : '[' S* IDENT S* [ [ '=' | INCLUDES ] S* [ IDENT | STRING ] S* ]? ']'
  ;
pseudo
  : ':' IDENT
  ;
declaration
  : property ':' S* expr prio?
  | /* empty */
  ;
prio
  : IMPORTANT_SYM S*
  ;
expr
  : term [ operator term ]*
  ;
term
  : unary_operator?
    [ NUMBER S* | PERCENTAGE S* | LENGTH S* | EMS S* | EXS S* | ANGLE S* |
      TIME S* | FREQ S* | function ]
  | STRING S* | IDENT S* | URI S* | RGB S* | UNICODERANGE S* | hexcolor
  ;
function
  : FUNCTION S* expr ')' S*
  ;
/*
 * There is a constraint on the color that it must
 * have either 3 or 6 hex-digits (i.e., [0-9a-fA-F])
 * after the "#"; e.g., "#000" is OK, but "#abcd" is not.
 */
hexcolor
  : HASH S*
  ;
```

4.2 Lexical scanner

The following is the tokenizer, written in flex [FLEX] [p. 267] notation. The tokenizer is case-insensitive.

The two occurrences of "\377" represent the highest character number that current versions of Flex can deal with (decimal 255). They should be read as "\4177777" (decimal 1114111), which is the highest possible code point in Unicode/ISO-10646.

```
%option case-insensitive

h               [0-9a-f]
nonascii        [\200-\377]
unicode         \\{h}{1,6}
escape          {unicode}|\\[ -~\200-\377]
nmstart         [a-z]|{nonascii}|{escape}
nmchar          [a-z0-9-]|{nonascii}|{escape}
string1         \"([\t\n !#$%&(-~]|\’|{nonascii}|{escape})*\"
string2         \’([\t\n !#$%&(-~]|\"|{nonascii}|{escape})*\’

ident           {nmstart}{nmchar}*
name            {nmchar}+
num             [0-9]+|[0-9]*"."[0-9]+
string          {string1}|{string2}
url             ([!#$%&*-~]|{nonascii}|{escape})*
w               [ \t\r\n\f]*

range           {h}(\?{0,5}|{h}(\?{0,4}|{h}(\?{0,3}|{h}(\?{0,2}|{h}(\??|{h})))))

%%

[ \t\r\n\f]+            {return S;}

\/\*[^*]*\*+([^/][^*]*\*+)*\/   /* ignore comments */

"<!--"                  {return CDO;}
"-->"                   {return CDC;}
"~="                    {return INCLUDES;}

{string}                {return STRING;}

{ident}                 {return IDENT;}

"#"{name}               {return HASH;}

"@import"               {return IMPORT_SYM;}
"@page"                 {return PAGE_SYM;}
"@media"                {return MEDIA_SYM;}
"@font-face"            {return FONT_FACE_SYM;}
"@"{ident}              {return ATKEYWORD;}

"!important"            {return IMPORTANT_SYM;}

{num}em                 {return EMS;}
{num}ex                 {return EXS;}
{num}px                 {return LENGTH;}
{num}cm                 {return LENGTH;}
{num}mm                 {return LENGTH;}
{num}in                 {return LENGTH;}
{num}pt                 {return LENGTH;}
{num}pc                 {return LENGTH;}
{num}deg                {return ANGLE;}
{num}rad                {return ANGLE;}
{num}grad               {return ANGLE;}
```

```
{num}ms                 {return TIME;}
{num}s                  {return TIME;}
{num}Hz                 {return FREQ;}
{num}kHz                {return FREQ;}
{num}{ident}            {return DIMEN;}
{num}%                  {return PERCENTAGE;}
{num}                   {return NUMBER;}

"url("{w}{string}{w}")" {return URI;}
"url("{w}{url}{w}")"    {return URI;}
{ident}"("              {return FUNCTION;}

U\+{range}              {return UNICODERANGE;}
U\+{h}{1,6}-{h}{1,6}    {return UNICODERANGE;}

.                       {return *yytext;}
```

References

Contents

1. Normative references [p. 267]
2. Informative references [p. 268]

1.1 Normative references

[COLORIMETRY]
"Colorimetry, Second Edition", CIE Publication 15.2-1986, ISBN 3-900-734-00-3.
Available at
http://www.hike.te.chiba-u.ac.jp/ikeda/CIE/publ/abst/15-2-86.html.

[CSS1]
"Cascading Style Sheets, level 1", H. W. Lie and B. Bos, 17 December 1996.
Available at http://www.w3.org/TR/REC-CSS1-961217.html

[FLEX]
"Flex: The Lexical Scanner Generator", Version 2.3.7, ISBN 1882114213.

[ICC32]
"ICC Profile Format Specification, version 3.2", 1995.
Available at ftp://sgigate.sgi.com/pub/icc/ICC32.pdf.

[ISO8879]
ISO 8879:1986 "Information Processing -- Text and Office Systems -- Standard Generalized Markup Language (SGML)", ISO 8879:1986.
For the list of SGML entities, consult ftp://ftp.ifi.uio.no/pub/SGML/ENTITIES/.

[ISO10646]
"Information Technology - Universal Multiple- Octet Coded Character Set (UCS) - Part 1: Architecture and Basic Multilingual Plane", ISO/IEC 10646-1:1993. The current specification also takes into consideration the first five amendments to ISO/IEC 10646-1:1993. Useful roadmap of the BMP and roadmap of plane 1 documents show which scripts sit at which numeric ranges.

[PNG10]
"PNG (Portable Network Graphics) Specification, Version 1.0 specification", T. Boutell ed., 1 October 1996.
Available at http://www.w3.org/pub/WWW/TR/REC-png-multi.html.

[RFC1738]
"Uniform Resource Locators", T. Berners-Lee, L. Masinter, and M. McCahill, December 1994.
Available at ftp://ds.internic.net/rfc/rfc1738.txt.

[RFC1808]
"Relative Uniform Resource Locators", R. Fielding, June 1995.
Available at ftp://ds.internic.net/rfc/rfc1808.txt.

[RFC2070]
"Internationalization of the HyperText Markup Language", F. Yergeau, G. Nicol, G. Adams, and M. Dürst, January 1997.
Available at ftp://ds.internic.net/rfc/rfc2070.txt.

[RFC2119]
"Key words for use in RFCs to Indicate Requirement Levels", S. Bradner, March 1997.
Available at ftp://ds.internic.net/rfc/rfc2119.txt.
[SRGB]
"Proposal for a Standard Color Space for the Internet - sRGB", M Anderson, R Motta, S Chandrasekar, M Stokes.
Available at http://www.w3.org/Graphics/Color/sRGB.html.
[UNICODE]
The latest version of Unicode. For more information, consult the Unicode Consortium's home page at http://www.unicode.org/
[URI]
"Uniform Resource Identifiers (URI): Generic Syntax and Semantics", T. Berners-Lee, R. Fielding, L. Masinter, 18 November 1997.
Available at http://www.ics.uci.edu/pub/ietf/uri/draft-fielding-uri-syntax-01.txt. This is a work in progress that is expected to update [RFC1738] [p. 267] and [RFC1808] [p. 267] .
[XML]
Please consult http://www.w3.org/XML/ for information about the XML specification.
[YACC]
"YACC - Yet another compiler compiler", S. C. Johnson, Technical Report, Murray Hill, 1975.

1.2 Informative references

[DOM]
"Document Object Model Specification", L. Wood, A. Le Hors, 9 October 1997.
Available at http://www.w3.org/TR/WD-DOM/
[ISO10179]
ISO/IEC 10179:1996 "Information technology -- Processing languages -- Document Style Semantics and Specification Language (DSSSL)"
Available at http://occam.sjf.novell.com:8080/dsssl/dsssl96
[GAMMA]
"Gamma correction on the Macintosh Platform", C. A. Poynton.
Available at
ftp://ftp.inforamp.net/pub/users/poynton/doc/Mac/Mac_gamma.pdf.
[HTML32]
"HTML 3.2 Reference Specification", Dave Raggett, 14 January 1997.
Available at http://www.w3.org/TR/REC-html32.html
[HTML40]
"HTML 4.0 Specification (Working Draft)", D. Raggett, A. Le Hors, I. Jacobs, 8 July 1997.
Available at http://www.w3.org/TR/REC-html40/. The Recommendation defines three document type definitions: Strict, Transitional, and Frameset, all reachable from the Recommendation.
[INFINIFONT]
See http://www.fonts.com/hp/infinifont/moredet.html.

[ISO9899]
ISO/IEC 9899:1990 Programming languages -- C.
[MONOTYPE]
See http://www.monotype.com/html/oem/uni_scrmod.html
[NEGOT]
"Transparent Content Negotiation in HTTP", K. Holtman, A. Mutz, 9 March, 1997.
Available at
http://gewis.win.tue.nl/~koen/conneg/draft-ietf-http-negotiation-01.html
[OPENTYPE]
See http://www.microsoft.com/OpenType/OTSpec/tablist.htm.
[PANOSE]
For information about PANOSE classification metrics, consult http://www.fonts.com/hp/panose/greybook and the following chapters: Latin Text, Latin Script, Latin Decorative, and Latin Pictorial.
Panose numbers for some fonts are available online and may be queried.
[PANOSE2]
See http://www.w3.org/Fonts/Panose/pan2.html Panose-2 is not limited to Latin typefaces.
[RFC1630]
"Universal Resource Identifiers in WWW: A Unifying Syntax for the Expression of Names and Addresses of Objects on the Network as used in the World-Wide Web", T. Berners-Lee, June 1994.
Available at http://ds.internic.net/rfc/rfc1630.txt.
[RFC1866]
"HyperText Markup Language 2.0", T. Berners-Lee and D. Connolly, November 1995.
Available at ftp://ds.internic.net/rfc/rfc1866.txt.
[RFC1942]
"HTML Tables", Dave Raggett, May 1996.
Available at ftp://ds.internic.net/rfc/rfc1942.txt.
[TRUETYPEGX]
See http://fonts.apple.com/TTRefMan/index.html for details about TrueType GX from Apple Computer, including descriptions of the added tables and font quality specifications
[W3CSTYLE]
W3C resource page on web style sheets.
Examine at http://www.w3.org/pub/WWW/Style

Property index

Name	Values	Initial value	Applies to (Default: all)	Inherited? (Default: no)	Percentages (Default: N/A)	Media groups
'azimuth' [p. 237]	<angle> | [[left-side | far-left | left | center-left | center | center-right | right | far-right | right-side] || behind] | leftwards | rightwards | inherit	center		yes		aural [p. 65]
'background' [p. 145]	['background-color' || 'background-image' || 'background-repeat' || 'background-attachment' || 'background-position'] | inherit	XX			allowed on 'background-position'	visual [p. 65]
'background-attachment' [p. 144]	scroll | fixed | inherit	scroll				visual [p. 65]
'background-color' [p. 142]	<color> | transparent | inherit	transparent				visual [p. 65]
'background-image' [p. 142]	<uri> | none | inherit	none				visual [p. 65]
'background-position' [p. 144]	[[<percentage> | <length>]{1,2} | [top | center | bottom] || [left | center | right]] | inherit	0% 0%	block-level and replaced elements		refer to the size of the element itself	visual [p. 65]
'background-repeat' [p. 143]	repeat | repeat-x | repeat-y | no-repeat | inherit	repeat				visual [p. 65]
'border' [p. 111]	['border-width' || 'border-style' || <color>] | inherit	see individual properties				visual [p. 65]
'border-bottom' [p. 110]	['border-bottom-width' || 'border-style' || <color>] | inherit	see individual properties				visual [p. 65]
'border-bottom-color' [p. 106]	<color> | inherit	the value of the 'color' property				visual [p. 65]
'border-bottom-style' [p. 108]	<border-style> | inherit	none				visual [p. 65]
'border-bottom-width' [p. 105]	<border-width> | inherit	medium				visual [p. 65]
'border-collapse' [p. 207]	collapse | separate	separate	'table' elements			visual [p. 65]
'border-color' [p. 107]	<color>{1,4} | inherit	see individual properties				visual [p. 65]
'border-left' [p. 110]	['border-left-width' || 'border-style' || <color>] | inherit	see individual properties				visual [p. 65]
'border-left-color' [p. 107]	<color> | inherit	the value of the 'color' property				visual [p. 65]
'border-left-style' [p. 109]	<border-style> | inherit	none				visual [p. 65]
'border-left-width' [p. 105]	<border-width> | inherit	medium				visual [p. 65]

'border-right' [p. 110]	['border-right-width' || 'border-style' || <color>] | inherit	see individual properties				visual [p. 65]
'border-right-color' [p. 106]	<color> | inherit	the value of the 'color' property				visual [p. 65]
'border-right-style' [p. 108]	<border-style> | inherit	none				visual [p. 65]
'border-right-width' [p. 105]	<border-width> | inherit	medium				visual [p. 65]
'border-style' [p. 109]	<border-style>{1,4} | inherit	see individual properties				visual [p. 65]
'border-top' [p. 109]	['border-top-width' || 'border-style' || <color>] | inherit	see individual properties				visual [p. 65]
'border-top-color' [p. 106]	<color> | inherit	the value of the 'color' property				visual [p. 65]
'border-top-style' [p. 108]	<border-style> | inherit	none				visual [p. 65]
'border-top-width' [p. 104]	<border-width> | inherit	medium				visual [p. 65]
'border-width' [p. 105]	<border-width>{1,4} | inherit	see individual properties				visual [p. 65]
'bottom' [p. 78]	<length> | <percentage> | auto | inherit	auto			refer to height of containing block	visual [p. 65]
'caption-side' [p. 222]	top | bottom | inherit	top	caption elements	yes		visual [p. 65]
'cell-spacing' [p. 217]	none | <length> <length>? | inherit	none	table	yes		visual [p. 65]
'clear' [p. 84]	none | left | right | both | inherit	none	block-level elements			visual [p. 65]
'clip' [p. 125]	<shape> | auto | inherit	auto	elements with the 'position' property set to 'absolute'			visual [p. 65]
'color' [p. 141]	<color> | inherit	depends on user agent		yes		visual [p. 65]
'column-span' [p. 204]	<integer> | inherit	1	cell, column, and column-group elements			visual [p. 65]
'content' [p. 130]	[<string> | <uri> | <counter>]+ | inherit	empty string	:before and :after			all [p. 65]
'cue' [p. 235]	['cue-before' || 'cue-after'] | inherit	XX				aural [p. 65]
'cue-after' [p. 235]	<uri> | none | inherit	none				aural [p. 65]
'cue-before' [p. 234]	<uri> | none | inherit	none				aural [p. 65]
'cursor' [p. 227]	[[auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | se-resize | sw-resize | s-resize | w-resize| text | wait | help] || <uri>?] | inherit	auto		yes		visual [p. 65]

’direction’ [p. 81]	ltr | rtl | ltr-override | rtl-override | inherit	ltr		yes		visual [p. 65]
’display’ [p. 68]	inline | block | list-item | none | run-in | compact | table | inline-table| table-row-group | table-column-group | table-header-group | table-footer-group | table-row | table-cell | table-caption | inherit	inline				visual [p. 65]
’elevation’ [p. 238]	<angle> | below | level | above | higher | lower | inherit	level		yes		aural [p. 65]
’float’ [p. 82]	left | right | none | inherit	none	elements that are not positioned absolutely			visual [p. 65]
’font’ [p. 158]	[[’font-style’ || ’font-variant’ || ’font-weight’]? ’font-size’ [/ ’line-height’]? ’font-family’] | caption | icon | menu | messagebox | smallcaption | statusbar | inherit	see individual properties		yes	allowed on ’font-size’ and ’line-height’	visual [p. 65]
’font-family’ [p. 150]	[[[<family-name> | <generic-family>],]* [<family-name> | <generic-family>]] | inherit	depends on user agent		yes		visual [p. 65]
’font-size’ [p. 154]	<absolute-size> | <relative-size> | <length> | <percentage> | inherit	medium		yes	relative to parent element’s font size	visual [p. 65]
’font-size-adjust’ [p. 155]	z | none | inherit	none		yes, but not adjusted values	relative to parent element’s font size	visual [p. 65]
’font-style’ [p. 151]	normal | italic | oblique | inherit	normal		yes		visual [p. 65]
’font-variant’ [p. 152]	normal | small-caps | inherit	normal		yes		visual [p. 65]
’font-weight’ [p. 152]	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit	normal		yes		visual [p. 65]
’height’ [p. 115]	<length> | <percentage> | auto | inherit	auto	all elements but non-replaced inline elements, table columns and column groups		see prose	visual [p. 65]
’left’ [p. 78]	<length> | <percentage> | auto | inherit	auto			refer to width of containing block	visual [p. 65]
’letter-spacing’ [p. 189]	normal | <length> | auto | inherit	normal		yes		visual [p. 65]
’line-height’ [p. 119]	normal | <number> | <length> | <percentage> | inherit	normal		yes	relative to the font size of the element itself	visual [p. 65]
’list-style’ [p. 198]	[’list-style-type’ || ’list-style-position’ || ’list-style-image’] | inherit	XX	elements with the ’display’ property set to ’list-item’	yes		visual [p. 65]

'list-style-image' [p. 196]	<uri> | none | inherit	none	elements with the 'display' property set to 'list-item'	yes		visual [p. 65]
'list-style-position' [p. 197]	inside | outside | inherit	outside	elements with the 'display' property set to 'list-item'	yes		visual [p. 65]
'list-style-type' [p. 195]	disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none | inherit	disc	elements with the 'display' property set to 'list-item'	yes		visual [p. 65]
'margin' [p. 101]	<margin-width>{1,4} | inherit	XX			refer to width of containing block	visual [p. 65]
'margin-bottom' [p. 100]	<margin-width> | inherit	0			refer to width of containing block	visual [p. 65]
'margin-left' [p. 101]	<margin-width> | inherit	0			refer to width of containing block	visual [p. 65]
'margin-right' [p. 100]	<margin-width> | inherit	0			refer to width of containing block	visual [p. 65]
'margin-top' [p. 100]	<margin-width> | inherit	0			refer to width of containing block	visual [p. 65]
'marks' [p. 135]	crop || cross | none | inherit	none	page context	N/A		visual [p. 65] , paged [p. 65]
'max-height' [p. 117]	<length> | <percentage> | inherit	100%	all		refer to parent's height	visual [p. 65]
'max-width' [p. 114]	<length> | <percentage> | inherit	100%	all		refer to parent's width	visual [p. 65]
'min-height' [p. 117]	<length> | <percentage> | inherit	0	all		refer to parent's height	visual [p. 65]
'min-width' [p. 114]	<length> | <percentage> | inherit	0	all		refer to parent's width	visual [p. 65]
'orphans' [p. 138]	<integer> | inherit	2	block-level elements	yes		visual [p. 65] , paged [p. 65]
'overflow' [p. 123]	visible | hidden | scroll | auto | inherit	visible	elements with the 'position' property set to 'absolute'			visual [p. 65]
'padding' [p. 103]	<padding-width>{1,4} | inherit	XX			refer to width of containing block	visual [p. 65]
'padding-bottom' [p. 103]	<padding-width> | inherit	0			refer to width of containing block	visual [p. 65]
'padding-left' [p. 103]	<padding-width> | inherit	0			refer to width of containing block	visual [p. 65]
'padding-right' [p. 102]	<padding-width> | inherit	0			refer to width of containing block	visual [p. 65]
'padding-top' [p. 102]	<padding-width> | inherit	0			refer to width of containing block	visual [p. 65]
'page-break-after' [p. 137]	auto | always | avoid | left | right | inherit	auto	block-level and inline elements except those in tables			visual [p. 65] , paged [p. 65]
'page-break-before' [p. 137]	auto | always | avoid | left | right | inherit	auto	block-level and inline elements except those in tables			visual [p. 65] , paged [p. 65]

’pause’ [p. 234]	[[<time> | <percentage>]{1,2}] | inherit	depends on user agent			see descriptions of ’pause-before’ and ’pause-after’	aural [p. 65]
’pause-after’ [p. 233]	<time> | <percentage> | inherit	depends on user agent			see prose	aural [p. 65]
’pause-before’ [p. 233]	<time> | <percentage> | inherit	depends on user agent			see prose	aural [p. 65]
’pitch’ [p. 240]	<frequency> | x-low | low | medium | high | x-high | inherit	medium		yes		aural [p. 65]
’pitch-range’ [p. 241]	<number> | inherit	50		yes		aural [p. 65]
’play-during’ [p. 236]	<uri> | mix? repeat? | auto | none | inherit	auto				aural [p. 65]
’position’ [p. 76]	normal | relative | absolute | fixed | inherit	normal	elements that generate absolutely positioned and floated boxes			visual [p. 65]
’richness’ [p. 242]	<number> | inherit	50		yes		aural [p. 65]
’right’ [p. 77]	<length> | <percentage> | auto | inherit	auto			refer to width of containing block	visual [p. 65]
’row-span’ [p. 204]	<integer> | inherit	1	cell elements			visual [p. 65]
’size’ [p. 133]	<length>{1,2} | auto | portrait | landscape | inherit	auto	page context	N/A		visual [p. 65] , paged [p. 65]
’speak’ [p. 232]	normal | none | spell-out | inherit	normal		yes		aural [p. 65]
’speak-date’ [p. 242]	mdy | dmy | ymd | inherit	depends on user agent		yes		aural [p. 65]
’speak-header’ [p. 222]	once | always | inherit	once	header cells	yes		visual [p. 65]
’speak-numeral’ [p. 243]	digits | continuous | none | inherit	none		yes		aural [p. 65]
’speak-punctuation’ [p. 242]	code | none | inherit	none		yes		aural [p. 65]
’speak-time’ [p. 243]	24 | 12 | none | inherit	none		yes		aural [p. 65]
’speech-rate’ [p. 239]	<number> | x-slow | slow | medium | fast | x-fast | faster | slower | inherit	medium		yes		aural [p. 65]
’stress’ [p. 241]	<number> | inherit	50		yes		aural [p. 65]
’table-layout’ [p. 210]	auto | fixed	auto	’table’ elements			visual [p. 65]
’text-align’ [p. 186]	left | right | center | justify | inherit	depends on user agent	block-level elements	yes		visual [p. 65]
’text-decoration’ [p. 186]	none | [underline || overline || line-through || blink] | inherit	none		no (see prose)		visual [p. 65]
’text-indent’ [p. 185]	<length> | <percentage> | inherit	0	block-level elements	yes	refer to parent element’s width	visual [p. 65]

'text-shadow' [p. 187]	none | [<color> || <length> <length> <length>? ,]* [<color> || <length> <length> <length>?] | inherit	none	all	no (see prose)		visual [p. 65]
'text-transform' [p. 190]	capitalize | uppercase | lowercase | none | inherit	none		yes		visual [p. 65]
'top' [p. 77]	<length> | <percentage> | auto | inherit	auto			refer to height of containing block	visual [p. 65]
'vertical-align' [p. 120]	baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage> | <length> | inherit	baseline	inline elements		refer to the 'line-height' of the element itself	visual [p. 65]
'visibility' [p. 127]	inherit | visible | hidden	inherit		if value is 'inherit'		visual [p. 65]
'voice-family' [p. 240]	[[<specific-voice> | <generic-voice>],]* [<specific-voice> | <generic-voice>] | inherit	depends on user agent		yes		aural [p. 65]
'volume' [p. 231]	<number> | <percentage> | silent | x-soft | soft | medium | loud | x-loud | inherit	medium		yes	relative to inherited value	aural [p. 65]
'white-space' [p. 191]	normal | pre | nowrap | inherit	normal	block-level elements	yes		visual [p. 65]
'widows' [p. 138]	<integer> | inherit	2	block-level elements	yes		visual [p. 65] , paged [p. 65]
'width' [p. 112]	<length> | <percentage> | auto | inherit	auto	all elements but non-replaced inline elements, table rows and row groups		refer to width of containing block	visual [p. 65]
'word-spacing' [p. 189]	normal | <length> | inherit	normal		yes		visual [p. 65]
'z-index' [p. 96]	auto | <integer> | inherit	auto	elements that generate absolutely and relatively positioned boxes			visual [p. 65]

Descriptor index

Name	Values	Initial value
’ascent’ [p. 171]	<number>	undefined
’baseline’ [p. 172]	<number>	0
’cap-height’ [p. 170]	<number>	undefined
’centerline’ [p. 173]	<number>	undefined
’definition-src’ [p. 172]	<uri>	undefined
’descent’ [p. 171]	<number>	undefined
’font-family’ [p. 165]	[<family-name> | <generic-family>] [, [<family-name> | <generic-family>]]*	depends on user agent
’font-size’ [p. 166]	all | [<length> [,[<length>]]*]	all
’font-style’ [p. 165]	[normal | italic | oblique] [, [normal | italic | oblique]]*	normal
’font-variant’ [p. 165]	[normal | small-caps] [,[normal | small-caps]]*	normal
’font-weight’ [p. 166]	all | [normal | bold | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900] [, [normal | bold | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900]]*	normal
’mathline’ [p. 173]	<number>	undefined
’panose-1’ [p. 169]	[<number>]{10}	0 0 0 0 0 0 0 0 0 0
’slope’ [p. 170]	<number>	0
’src’ [p. 168]	[<uri> [format [,format]*]? | <font-face-name>] [, <uri> [format [,format]*]?] | <font-face-name>]*	undefined
’stemh’ [p. 170]	<number>	undefined
’stemv’ [p. 170]	<number>	undefined
’topline’ [p. 173]	<number>	undefined

’unicode-range’ [p. 166]	<urange>+	U+0-7FFFFFFF
’units-per-em’ [p. 168]	<number>	undefined
’widths’ [p. 171]	[<urange>]? [<number>]+ [,[<urange>]? <number>]+]	undefined
’x-height’ [p. 170]	<number>	undefined

Index

2½D borders 216

:active **54**
:after **129**, **54**
:before **129**, **54**
:first **136**
:first-letter **52**, 54
:first-line 51
:hover **54**
:left **135**
:link **54**
:right **135**
:visited **54**

@-rules 32
@font-face **163**, 163, 164, 168, 180, 181
@import **63**, **64**
@media **63**, 64, 64
@page **132**

absolute length **37**
<absolute-size>
    definition of **154**
’active’ pseudo-class **54**
actual value **58**
adorned font name **169**
’all’ media group 65
alternate text 26
ancestor **26**
<angle> 237, 238
    definition of 40
anonymous elements **79**
’ascent’ (descriptor) **171**
at-rules 32
attr() function **48**
audio icon **231**
’aural’ media group 65
’azimuth’ **237**

’background’ **146**
’background-attachment’ **144**
’background-color’ **142**
’background-image’ **143**
’background-position’ **144**
’background-repeat’ **143**
backslash escapes 32
’baseline’ (descriptor) **172**

’bitmap’ media group 65
block 33
’block’, definition of 69
block-level element **69**
border
    of a box 71
border edge 72
’border’ **111**
’border-bottom’ **110**
’border-bottom-color’ **106**
’border-bottom-style’ **108**
’border-bottom-width’ **105**
’border-collapse’ **207**
’border-color’ **107**
’border-left’ **110**
’border-left-color’ **107**
’border-left-style’ **109**
’border-left-width’ **105**
’border-right’ **110**
’border-right-color’ **106**
’border-right-style’ **108**
’border-right-width’ **105**
<border-style> 209
<border-style>, definition of 107
’border-style’ **109**
’border-top’ **109**
’border-top-color’ **106**
’border-top-style’ **108**
’border-top-width’ **104**
<border-width>
    definition of 104
’border-width’ **105**
<bottom>
    definition of 126
’bottom’ **78**
box
    box height 73
    box width 73
    overflow **123**
box border 71
box content 71
box margin 71
box padding 71
box:content height 72
box:content width 72

canvas 22
’cap-height’ (descriptor) **170**
’caption-side’ **222**

cascade **59**
case sensitivity 31
’cell-spacing’ **217**
’centerline’ (descriptor) **173**
child **26**
child selector **46**
’clear’ **84**
’clip’ **125**
clipping region **125**
collapse 211
color 264
<color> 142
    definition of 39
’color’ **141**
’column-span’ **204**
comments 35
compact box 70
’compact’, definition of 70
computed value **57**
conditional import **64**
conformance **27**, 186, 189, 190, 191, 192, 108, 70
containing block **111**, 67
    initial 111
content
    of a box 71
content edge 72
’content’ **130**
’continuous’ media group 65
<counter>, definition of 130
crop marks **135**
cross marks **135**
’cue’ **235**
’cue-after’ **235**
’cue-before’ **235**
’cursor’ **227**

’dashed’ 209, 107
declaration **34**
declaration-block **33**
default style sheet 59
’definition-src’ (descriptor) **172**
descendant **26**
descendant-selectors **45**
’descent’ (descriptor) **171**
’direction’ **81**
’display’ **69**
’dotted’ 209, 107
’double’ 209, 108
drop caps 52

element **25**
content 25
rendered content 25
'elevation' **238**
encoding vector **176**

<family-name>
definition of 150
fictional tag sequence **51**, 53, 53
first-child **46**
'float' **83**
following element 26
font **158**
font data **163**
font definition resource **172**
font description **163**
font descriptions **163**
font descriptors **163**
font encoding table **176**
font family **149**
font set **150**
font size 36
font synthesis **162**
'font' **158**
<font-description>
definition of 163
<font-face-name>
definition of 169
'font-family' **150**
'font-family' (descriptor) **165**
'font-size' **154**
'font-size' (descriptor) **166**
'font-size-adjust' **155**
'font-style' **151**
'font-style' (descriptor) **165**
'font-variant' **152**
'font-variant' (descriptor) **165**
'font-weight' **152**
'font-weight' (descriptor) **166**
formatting context **79**
<frequency> 241
definition of 40

<generic-family>
definition of 150
<generic-voice>, definition of 240
glyph **148**
glyph representation **148**
'grid' media group 65

groove 216
'groove' 216, 108

half-leading **118**
'height' **115**
'hidden 209
'hidden' 107
horizontal margin 117
'hover' pseudo-class **54**

identifier 31
'important' **60**, 60
inherit, definition of 59
inheritance of property values **58**
initial caps 52
initial containing block 111
initial value **57**
inline element **69**
'inline', definition of 69
<inline-table>, definition of 70
inner edge 72
inset 216
'inset' 216, 108
<integer>
    definition of 36
intelligent matching **162**
iso-10646 265

leading **118**
<left>
    definition of 126
'left' **78**
<length> 145, 145, 185, 188, 189, 190, 113, 115, 116, 117, 119
    definition of 36
'letter-spacing' **189**
ligatures 189
line box 79
line-box 121
'line-height' **119**
'link' pseudo-class **54**
'list-item', definition of 69
'list-style' **198**
'list-style-image' **196**
'list-style-position' **197**
'list-style-type' **195**
LL(1) 263

margin
    horizontal 117
    of a box 71
    vertical 117

margin edge 72
’margin’ **101**
’margin-bottom’ **101**
’margin-left’ **101**
’margin-right’ **100**
’margin-top’ **100**
<margin-width>
  definition of 100
’marks’ **135**
match **43**
’mathline’ (descriptor) **173**
’max-height’ **117**
’max-width’ **115**
media group **65**
media-dependent import **64**
’min-height’ **117**
’min-width’ **114**

name matching **162**
newline 41
’none’
  as border style 209, 107
  as list style **69**
<number> 232, 239, 241, 241, 242, 40, 40, 40, 119
  definition of 36

’orphans’ **138**
outer edge 72
outset 216
’outset’ 216, 108
overflow **123**
’overflow’ **123**
overlapping pseudo-elements **51**

padding
  of a box 71
padding edge 72
’padding’ **103**
’padding-bottom’ **103**
’padding-left’ **103**
’padding-right’ **102**
’padding-top’ **102**
<padding-width>
  definition of 102
page area 132
page box **132**
page model **131**
’page-break-after’ **137**
’page-break-before’ **137**

page-context 132
’paged’ media group 65
Panose-1 **178**
panose-1, 170
’panose-1’ (descriptor) **169**
parent **26**
pattern subject **43**
’pause’ **234**
’pause-after’ **234**
’pause-before’ **233**
<percentage> 232, 234, 144, 144, 185, 113, 115, 116, 117, 119
    definition of 38
’pitch’ **240**
’pitch-range’ **241**
’play-during’ **236**
’position’ **76**
positioning scheme **76**
preceding box 67, 67
preceding element 26
progressive rendering **163**
property 34
pseudo-class
    :first **136**
    :left **135**
    :right **135**
pseudo-classes **50**
    :active **54**
    :hover **54**
    :link **54**
    :visited **54**
pseudo-elements **50**
    :after **129**, **54**
    :before **129**, **54**
    :first-letter **52**
    :first-line 51, 52

reference pixel **37**
relative positioning **82**
relative units **36**
<relative-size>
    definition of **155**
rendering structure 21
replaced element 26
’richness’ **242**
ridge 216
’ridge’ 216, 108
<right>
    definition of 126
’right’ **77**

root **26**
’row-span’ **204**
rule sets 32
run-in box 71
’run-in’, definition of 70

screen reader **231**
selector 264, **33**
    match **43**
selectors **43**
<shape>
    definition of 126
sheet **131**
shorthand **16**
sibling **26**
’size’ **133**
skip 27, 27, 151, 164, 164, 164, 167, 172, 181, 51, 53, 31, 31, 32, 32, 32, 33, 33, 33, 34, 34, 35, 35, 35, 36, 36, 217, 192, 70, 77
’slope’ (descriptor) **170**
’solid’ 209, 107
’speak’ **233**
’speak-date’ **243**
’speak-header’ **222**
’speak-numeral’ **243**
’speak-punctuation’ **242**
’speak-time’ **244**
<specific-voice>
    definition of 240
specified value **57**
’speech-rate’ **239**
’src’ (descriptor) **168**
stack level **96**
stacking context **97**
statements 32
’stemh’ (descriptor) **170**
’stemv’ (descriptor) **170**
’stress’ **241**
string 33
<string>, definition of 130
strings **41**
system fonts **159**

’table’, definition of 70
’table-caption’, definition of 70
’table-cell’, definition of 70
’table-column-group’, definition of 70
’table-footer-group’, definition of 70
’table-header-group’, definition of 70
’table-layout’ **210**

’table-row’, definition of 70
’table-row-group’, definition of 70
’tactile’ media group 65
’text-align’ **186**
’text-decoration’ **186**
’text-indent’ **185**
’text-shadow’ **187**
’text-transform’ **190**
<time> 234
    definition of 40
tokenizer **265**
<top>
    definition of 126
’top’ **77**
’topline’ (descriptor) **173**
type selector **45**

UA **27**
unicode 265
’unicode-range’ (descriptor) **166**
Uniform Resource Identifier (URI) **38**
’units-per-em’ (descriptor) **168**
universal selector **44**
<urange>
    definition of 167
<uri> 235, 236, 236, 236, 143
    definition of 38
user agent **27**

valid style sheet **25**
value **34**
vertical margin 117
’vertical-align’ **120**
viewport 68
’visibility’ **127**
’visited’ pseudo-class **54**
visual rendering model **67**
’visual’ media group 65
’voice-family’ **240**
volume **231**
’volume’ **231**

’white-space’ **191**
’widows’ **138**
’width’ **112**
’widths’ (descriptor) **171**
’word-spacing’ **190**

x-height 155, **36**

’x-height’ (descriptor) **170**

’z-index’ **96**