

[next](#) [contents](#) [properties](#) [index](#)

REC-CSS2-19980512

Cascading Style Sheets, level 2

CSS2 Specification

W3C Recommendation 12-May-1998

This version:

<http://www.w3.org/TR/1998/REC-CSS2-19980512>

Latest version:

<http://www.w3.org/TR/REC-CSS2>

Previous version:

<http://www.w3.org/TR/1998/PR-CSS2-19980324>

Editors:

[Bert Bos](#) <bbos@w3.org>

[Håkon Wium Lie](#) <howcome@w3.org>

[Chris Lilley](#) <chris@w3.org>

[Ian Jacobs](#) <ijj@w3.org>

Abstract

This specification defines Cascading Style Sheets, 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 documents and XML applications). By separating the presentation style of documents from the content of documents, CSS2 simplifies Web authoring and site maintenance.

CSS2 builds on CSS1 (see [\[CSS1\]](#)) and, with very few exceptions, all valid CSS1 style sheets are valid CSS2 style sheets. CSS2 supports media-specific style sheets so that authors may tailor the presentation of their documents to visual browsers, aural devices, printers, braille devices, handheld devices, etc. This specification also supports content positioning, downloadable fonts, table layout, features for internationalization, automatic counters and numbering, and some properties related to user interface.

Status of this document

This document has been reviewed by W3C Members and other interested parties and has been endorsed by the Director as a W3C Recommendation. It is a stable document and may be used as reference material or cited as a normative reference from another document. W3C's role in making the Recommendation is to draw attention to the specification and to promote its widespread deployment. This enhances the functionality and interoperability of the Web.

A list of current W3C Recommendations and other technical documents can be found at <http://www.w3.org/TR>.

Public discussion of CSS features takes place on www-style@w3.org.

Available formats

The CSS2 specification is available in the following formats:

HTML:

<http://www.w3.org/TR/1998/REC-CSS2-19980512>

a plain text file:

<http://www.w3.org/TR/1998/REC-CSS2-19980512/css2.txt>,

HTML as a gzip'ed tar file:

<http://www.w3.org/TR/1998/REC-CSS2-19980512/css2.tgz>,

HTML as a zip file (this is a '.zip' file not an '.exe'):

<http://www.w3.org/TR/1998/REC-CSS2-19980512/css2.zip>,

as well as a gzip'ed PostScript file:

<http://www.w3.org/TR/1998/REC-CSS2-19980512/css2.ps.gz>,

and a PDF file:

<http://www.w3.org/TR/1998/REC-CSS2-19980512/css2.pdf>.

In case of a discrepancy between the various forms of the specification, <http://www.w3.org/TR/1998/REC-CSS2-19980512> 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/Style/css2-updates/translations.html>.

Errata

The list of known errors in this specification is available at <http://www.w3.org/Style/css2-updates/REC-CSS2-19980512-errata.html>. Please report errors in this document to css2-editors@w3.org.

Quick Table of Contents

- [1 About the CSS2 Specification](#)
- [2 Introduction to CSS2](#)
- [3 Conformance: Requirements and Recommendations](#)
- [4 CSS2 syntax and basic data types](#)
- [5 Selectors](#)
- [6 Assigning property values, Cascading, and Inheritance](#)
- [7 Media types](#)
- [8 Box model](#)
- [9 Visual formatting model](#)
- [10 Visual formatting model details](#)
- [11 Visual effects](#)
- [12 Generated content, automatic numbering, and lists](#)
- [13 Paged media](#)
- [14 Colors and Backgrounds](#)

[15 Fonts](#)[16 Text](#)[17 Tables](#)[18 User interface](#)[19 Aural style sheets](#)[Appendix A. A sample style sheet for HTML 4.0](#)[Appendix B. Changes from CSS1](#)[Appendix C. Implementation and performance notes for fonts](#)[Appendix D. The grammar of CSS2](#)[Appendix E. References](#)[Appendix F. Property index](#)[Appendix G. Descriptor index](#)[Appendix H. Index](#)

Full Table of Contents

[1 About the CSS2 Specification](#)

[1.1 Reading the specification](#)[1.2 How the specification is organized](#)[1.3 Conventions](#)[1.3.1 Document language elements and attributes](#)[1.3.2 CSS property definitions](#)[Value](#)[Initial](#)[Applies to](#)[Inherited](#)[Percentage values](#)[Media groups](#)[1.3.3 Shorthand properties](#)[1.3.4 Notes and examples](#)[1.3.5 Images and long descriptions](#)[1.4 Acknowledgments](#)[1.5 Copyright Notice](#)

[2 Introduction to CSS2](#)

[2.1 A brief CSS2 tutorial for HTML](#)[2.2 A brief CSS2 tutorial for XML](#)[2.3 The CSS2 processing model](#)[2.3.1 The canvas](#)[2.3.2 CSS2 addressing model](#)[2.4 CSS design principles](#)

[3 Conformance: Requirements and Recommendations](#)

[3.1 Definitions](#)[3.2 Conformance](#)[3.3 Error conditions](#)[3.4 The text/css content type](#)

[4 CSS2 syntax and basic data types](#)

[4.1 Syntax](#)[4.1.1 Tokenization](#)[4.1.2 Keywords](#)[4.1.3 Characters and case](#)[4.1.4 Statements](#)[4.1.5 At-rules](#)[4.1.6 Blocks](#)

- [4.1.7 Rule sets, declaration blocks, and selectors](#)
 - [4.1.8 Declarations and properties](#)
 - [4.1.9 Comments](#)
- [4.2 Rules for handling parsing errors](#)
- [4.3 Values](#)
 - [4.3.1 Integers and real numbers](#)
 - [4.3.2 Lengths](#)
 - [4.3.3 Percentages](#)
 - [4.3.4 URL + URN = URI](#)
 - [4.3.5 Counters](#)
 - [4.3.6 Colors](#)
 - [4.3.7 Angles](#)
 - [4.3.8 Times](#)
 - [4.3.9 Frequencies](#)
 - [4.3.10 Strings](#)
- [4.4 CSS document representation](#)
 - [4.4.1 Referring to characters not represented in a character encoding](#)
- 5 Selectors**
 - [5.1 Pattern matching](#)
 - [5.2 Selector syntax](#)
 - [5.2.1 Grouping](#)
 - [5.3 Universal selector](#)
 - [5.4 Type selectors](#)
 - [5.5 Descendant selectors](#)
 - [5.6 Child selectors](#)
 - [5.7 Adjacent sibling selectors](#)
 - [5.8 Attribute selectors](#)
 - [5.8.1 Matching attributes and attribute values](#)
 - [5.8.2 Default attribute values in DTDs](#)
 - [5.8.3 Class selectors](#)
 - [5.9 ID selectors](#)
 - [5.10 Pseudo-elements and pseudo-classes](#)
 - [5.11 Pseudo-classes](#)
 - [5.11.1 :first-child pseudo-class](#)
 - [5.11.2 The link pseudo-classes: :link and :visited](#)
 - [5.11.3 The dynamic pseudo-classes: :hover, :active, and :focus](#)
 - [5.11.4 The language pseudo-class: :lang](#)
 - [5.12 Pseudo-elements](#)
 - [5.12.1 The :first-line pseudo-element](#)
 - [5.12.2 The :first-letter pseudo-element](#)
 - [5.12.3 The :before and :after pseudo-elements](#)
- 6 Assigning property values, Cascading, and Inheritance**
 - [6.1 Specified, computed, and actual values](#)
 - [6.1.1 Specified values](#)
 - [6.1.2 Computed values](#)
 - [6.1.3 Actual values](#)
 - [6.2 Inheritance](#)
 - [6.2.1 The 'inherit' value](#)
 - [6.3 The @import rule](#)
 - [6.4 The cascade](#)
 - [6.4.1 Cascading order](#)
 - [6.4.2 !important rules](#)
 - [6.4.3 Calculating a selector's specificity](#)
 - [6.4.4 Precedence of non-CSS presentational hints](#)

7 Media types

[7.1 Introduction to media types](#)

[7.2 Specifying media-dependent style sheets](#)

[7.2.1 The @media rule](#)

[7.3 Recognized media types](#)

[7.3.1 Media groups](#)

8 Box model

[8.1 Box dimensions](#)

[8.2 Example of margins, padding, and borders](#)

[8.3 Margin properties: 'margin-top', 'margin-right', 'margin-bottom', 'margin-left', and 'margin'](#)

[8.3.1 Collapsing margins](#)

[8.4 Padding properties: 'padding-top', 'padding-right', 'padding-bottom', 'padding-left', and 'padding'](#)

[8.5 Border properties](#)

[8.5.1 Border width: 'border-top-width', 'border-right-width', 'border-bottom-width', 'border-left-width', and 'border-width'](#)

[8.5.2 Border color: 'border-top-color', 'border-right-color', 'border-bottom-color', 'border-left-color', and 'border-color'](#)

[8.5.3 Border style: 'border-top-style', 'border-right-style', 'border-bottom-style', 'border-left-style', and 'border-style'](#)

[8.5.4 Border shorthand properties: 'border-top', 'border-bottom', 'border-right', 'border-left', and 'border'](#)

9 Visual formatting model

[9.1 Introduction to the visual formatting model](#)

[9.1.1 The viewport](#)

[9.1.2 Containing blocks](#)

[9.2 Controlling box generation](#)

[9.2.1 Block-level elements and block boxes](#)

[*Anonymous block boxes*](#)

[9.2.2 Inline-level elements and inline boxes](#)

[*Anonymous inline boxes*](#)

[9.2.3 Compact boxes](#)

[9.2.4 Run-in boxes](#)

[9.2.5 The 'display' property](#)

[9.3 Positioning schemes](#)

[9.3.1 Choosing a positioning scheme: 'position' property](#)

[9.3.2 Box offsets: 'top', 'right', 'bottom', 'left'](#)

[9.4 Normal flow](#)

[9.4.1 Block formatting context](#)

[9.4.2 Inline formatting context](#)

[9.4.3 Relative positioning](#)

[9.5 Floats](#)

[9.5.1 Positioning the float: the 'float' property](#)

[9.5.2 Controlling flow next to floats: the 'clear' property](#)

[9.6 Absolute positioning](#)

[9.6.1 Fixed positioning](#)

[9.7 Relationships between 'display', 'position', and 'float'](#)

[9.8 Comparison of normal flow, floats, and absolute positioning](#)

[9.8.1 Normal flow](#)

[9.8.2 Relative positioning](#)

[9.8.3 Floating a box](#)

[9.8.4 Absolute positioning](#)

[9.9 Layered presentation](#)

- [9.9.1 Specifying the stack level: the 'z-index' property](#)
- [9.10 Text direction: the 'direction' and 'unicode-bidi' properties](#)

[10 Visual formatting model details](#)

- [10.1 Definition of "containing block"](#)
- [10.2 Content width: the 'width' property](#)
- [10.3 Computing widths and margins](#)
 - [10.3.1 Inline, non-replaced elements](#)
 - [10.3.2 Inline, replaced elements](#)
 - [10.3.3 Block-level, non-replaced elements in normal flow](#)
 - [10.3.4 Block-level, replaced elements in normal flow](#)
 - [10.3.5 Floating, non-replaced elements](#)
 - [10.3.6 Floating, replaced elements](#)
 - [10.3.7 Absolutely positioned, non-replaced elements](#)
 - [10.3.8 Absolutely positioned, replaced elements](#)
- [10.4 Minimum and maximum widths: 'min-width' and 'max-width'](#)
- [10.5 Content height: the 'height' property](#)
- [10.6 Computing heights and margins](#)
 - [10.6.1 Inline, non-replaced elements](#)
 - [10.6.2 Inline, replaced elements block-level, replaced elements in normal flow, and floating, replaced elements](#)
 - [10.6.3 Block-level, non-replaced elements in normal flow, and floating, non-replaced elements](#)
 - [10.6.4 Absolutely positioned, non-replaced elements](#)
 - [10.6.5 Absolutely positioned, replaced elements](#)
- [10.7 Minimum and maximum heights: 'min-height' and 'max-height'](#)
- [10.8 Line height calculations: the 'line-height' and 'vertical-align' properties](#)
 - [10.8.1 Leading and half-leading](#)

[11 Visual effects](#)

- [11.1 Overflow and clipping](#)
 - [11.1.1 Overflow: the 'overflow' property](#)
 - [11.1.2 Clipping: the 'clip' property](#)
- [11.2 Visibility: the 'visibility' property](#)

[12 Generated content, automatic numbering, and lists](#)

- [12.1 The :before and :after pseudo-elements](#)
- [12.2 The 'content' property](#)
- [12.3 Interaction of :before and :after with 'compact' and 'run-in' elements](#)
- [12.4 Quotation marks](#)
 - [12.4.1 Specifying quotes with the 'quotes' property](#)
 - [12.4.2 Inserting quotes with the 'content' property](#)
- [12.5 Automatic counters and numbering](#)
 - [12.5.1 Nested counters and scope](#)
 - [12.5.2 Counter styles](#)
 - [12.5.3 Counters in elements with 'display: none'](#)
- [12.6 Markers and lists](#)
 - [12.6.1 Markers: the 'marker-offset' property](#)
 - [12.6.2 Lists: the 'list-style-type', 'list-style-image', 'list-style-position', and 'list-style' properties](#)

[13 Paged media](#)

- [13.1 Introduction to paged media](#)
- [13.2 Page boxes: the @page rule](#)
 - [13.2.1 Page margins](#)
 - [13.2.2 Page size: the 'size' property](#)
 - [*Rendering page boxes that do not fit a target sheet*](#)
 - [*Positioning the page box on the sheet*](#)

[13.2.3 Crop marks: the 'marks' property](#)

[13.2.4 Left, right, and first pages](#)

[13.2.5 Content outside the page box](#)

[13.3 Page breaks](#)

[13.3.1 Break before/after elements: 'page-break-before', 'page-break-after', 'page-break-inside'](#)

[13.3.2 Using named pages: 'page'](#)

[13.3.3 Breaks inside elements: 'orphans', 'widows'](#)

[13.3.4 Allowed page breaks](#)

[13.3.5 Forced page breaks](#)

[13.3.6 "Best" page breaks](#)

[13.4 Cascading in the page context](#)

[14 Colors and Backgrounds](#)

[14.1 Foreground color: the 'color' property](#)

[14.2 The background](#)

[14.2.1 Background properties: 'background-color', 'background-image', 'background-repeat', 'background-attachment', 'background-position', and 'background'](#)

[14.3 Gamma correction](#)

[15 Fonts](#)

[15.1 Introduction](#)

[15.2 Font specification](#)

[15.2.1 Font specification properties](#)

[15.2.2 Font family: the 'font-family' property](#)

[15.2.3 Font styling: the 'font-style', 'font-variant', 'font-weight' and 'font-stretch' properties](#)

[15.2.4 Font size: the 'font-size' and 'font-size-adjust' properties](#)

[15.2.5 Shorthand font property: the 'font' property](#)

[15.2.6 Generic font families](#)

[serif](#)

[sans-serif](#)

[cursive](#)

[fantasy](#)

[monospace](#)

[15.3 Font selection](#)

[15.3.1 Font Descriptions and @font-face](#)

[15.3.2 Descriptors for Selecting a Font: 'font-family', 'font-style', 'font-variant', 'font-weight', 'font-stretch' and 'font-size'](#)

[15.3.3 Descriptors for Font Data Qualification: 'unicode-range'](#)

[15.3.4 Descriptor for Numeric Values: 'units-per-em'](#)

[15.3.5 Descriptor for Referencing: 'src'](#)

[15.3.6 Descriptors for Matching: 'panose-1', 'stemv', 'stemh', 'slope', 'cap-height', 'x-height', 'ascent', and 'descent'](#)

[15.3.7 Descriptors for Synthesis: 'widths', 'bbox' and 'definition-src'](#)

[15.3.8 Descriptors for Alignment: 'baseline', 'centerline', 'mathline', and 'topline'](#)

[15.3.9 Examples](#)

[15.4 Font Characteristics](#)

[15.4.1 Introducing Font Characteristics](#)

[15.4.2 Full font name](#)

[15.4.3 Coordinate units on the em square](#)

[15.4.4 Central Baseline](#)

[15.4.5 Font Encoding](#)

[15.4.6 Font family name](#)

- [15.4.7 Glyph widths](#)
- [15.4.8 Horizontal stem width](#)
- [15.4.9 Height of uppercase glyphs](#)
- [15.4.10 Height of lowercase glyphs](#)
- [15.4.11 Lower Baseline](#)
- [15.4.12 Mathematical Baseline](#)
- [15.4.13 Maximal bounding box](#)
- [15.4.14 Maximum unaccented height](#)
- [15.4.15 Maximum unaccented depth](#)
- [15.4.16 Panose-1 number](#)
- [15.4.17 Range of ISO 10646 characters](#)
- [15.4.18 Top Baseline](#)
- [15.4.19 Vertical stem width](#)
- [15.4.20 Vertical stroke angle](#)

[15.5 Font matching algorithm](#)

- [15.5.1 Mapping font weight values to font names](#)
- [15.5.2 Examples of font matching](#)

16 Text

- [16.1 Indentation: the 'text-indent' property.](#)
- [16.2 Alignment: the 'text-align' property.](#)
- [16.3 Decoration](#)
 - [16.3.1 Underlining, overlining, striking, and blinking: the 'text-decoration' property.](#)
 - [16.3.2 Text shadows: the 'text-shadow' property.](#)
- [16.4 Letter and word spacing: the 'letter-spacing' and 'word-spacing' properties](#)
- [16.5 Capitalization: the 'text-transform' property.](#)
- [16.6 Whitespace: the 'white-space' property.](#)

17 Tables

- [17.1 Introduction to tables](#)
- [17.2 The CSS table model](#)
 - [17.2.1 Anonymous table objects](#)
- [17.3 Column selectors](#)
- [17.4 Tables in the visual formatting model](#)
 - [17.4.1 Caption position and alignment](#)
- [17.5 Visual layout of table contents](#)
 - [17.5.1 Table layers and transparency](#)
 - [17.5.2 Table width algorithms: the 'table-layout' property.](#)
 - [*Fixed table layout*](#)
 - [*Automatic table layout*](#)
 - [17.5.3 Table height algorithms](#)
 - [17.5.4 Horizontal alignment in a column](#)
 - [17.5.5 Dynamic row and column effects](#)
- [17.6 Borders](#)
 - [17.6.1 The separated borders model](#)
 - [*Borders around empty cells: the 'empty-cells' property.*](#)
 - [17.6.2 The collapsing border model](#)
 - [*Border conflict resolution*](#)
 - [17.6.3 Border styles](#)
- [17.7 Audio rendering of tables](#)
 - [17.7.1 Speaking headers: the 'speak-header' property.](#)

18 User interface

- [18.1 Cursors: the 'cursor' property.](#)
- [18.2 User preferences for colors](#)
- [18.3 User preferences for fonts](#)

[18.4 Dynamic outlines: the 'outline' property](#)[18.4.1 Outlines and the focus](#)[18.5 Magnification](#)[**19 Aural style sheets**](#)[19.1 Introduction to aural style sheets](#)[19.2 Volume properties: 'volume'](#)[19.3 Speaking properties: 'speak'](#)[19.4 Pause properties: 'pause-before', 'pause-after', and 'pause'](#)[19.5 Cue properties: 'cue-before', 'cue-after', and 'cue'](#)[19.6 Mixing properties: 'play-during'](#)[19.7 Spatial properties: 'azimuth' and 'elevation'](#)[19.8 Voice characteristic properties: 'speech-rate', 'voice-family', 'pitch', 'pitch-range', 'stress', and 'richness'](#)[19.9 Speech properties: 'speak-punctuation' and 'speak-numeral'](#)[**Appendix A. A sample style sheet for HTML 4.0**](#)[**Appendix B. Changes from CSS1**](#)[B.1 New functionality](#)[B.2 Updated descriptions](#)[B.3 Semantic changes from CSS1](#)[**Appendix C. Implementation and performance notes for fonts**](#)[C.1 Glossary of font terms](#)[C.2 Font retrieval](#)[C.3 Meaning of the Panose Digits](#)[C.4 Deducing Unicode Ranges for TrueType](#)[C.5 Automatic descriptor generation](#)[**Appendix D. The grammar of CSS2**](#)[D.1 Grammar](#)[D.2 Lexical scanner](#)[D.3 Comparison of tokenization in CSS2 and CSS1](#)[**Appendix E. References**](#)[E.1 Normative references](#)[E.2 Informative references](#)[**Appendix F. Property index**](#)[**Appendix G. Descriptor index**](#)[**Appendix H. Index**](#)

[Copyright](#) © 1998 [W3C](#) ([MIT](#), [INRIA](#), [Keio](#)), All Rights Reserved.

[next](#) [contents](#) [properties](#) [index](#)

[previous](#) [next](#) [contents](#) [properties](#) [index](#)

8 Box model

Contents

[8.1 Box dimensions](#)

[8.2 Example of margins, padding, and borders](#)

[8.3 Margin properties: 'margin-top', 'margin-right', 'margin-bottom', 'margin-left', and 'margin'](#)

[8.3.1 Collapsing margins](#)

[8.4 Padding properties: 'padding-top', 'padding-right', 'padding-bottom', 'padding-left', and 'padding'](#)

[8.5 Border properties](#)

[8.5.1 Border width: 'border-top-width', 'border-right-width', 'border-bottom-width', 'border-left-width', and 'border-width'](#)

[8.5.2 Border color: 'border-top-color', 'border-right-color', 'border-bottom-color', 'border-left-color', and 'border-color'](#)

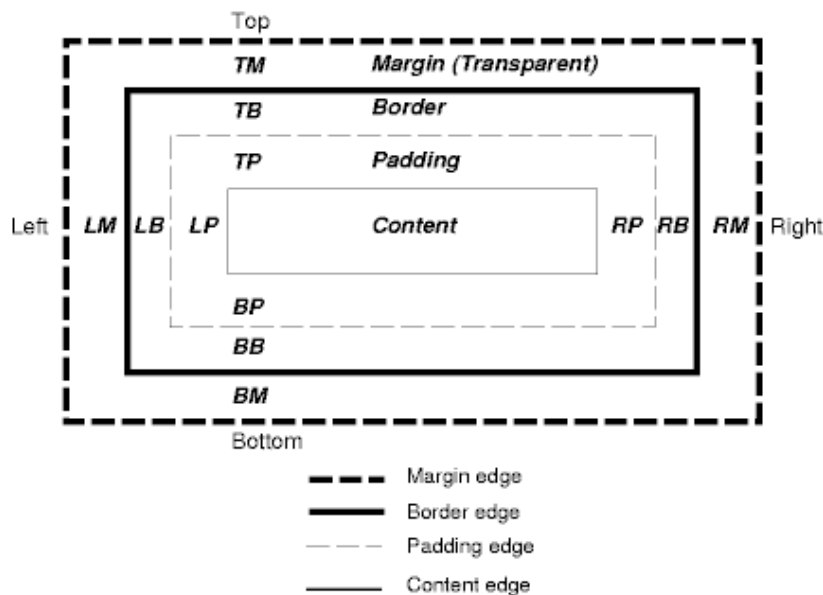
[8.5.3 Border style: 'border-top-style', 'border-right-style', 'border-bottom-style', 'border-left-style', and 'border-style'](#)

[8.5.4 Border shorthand properties: 'border-top', 'border-bottom', 'border-right', 'border-left', and 'border'](#)

The CSS box model describes the rectangular boxes that are generated for elements in the [document tree](#) and laid out according to the [visual formatting model](#). The [page box](#) is a special kind of box that is described in detail in the section on [paged media](#).

8.1 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 below. The following diagram shows how these areas relate and the terminology used to refer to pieces of margin, border, and padding:



[D]

The margin, border, and padding can be broken down into left, right, top, and bottom segments (e.g., in the diagram, "LM" for left margin, "RP" for right padding, "TB" for top border, 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](#).

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](#) established by the box.

border edge

The border edge surrounds the box's border. If the border 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* -- depend on several factors: whether the element generating the box has the *'width'* or *'height'* property set, whether the box contains text or other boxes, whether the box is a table, etc. Box widths and heights are discussed in the chapter on [visual formatting model details](#).

The *box width* is given by the sum of the left and right margins, border, and padding, and the content width. The *height* is given by the sum of the top and bottom margins, border, and padding, and the content height.

The background style of the various areas of a box are determined as follows:

- *Content area*: The *'background'* property of the generating element.
- *Padding area*: The *'background'* property of the generating element.
- *Border area*: The [border properties](#) of the generating element.

- *Margin area:* Margins are always transparent.

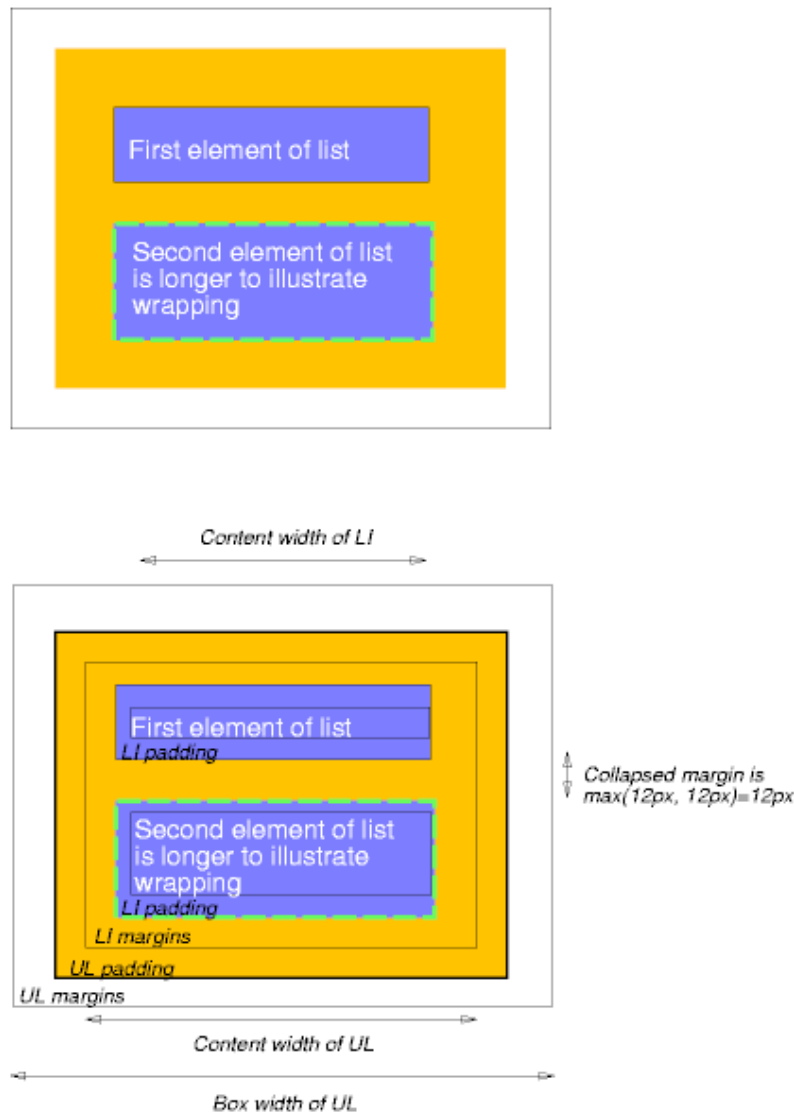
8.2 Example of margins, padding, and borders

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

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <HEAD>
    <TITLE>Examples of margins, padding, and borders</TITLE>
    <STYLE type="text/css">
      UL {
        background: green;
        margin: 12px 12px 12px 12px;
        padding: 3px 3px 3px 3px;
        /* No borders set */
      }
      LI {
        color: black;           /* text color is black */
        background: gray;      /* Content, padding will be gray */
        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: black;
      }
    </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](#) 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.



[D]

Note that:

- The [content width](#) for each LI box is calculated top-down; the [containing block](#) for each LI box is established by the UL element.
- The height of each LI box is given by its [content height](#), plus top and bottom padding, borders, and margins. Note that vertical margins between the LI boxes [collapse](#).
- 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 margins of the LI boxes are transparent -- margins are always transparent -- so the background color (green) of the UL padding and content areas shines through them.
- The second LI element specifies a dashed border (the '[border-style](#)' property).

8.3 Margin properties: '[margin-top](#)', '[margin-right](#)', '[margin-bottom](#)', '[margin-left](#)', and '[margin](#)'

Margin properties specify the width of the [margin area](#) of a box. The '[margin](#)' shorthand property sets the margin for all four sides while the other margin properties only set their

respective side.

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>

The percentage is calculated with respect to the *width* of the generated box's [containing block](#). This is true for `'margin-top'` and `'margin-bottom'`, except in the [page context](#), where percentages refer to page box height.

auto

See the section on [computing widths and margins](#) for behavior.

Negative values for margin properties are allowed, but there may be implementation-specific limits.

'margin-top', 'margin-right', 'margin-bottom', 'margin-left'

Value: `<margin-width>` | `inherit`
Initial: 0
Applies to: all elements
Inherited: no
Percentages: refer to width of containing block
Media: `visual`

These properties set the top, right, bottom, and left margin of a box.

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

'margin'

Value: `<margin-width>{1,4}` | `inherit`
Initial: not defined for shorthand properties
Applies to: all elements
Inherited: no
Percentages: refer to width of containing block
Media: `visual`

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) */
}
```

8.3.1 Collapsing margins

In this specification, the expression *collapsing margins* means that adjoining margins (no padding or border areas separate them) of two or more boxes (which may be next to one another or nested) combine to form a single margin.

In CSS2, horizontal margins never collapse.

Vertical margins may collapse between certain boxes:

- Two or more adjoining vertical margins of [block](#) 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 [floated](#) box and any other box do not collapse.
- Margins of [absolutely](#) and relatively positioned boxes do not collapse.

Please consult the [examples of margin, padding, and borders](#) for an illustration of collapsed margins.

8.4 Padding properties: 'padding-top', 'padding-right', 'padding-bottom', 'padding-left', and 'padding'

The padding properties specify the width of the [padding area](#) of a box. The 'padding' shorthand property sets the padding for all four sides while the other padding properties only set their respective side.

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>

The percentage is calculated with respect to the *width* of the generated box's [containing block](#), even for 'padding-top' and 'padding-bottom'.

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', 'padding-right', 'padding-bottom', 'padding-left'

<i>Value:</i>	<u><padding-width></u> <u>inherit</u>
<i>Initial:</i>	0
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	refer to width of containing block
<i>Media:</i>	<u>visual</u>

These properties set the top, right, bottom, and left padding of a box.

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

'padding'

Value: $\leq \text{padding-width} \{1,4\}$ | inherit
Initial: not defined for shorthand properties
Applies to: all elements
Inherited: no
Percentages: refer to width of containing block
Media: visual

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 to the element's font size: '1em' is equal to the size of the font in use.

8.5 Border properties

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

Note. *Notably for HTML, user agents may render borders for certain elements (e.g., buttons, menus, etc.) differently than for "ordinary" elements.*

8.5.1 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. 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', 'border-right-width', 'border-bottom-width', 'border-left-width'

Value: <border-width> | inherit
Initial: medium
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

These properties set the width of the top, right, bottom, and left border of a box.

'border-width'

Value: <border-width>{1,4} | inherit
Initial: see individual properties
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

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 */
```

8.5.2 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', 'border-right-color', 'border-bottom-color', 'border-left-color'

Value: <color> | inherit
Initial: the value of the 'color' property
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

'border-color'

<i>Value:</i>	<code><color></code> {1,4} transparent <u>inherit</u>
<i>Initial:</i>	see individual properties
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	<u>visual</u>

The 'border-color' property sets the color of the four borders. Values have the following meanings:

<color>

Specifies a color value.

transparent

The border is transparent (though it may have width).

The 'border-color' property 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](#) for the border color.

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

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

8.5.3 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. This value forces the computed value of 'border-width' to be '0'.

hidden

Same as 'none', except in terms of [border conflict resolution](#) for [table elements](#).

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 'groove': 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 border 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](#) may interpret 'dotted', 'dashed', 'double', 'groove', 'ridge', 'inset', and 'outset' to be 'solid'.

'border-top-style', 'border-right-style', 'border-bottom-style', 'border-left-style'

Value: <border-style> | inherit
Initial: none
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

'border-style'

Value: <border-style>{1,4} | inherit
Initial: see individual properties
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

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.

8.5.4 Border shorthand properties: 'border-top', 'border-bottom', 'border-right', 'border-left', and 'border'

'border-top', 'border-right', 'border-bottom', 'border-left'

Value: [<'border-top-width'> || <'border-style'> || <color>] | inherit
Initial: see individual properties
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

This is a shorthand property for setting the width, style, and color of the top, right, bottom, and left 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 are set to their initial values. Since the following rule does not specify a border color, the border will have the color specified by the 'color' property:

```
H1 { border-bottom: thick solid }
```

'border'

Value: [<'border-width'> || <'border-style'> || <color>] | inherit
Initial: see individual properties
Applies to: all elements
Inherited: no
Percentages: N/A
Media: visual

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.

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.

[previous](#) [next](#) [contents](#) [properties](#) [index](#)

[previous](#) [next](#) [contents](#) [properties](#) [index](#)

9 Visual formatting model

Contents

- [9.1 Introduction to the visual formatting model](#)
 - [9.1.1 The viewport](#)
 - [9.1.2 Containing blocks](#)
- [9.2 Controlling box generation](#)
 - [9.2.1 Block-level elements and block boxes](#)
 - Anonymous block boxes*
 - [9.2.2 Inline-level elements and inline boxes](#)
 - Anonymous inline boxes*
 - [9.2.3 Compact boxes](#)
 - [9.2.4 Run-in boxes](#)
 - [9.2.5 The 'display' property](#)
- [9.3 Positioning schemes](#)
 - [9.3.1 Choosing a positioning scheme: 'position' property](#)
 - [9.3.2 Box offsets: 'top', 'right', 'bottom', 'left'](#)
- [9.4 Normal flow](#)
 - [9.4.1 Block formatting context](#)
 - [9.4.2 Inline formatting context](#)
 - [9.4.3 Relative positioning](#)
- [9.5 Floats](#)
 - [9.5.1 Positioning the float: the 'float' property](#)
 - [9.5.2 Controlling flow next to floats: the 'clear' property](#)
- [9.6 Absolute positioning](#)
 - [9.6.1 Fixed positioning](#)
- [9.7 Relationships between 'display', 'position', and 'float'](#)
- [9.8 Comparison of normal flow, floats, and absolute positioning](#)
 - [9.8.1 Normal flow](#)
 - [9.8.2 Relative positioning](#)
 - [9.8.3 Floating a box](#)
 - [9.8.4 Absolute positioning](#)
- [9.9 Layered presentation](#)
 - [9.9.1 Specifying the stack level: the 'z-index' property](#)
- [9.10 Text direction: the 'direction' and 'unicode-bidi' properties](#)

9.1 Introduction to the visual formatting model

This chapter and the next describe the visual formatting model: how user agents process the [document tree](#) for visual [media](#).

In the visual formatting model, each element in the document tree generates zero or more boxes according to the [box model](#). The layout of these boxes is governed by:

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

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

The visual formatting model does not specify all aspects of formatting (e.g., it does not specify a letter-spacing algorithm). [Conforming user agents](#) may behave differently for those formatting issues not covered by this specification.

9.1.1 The viewport

User agents for [continuous media](#) 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](#)). When the viewport is smaller than the document's initial containing block, the user agent should offer a scrolling mechanism. There is at most one viewport per [canvas](#), but user agents may render to more than one canvas (i.e., provide different views of the same document).

9.1.2 Containing blocks

In CSS2, many box positions and sizes are calculated with respect to the edges of a rectangular box called a *containing block*. In general, generated boxes act as containing blocks for descendant boxes; we say that a box "establishes" the containing block for its descendants. The phrase "a box's containing block" means "the containing block in which the box lives," not the one it generates.

Each box is given a position with respect to its containing block, but it is not confined by this containing block; it may [overflow](#).

The root of the [document tree](#) generates a box that serves as the *initial containing block* for subsequent layout.

The width of the initial containing block may be specified with the ['width'](#) property for the root element. If this property has the value 'auto', the user agent supplies the initial width (e.g., the user agent uses the current width of the [viewport](#)).

The height of the initial containing block may be specified with the ['height'](#) property for the root element. If this property has the value 'auto', the containing block height will grow to accommodate the document's content.

The initial containing block cannot be positioned or floated (i.e., user agents [ignore](#) the ['position'](#) and ['float'](#) properties for the root element).

The [details](#) of how a containing block's dimensions are calculated are described in the [next chapter](#).

9.2 Controlling box generation

The following sections describe the types of boxes that may be generated in CSS2. A box's type affects, in part, its behavior in the visual formatting model. The ['display'](#) property, described

below, specifies a box's type.

9.2.1 Block-level elements and block boxes

Block-level elements are those elements of the source document that 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](#)), and 'table'.

Block-level elements generate a *principal block box* that only contains *block boxes*. The principal block box establishes the [containing block](#) for descendant boxes and generated content and is also the box involved in any positioning scheme. Principal block boxes participate in a [block formatting context](#).

Some block-level elements generate additional boxes outside of the principal box: 'list-item' elements and those with [markers](#). These additional boxes are placed with respect to the principal box.

Anonymous block boxes

In a document like this:

```
<DIV>
  Some text
  <P>More text
</DIV>
```

(and assuming the DIV and the P both have 'display: block'), the DIV appears to have both inline content and block content. To make it easier to define the formatting, we assume that there is an *anonymous block box* around "Some text".

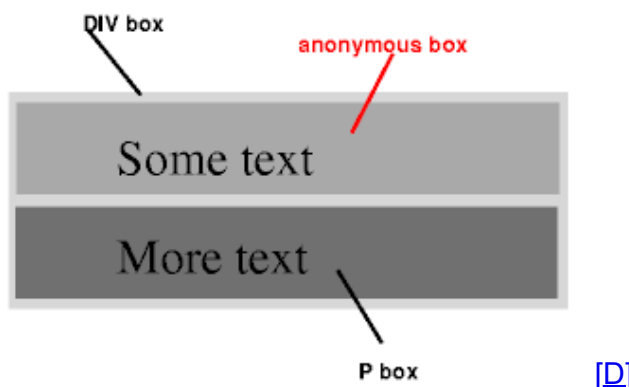


Diagram showing the three boxes, of which one is anonymous, for the example above.

In other words: if a block box (such as that generated for the DIV above) has another block box inside it (such as the P above), then we force it to have *only* block boxes inside it, by wrapping any inline boxes in an anonymous block box.

This model would apply in the following example if the following rules:

```
/* Note: HTML UAs may not respect these rules */
BODY { display: inline }
P     { display: block }
```

were used with this HTML document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HEAD>
<TITLE>Anonymous text interrupted by a block</TITLE>
</HEAD>
<BODY>
  This is anonymous text before the P.
  <P>This is the content of P.</P>
  This is anonymous text after the P.
</BODY>
```

The BODY element contains a chunk (C1) of anonymous text followed by a block-level element followed by another chunk (C2) of anonymous text. The resulting boxes would be an anonymous block box for BODY, containing an anonymous block box around C1, the P block box, and another anonymous block box around C2.

The properties of anonymous boxes are inherited from the enclosing non-anonymous box (in the example: the one for DIV). Non-inherited properties have their initial value. For example, the font of the anonymous box is inherited from the DIV, but the margins will be 0.

9.2.2 Inline-level elements and inline boxes

Inline-level elements are those elements of the source document that do not form new blocks of content; the content is distributed in lines (e.g., emphasized pieces of text within a paragraph, 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](#)). Inline-level elements generate inline boxes.

Inline boxes may participate in several formatting contexts:

- Within a block box, an inline boxes participate in an [inline formatting context](#).
- A [compact](#) inline box is given a position in the margin of a block box.
- [Marker](#) boxes are also given positions outside of a block box.

Anonymous inline boxes

In a document like this:

```
<P>Some <EM>emphasized</em> text</P>
```

The P generates a block box, with several inline boxes inside it. The box for "emphasized" is an inline box generated by an inline element (EM), but the other boxes ("Some" and "text") are inline boxes generated by a block-level element (P). The latter are called anonymous inline boxes, because they don't have an associated inline-level element.

Such anonymous inline boxes inherit inheritable properties from their block parent box. Non-inherited properties have their initial value. In the example, the color of the anonymous initial boxes is inherited from the P, but the background is transparent.

If it is clear from the context which type of anonymous box is meant, both anonymous inline boxes and anonymous block boxes are simply called anonymous boxes in this specification.

There are more types of anonymous boxes that arise when formatting [tables](#).

9.2.3 Compact boxes

A *compact box* behaves as follows:

- If a [block](#) box (that does not float and is not [absolutely positioned](#)) follows the compact box, the compact box is formatted like a one-line inline box. The resulting box width is compared to one of the side margins of the block box. The choice of left or right margin is determined by the '[direction](#)' specified for the element producing the [containing block](#) for the compact box and following box. If the inline box width is less than or equal to the margin, the inline box is given a position in the margin as described immediately below.
- Otherwise, the compact box becomes a block box.

The compact box is given a position in the margin as follows: it is outside (to the left or right) of the first [line box](#) of the block, but it affects the calculation of that [line box's height](#). 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 box.

An element that cannot be formatted on one line cannot be placed in the margin of the following block. For example, a 'compact' element in HTML that contains a BR element will always be formatted as a block box (assuming the default style for BR, which inserts a newline). For placing multi-line texts in the margin, the '[float](#)' property is often more appropriate.

The following example illustrates a compact box.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//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 formatted as:

```
short      Description goes here

too long for the margin
  Description goes here
```

The '[text-align](#)' property can be used to align the compact element inside the margin: against the left edge of the margin ('left'), against the right edge ('right'), or centered in the margin ('center'). The value 'justify' doesn't apply, and is handled as either 'left' or 'right', depending on the '[direction](#)' of the block-level element in whose margin the compact element is formatted. ('left' if the direction is 'ltr', 'right' if it is 'rtl'.)

Please consult the section on [generated content](#) for information about how compact boxes interact with generated content.

9.2.4 Run-in boxes

A *run-in box* behaves as follows:

- If a [block](#) box (that does not float and is not [absolutely positioned](#)) follows the run-in box, the run-in box becomes the first inline box of the block box.
- Otherwise, the run-in box becomes a block box.

A 'run-in' box is useful for run-in headers, as in this example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <HEAD>
    <TITLE>A run-in box example</TITLE>
    <STYLE type="text/css">
      H3 { display: run-in }
    </STYLE>
  </HEAD>
  <BODY>
    <H3>A run-in heading.</H3>
    <P>And a paragraph of text that
      follows it.
    </BODY>
  </HTML>
```

This example might be formatted as:

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

The properties of the run-in element are inherited from its parent in the source tree, not from the block box it visually becomes part of.

Please consult the section on [generated content](#) for information about how run-in boxes interact with generated content.

9.2.5 The 'display' property

'display'

<i>Value:</i>	inline block list-item run-in compact marker table inline-table table-row-group table-header-group table-footer-group table-row table-column-group table-column table-cell table-caption none <u>inherit</u>
<i>Initial:</i>	inline
<i>Applies to:</i>	all elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	<u>all</u>

The values of this property have the following meanings:

block

This value causes an element to generate a principal block box.

inline

This value causes an element to generate one or more inline boxes.

list-item

This value causes an element (e.g., LI in HTML) to generate a principal block box and a list-item inline box. For information about lists and examples of list formatting, please consult the section on [lists](#).

marker

This value declares [generated content](#) before or after a box to be a marker. This value should only be used with [:before and :after pseudo-elements](#) attached to block-level elements. In other cases, this value is interpreted as 'inline'. Please consult the section on [markers](#) for more information.

none

This value causes an element to generate **no** boxes in the [formatting structure](#) (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 formatting structure that affect formatting but are not visible themselves. Please consult the section on [visibility](#) for details.

run-in and compact

These values create either block or inline boxes, depending on context. Properties apply to run-in and compact boxes based on their final status (inline-level or block-level). For example, the '[white-space](#)' property only applies if the box becomes a block box.

table, inline-table, table-row-group, table-column, 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](#)).

Note that although the [initial value](#) of '[display](#)' is 'inline', rules in the user agent's [default style sheet](#) may [override](#) this value. See the [sample style sheet](#) 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 may [ignore](#) the '[display](#)' property.

9.3 Positioning schemes

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

1. [Normal flow](#). In CSS2, normal flow includes [block formatting](#) of [block](#) boxes, [inline formatting](#) of [inline](#) boxes, [relative positioning](#) of block or inline boxes, and positioning of [compact](#) and [run-in](#) boxes.
2. [Floats](#). In the float model, a box is first laid out according to the normal flow, then taken out of the flow and shifted to the left or right as far as possible. Content may flow along the side of a float.
3. [Absolute positioning](#). In the absolute positioning model, a box is removed from the normal flow entirely (it has no impact on later siblings) and assigned a position with respect to a containing block.

Note. CSS2's positioning schemes help authors make their documents more accessible by allowing them to avoid mark-up tricks (e.g., invisible images) used for layout effects.

9.3.1 Choosing a positioning scheme: 'position' property

The 'position' and 'float' properties determine which of the CSS2 positioning algorithms is used to calculate the position of a box.

'position'

Value:	static relative absolute fixed <u>inherit</u>
Initial:	static
Applies to:	all elements, but not to generated content
Inherited:	no
Percentages:	N/A
Media:	<u>visual</u>

The values of this property have the following meanings:

static

The box is a normal box, laid out according to the [normal flow](#). The 'left' and 'top' properties do not apply.

relative

The box's position is calculated according to the [normal flow](#) (this is called the position in normal flow). Then the box is offset [relative](#) to its normal position. When a box B is relatively positioned, the position of the following box is calculated as though B were not offset.

absolute

The box's position (and possibly size) is specified with the 'left', 'right', 'top', and 'bottom' properties. These properties specify offsets with respect to the box's [containing block](#). Absolutely positioned boxes are taken out of the normal flow. This means they have no impact on the layout of later siblings. Also, though [absolutely positioned](#) boxes have margins, they do not [collapse](#) with any other margins.

fixed

The box's position is calculated according to the 'absolute' model, but in addition, the box is [fixed](#) with respect to some reference. In the case of [continuous media](#), the box is fixed with respect to the [viewport](#) (and doesn't move when scrolled). In the case of [paged media](#), the box is fixed with respect to the page, even if that page is seen through a [viewport](#) (in the case of a print-preview, for example). Authors may wish to specify 'fixed' in a media-dependent way. For instance, an author may want a box to remain at the top of the [viewport](#) on the screen, but not at the top of each printed page. The two specifications may be separated by using an [@media rule](#), as in:

```
@media screen {
  H1#first { position: fixed }
}
@media print {
  H1#first { position: static }
}
```

9.3.2 Box offsets: 'top', 'right', 'bottom', 'left'

An element is said to be *positioned* if its 'position' property has a value other than 'static'. Positioned elements generate positioned boxes, laid out according to four properties:

'top'

Value: <length> | <percentage> | auto | inherit
Initial: auto
Applies to: positioned elements
Inherited: no
Percentages: refer to height of containing block
Media: visual

This property specifies how far a box's top content edge is offset below the top edge of the box's [containing block](#).

'right'

Value: <length> | <percentage> | auto | inherit
Initial: auto
Applies to: positioned elements
Inherited: no
Percentages: refer to width of containing block
Media: visual

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](#).

'bottom'

Value: <length> | <percentage> | auto | inherit
Initial: auto
Applies to: positioned elements
Inherited: no
Percentages: refer to height of containing block
Media: visual

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

'left'

Value: <length> | <percentage> | auto | inherit
Initial: auto
Applies to: positioned elements
Inherited: no
Percentages: refer to width of containing block
Media: visual

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](#).

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'). For 'top' and 'bottom', if the height of the containing block is not specified explicitly (i.e., it depends on content height), the percentage value is interpreted like 'auto'.

auto

The effect of this value depends on which of related properties have the value 'auto' as well. See the sections on the [width](#) and [height](#) of [absolutely positioned](#), non-replaced elements for details.

For [absolutely positioned](#) boxes, the offsets are with respect to the box's [containing block](#). For relatively positioned boxes, the offsets are with respect to the outer edges of the box itself (i.e., the box is given a position in the normal flow, then offset from that position according to these properties).

9.4 Normal flow

Boxes in the normal flow belong to a formatting context, which may be block or inline, but not both simultaneously. [Block](#) boxes participate in a [block formatting](#) context. [Inline boxes](#) participate in an [inline formatting](#) context.

9.4.1 Block formatting context

In a block formatting context, boxes are laid out one after the other, vertically, beginning at the top of a containing block. The vertical distance between two sibling boxes is determined by the 'margin' properties. Vertical margins between adjacent block boxes in a block formatting context [collapse](#).

In a block formatting context, each box's left outer edge touches the left edge of the containing block (for right-to-left formatting, right edges touch). This is true even in the presence of floats (although a box's *content* area may shrink due to the floats).

For information about page breaks in paged media, please consult the section on [allowed page breaks](#).

9.4.2 Inline formatting context

In an inline formatting context, boxes are laid out horizontally, one after the other, beginning at the top of a containing block. Horizontal margins, borders, and padding are respected between these boxes. The boxes 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](#). The height of a line box is determined by the rules given in the section on [line height calculations](#). 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 horizontally 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.

In general, the left edge of a line box touches the left edge of its containing block and the right edge touches the right edge of its containing block. However, floating boxes may come between the containing block edge and the line box edge. Thus, although line boxes in the same inline formatting context generally have the same width (that of the containing block), they may vary in width if available horizontal space is reduced due to [floats](#). Line boxes in the

same inline formatting context generally vary in height (e.g., one line might contain a tall image while the others contain only text).

When the total width of the inline boxes on a line is less than the width of the line box containing them, their horizontal distribution within the line box is determined by the `'text-align'` property. If that property has the value 'justify', the user agent may stretch the inline boxes as well.

Since an inline box may not exceed the width of a line box, long inline boxes are split into several boxes and these boxes distributed across several line boxes. When an inline box is split, margins, borders, and padding have no visual effect where the split occurs. Formatting of margins, borders, and padding may not be fully defined if the split occurs within a bidirectional embedding.

Inline boxes may also be split into several boxes *within the same line box* due to [bidirectional text processing](#).

Here is an example of inline box construction. 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>
```

The P element generates a block box that contains five inline boxes, three of which are anonymous:

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

To format the paragraph, the user agent flows the five boxes into line boxes. In this example, the box generated for the P element establishes the containing block for the line boxes. If the containing block is sufficiently wide, all the inline boxes will fit into a single line box:

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

If not, the inline boxes 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"). Margins, borders, padding, or text decorations have no visible effect after split1 or before split2.

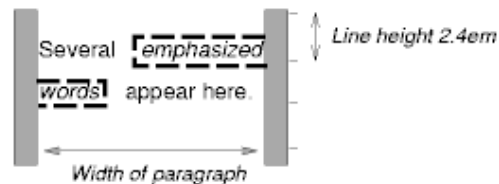
Consider the following example:

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<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:



[D]

- 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.

9.4.3 Relative positioning

Once a box has been laid out according to the [normal flow](#), it may be shifted relative to this position. This is called *relative positioning*. Offsetting a box (B1) in this way has no effect on the box (B2) that follows: B2 is given a position as if B1 were not offset and B2 is not re-positioned after B1's 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](#) for normal flow children and positioned descendants.

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 also 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](#) for more information.

Examples of relative positioning are provided in the section [comparing normal flow, floats, and absolute positioning](#).

9.5 Floats

A float is a box that is shifted to the left or right on the current line. The most interesting characteristic of a float (or "floated" or "floating" box) is that content may flow along its side (or be prohibited from doing so by the '[clear](#)' property). Content flows down the right side of a left-floated box and down the left side of a right-floated box. The following is an introduction to float positioning and content flow; the exact [rules](#) governing float behavior are given in the description of the '[float](#)' property.

A floated box must have an explicit width (assigned via the '[width](#)' property, or its [intrinsic](#) width in the case of [replaced elements](#)). Any floated box becomes a [block box](#) that is shifted to the left or right until its outer edge touches the containing block edge or the outer edge of another float. The top of the floated box is aligned with the top of the current line box (or bottom of the preceding block box if no line box exists). If there isn't enough horizontal room on the current line for the float, it is shifted downward, line by line, until a line has room for it.

Since a float is not in the flow, non-positioned block boxes created before and after the float box flow vertically as if the float didn't exist. However, line boxes created next to the float are shortened to make room for the floated box. Any content in the current line before a floated box is reflowed in the first available line on the other side of the float.

Several floats may be adjacent, and this model also applies to adjacent floats in the same line.

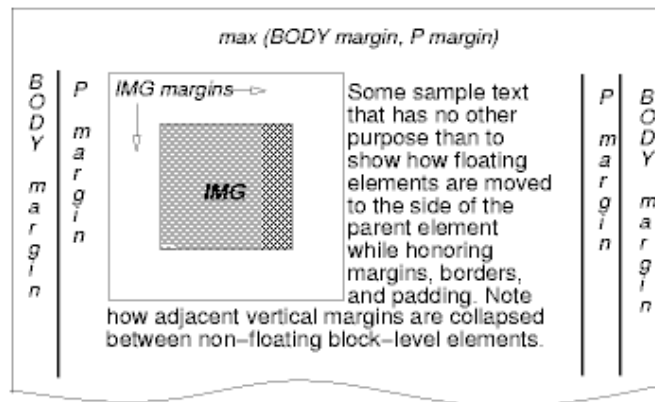
The following rule floats all IMG boxes with `class="icon"` to the left (and sets the left margin to '0'):

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

Consider the following HTML source and style sheet:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <HEAD>
    <TITLE>Float example</TITLE>
    <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>
```

The IMG box is floated to the left. The content that follows is formatted to the right of the float, starting on the same line as the float. The line boxes to the right of the float are shortened due to the float's presence, but resume their "normal" width (that of the containing block established by the P element) after the float. This document might be formatted as:



[D]

Formatting would have been exactly the same if the document had been:

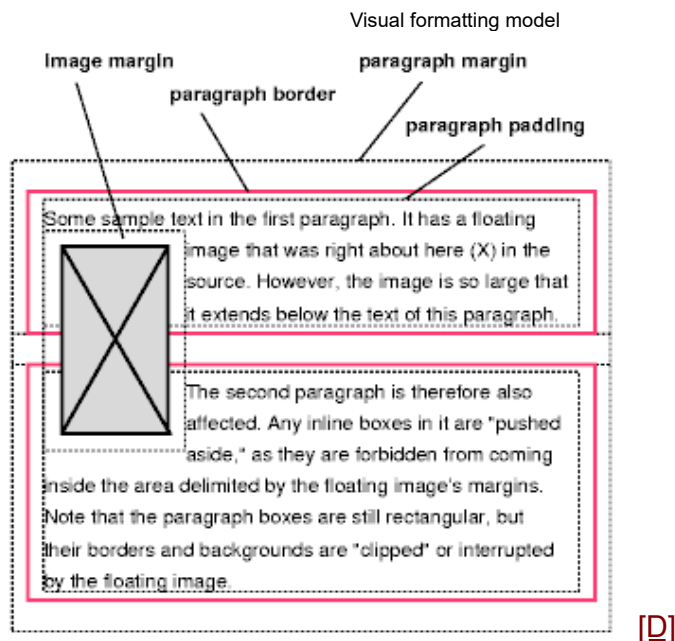
```
<BODY>
  <P>Some sample text
  <IMG src=img.gif alt="This image will illustrate floats">
    that has no other...
</BODY>
```

because the content to the left of the float is displaced by the float and reflowed down its right side.

The margins of floating boxes never [collapse](#) with margins of adjacent boxes. Thus, in the previous example, vertical margins do not [collapse](#) between the P box and the floated IMG box.

A float can overlap other boxes in the normal flow (e.g., when a normal flow box next to a float has negative margins). When an inline box overlaps with a float, the content, background, and borders of the inline box are rendered in front of the float. When a block box overlaps, the background and borders of the block box are rendered behind the float and are only be visible where the box is transparent. The content of the block box is rendered in front of the float.

Here is another illustration, showing what happens when a float overlaps borders of elements in the normal flow.



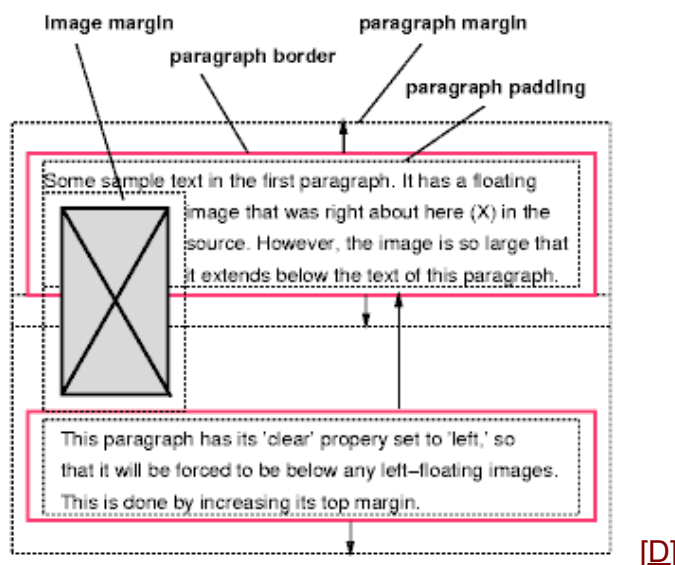
A floating image obscures borders of block boxes it overlaps.

The following example illustrates the use of the 'clear' property to prevent content from flowing next to a float.

Assuming a rule such as this:

```
P { clear: left }
```

formatting might look like this:



Both paragraphs have set 'clear: left', which causes the second paragraph to be "pushed down" to a position below the float -- its top

margin expands to accomplish this (see the ['clear'](#) property).

9.5.1 Positioning the float: the ['float'](#) property

'float'

<i>Value:</i>	left right none inherit
<i>Initial:</i>	none
<i>Applies to:</i>	all but positioned elements and generated content
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	visual

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 [absolutely positioned](#). The values of this property have the following meanings:

left

The element generates a [block](#) box that is floated to the left. Content flows on the right side of the box, starting at the top (subject to the ['clear'](#) property). The ['display'](#) is ignored, unless it has the value 'none'.

right

Same as 'left', but content flows on the left side of the box, starting at the top.

none

The box is not floated.

Here are the precise rules that govern the behavior of floats:

1. The left [outer edge](#) of a left-floating box may not be to the left of the left edge of its [containing block](#). An analogous rule holds for right-floating elements.
2. If the current box is left-floating, and there are any left floating boxes generated by elements earlier in the source document, then for each such earlier box, either the left [outer edge](#) of the current box must be to the right of the right [outer edge](#) of the earlier box, or its top must be lower than the bottom of the earlier box. Analogous rules hold for right-floating boxes.
3. The right [outer edge](#) of a left-floating box may not be to the right of the left [outer edge](#) of any right-floating box that is to the right of it. Analogous rules hold for right-floating elements.
4. A floating box's [outer top](#) may not be higher than the top of its [containing block](#).
5. The [outer top](#) of a floating box may not be higher than the outer top of any [block](#) or [floated](#) box generated by an element earlier in the source document.
6. The [outer top](#) of an element's floating box may not be higher than the top of any [line-box](#) containing a box generated by an element earlier in the source document.
7. A left-floating box that has another left-floating box to its left may not have its right outer edge to the right of its containing block's right edge. (Loosely: a left float may not stick out at the right edge, unless it is already as far to the left as possible.) An analogous rule holds for right-floating elements.
8. A floating box must be placed as high as possible.
9. 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.

9.5.2 Controlling flow next to floats: the ['clear'](#) property

'clear'

<i>Value:</i>	none left right both <u>inherit</u>
<i>Initial:</i>	none
<i>Applies to:</i>	block-level elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	<u>visual</u>

This property indicates which sides of an element's box(es) may *not* be adjacent to an earlier floating box. (It may be that the element itself has floating descendants; the 'clear' property has no effect on those.)

This property may only be specified for [block-level](#) elements (including floats). For [compact](#) and [run-in boxes](#), this property applies to the final block box to which the compact or run-in box belongs.

Values have the following meanings when applied to non-floating block boxes:

left

The top margin of the generated box is increased enough that the top border edge is below the bottom outer edge of any left-floating boxes that resulted from elements earlier in the source document.

right

The top margin of the generated box is increased enough that the top border edge is below the bottom outer edge of any right-floating boxes that resulted from elements earlier in the source document.

both

The generated box is moved below all floating boxes of earlier elements in the source document..

none

No constraint on the box's position with respect to floats.

When the property is set on floating elements, it results in a modification of the [rules](#) for positioning the float. An extra constraint (#10) is added:

- The top [outer edge](#) of the float must be below the bottom outer edge of all earlier left-floating boxes (in the case of 'clear: left'), or all earlier right-floating boxes (in the case of 'clear: right'), or both ('clear: both').

9.6 Absolute positioning

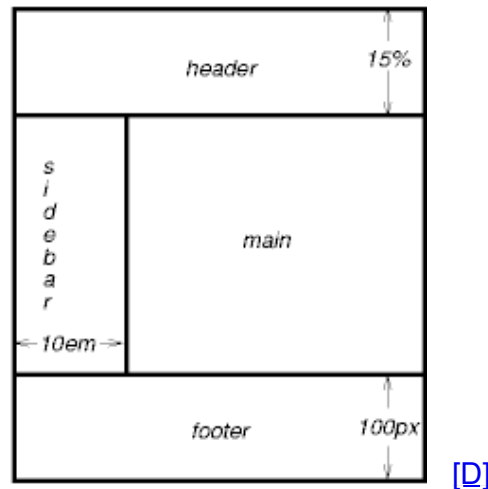
In the absolute positioning model, a box is explicitly offset with respect to its containing block. It is removed from the normal flow entirely (it has no impact on later siblings). An absolutely positioned box establishes a new containing block for normal flow children and positioned descendants. However, 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 [stack levels](#) of the overlapping boxes.

References in this specification to an *absolutely positioned element* (or its box) imply that the element's 'position' property has the value 'absolute' or 'fixed'.

9.6.1 Fixed positioning

Fixed positioning is a subcategory of absolute positioning. The only difference is that for a fixed positioned box, the containing block is established by the [viewport](#). For [continuous media](#), fixed boxes do not move when the document is scrolled. In this respect, they are similar to [fixed background images](#). For [paged media](#), 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:



[D]

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

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
  <TITLE>A frame document with CSS2</TITLE>
  <STYLE type="text/css">
    BODY { height: 8.5in } /* Required for percentage heights below */
    #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;
    }
  </STYLE>
</HEAD>
<BODY>
```

```

    #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>

```

9.7 Relationships between 'display', 'position', and 'float'

The three properties that affect box generation and layout -- 'display', 'position', and 'float' -- interact as follows:

1. If 'display' has the value 'none', user agents must [ignore](#) 'position' and 'float'. In this case, the element generates no box.
2. Otherwise, 'position' has the value 'absolute' or 'fixed', 'display' is set to 'block' and 'float' is set to 'none'. The position of the box will be determined by the 'top', 'right', 'bottom' and 'left' properties and the box's containing block.
3. Otherwise, if 'float' has a value other than 'none', 'display' is set to 'block' and the box is floated.
4. Otherwise, the remaining 'display' properties apply as specified.

Note. *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.*

9.8 Comparison of normal flow, floats, and absolute positioning

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

```

<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<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>

```

In this document, we assume the following rules:

```
BODY { display: block; line-height: 200%;
      width: 400px; height: 400px }
P     { display: block }
SPAN  { display: inline }
```

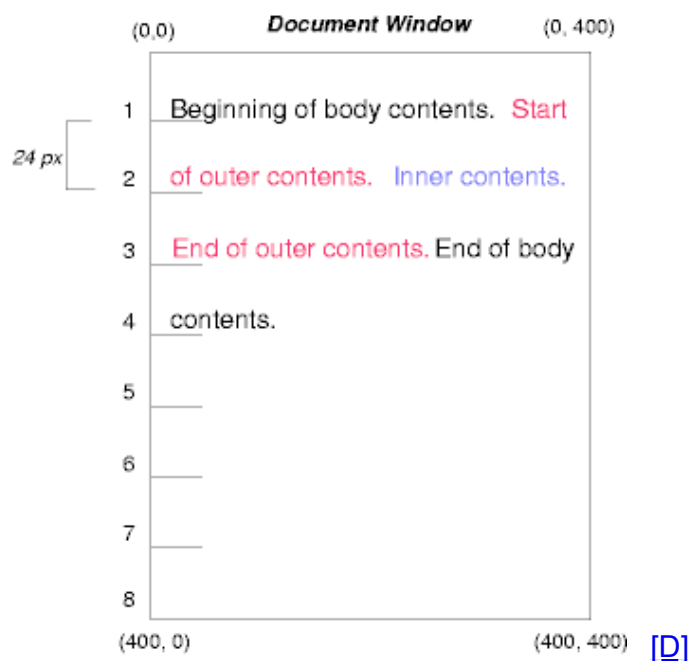
The final positions of boxes generated by the *outer* and *inner* elements vary in each example. In each illustration, the numbers to the left of the illustration indicate the [normal flow](#) position of the double-spaced (for clarity) lines. (Note: the illustrations use different horizontal and vertical scales.)

9.8.1 Normal flow

Consider the following CSS declarations for *outer* and *inner* that don't alter the [normal flow](#) of boxes:

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

The P element contains all inline content: [anonymous inline text](#) and two SPAN element. Therefore, all of the content will be laid out in an inline formatting context, within a containing block established by the P element, producing something like:



9.8.2 Relative positioning

To see the effect of [relative positioning](#), we specify:

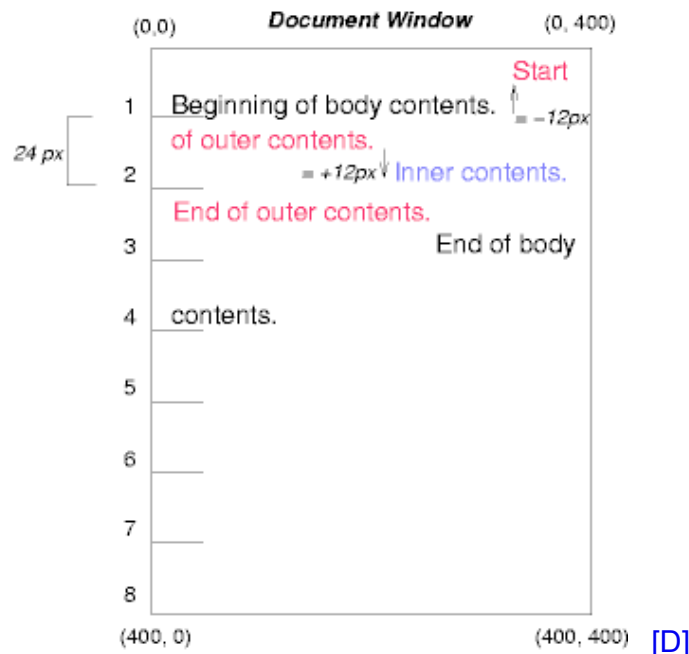
```
#outer { position: relative; top: -12px; color: red }
#inner { position: relative; top: 12px; color: blue }
```

Text flows normally up to the *outer* element. The *outer* text is then flowed into its normal flow position and dimensions at the end of line 1. Then, the inline boxes containing the text

(distributed over three lines) are shifted as a unit by '-12px' (upwards).

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*.



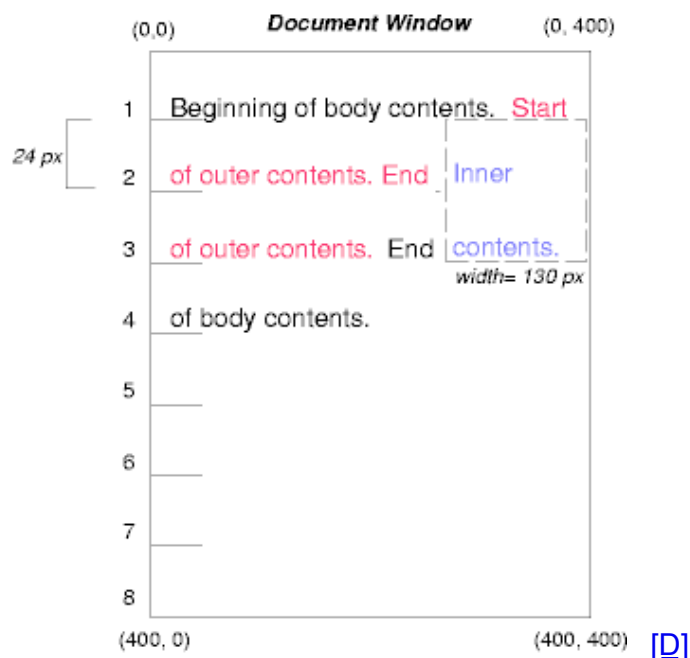
Note also that had the offset of *outer* been '-24px', the text of *outer* and the body text would have overlapped.

9.8.3 Floating a box

Now consider the effect of [floating](#) the *inner* element's text to the right by means of the following rules:

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

Text flows normally up to the *inner* box, which is pulled out of the flow and floated to the right margin (its width has been assigned explicitly). Line boxes to the left of the float are shortened, and the document's remaining text flows into them.



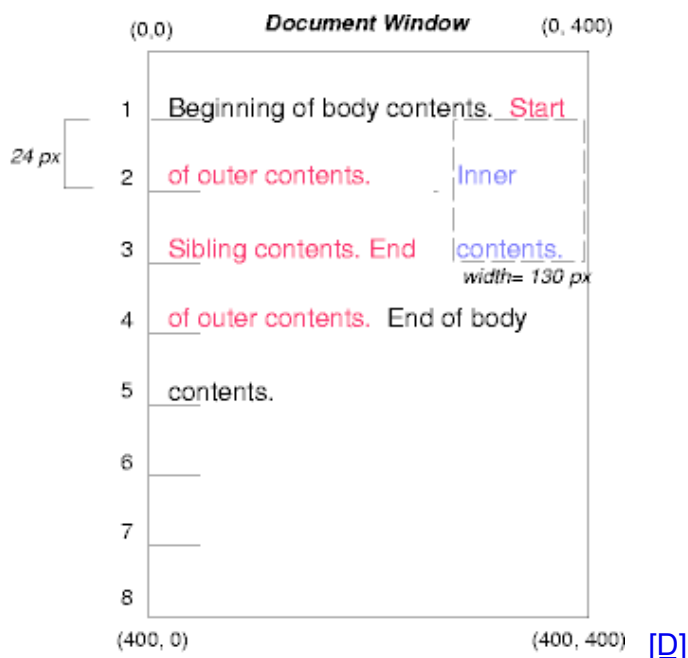
To show the effect of the 'clear' property, we add a *sibling* element to the example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<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:

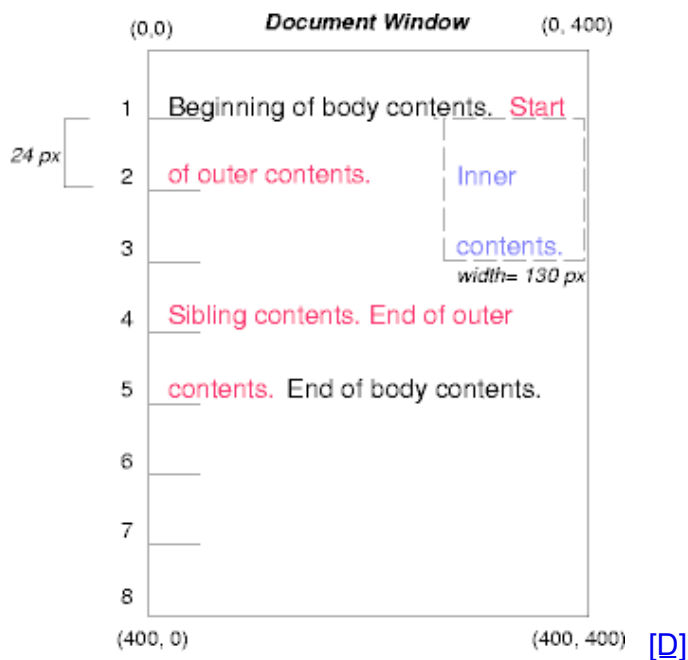
```
#inner { float: right; width: 130px; color: blue }
#sibling { color: red }
```

cause the *inner* box to float to the right as before and the document's remaining text to flow into 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 a position next to floating boxes to its right), the *sibling* content begins to flow below the float:

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

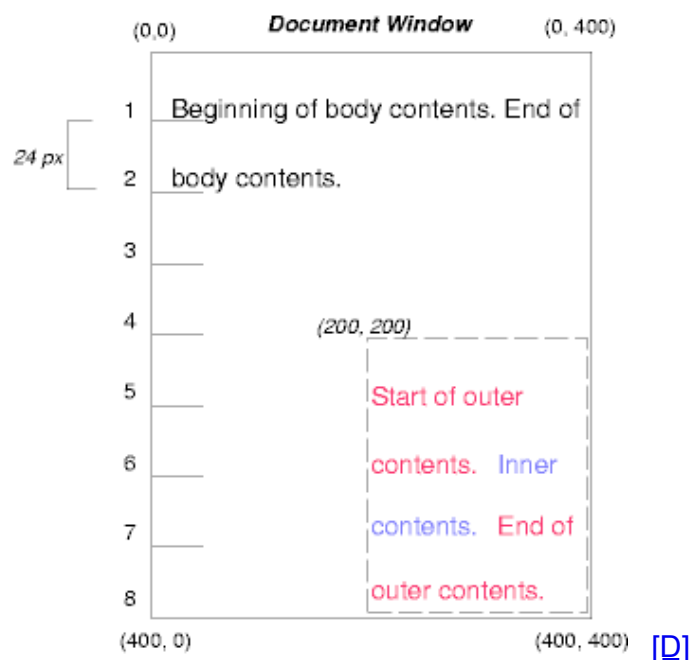


9.8.4 Absolute positioning

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

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

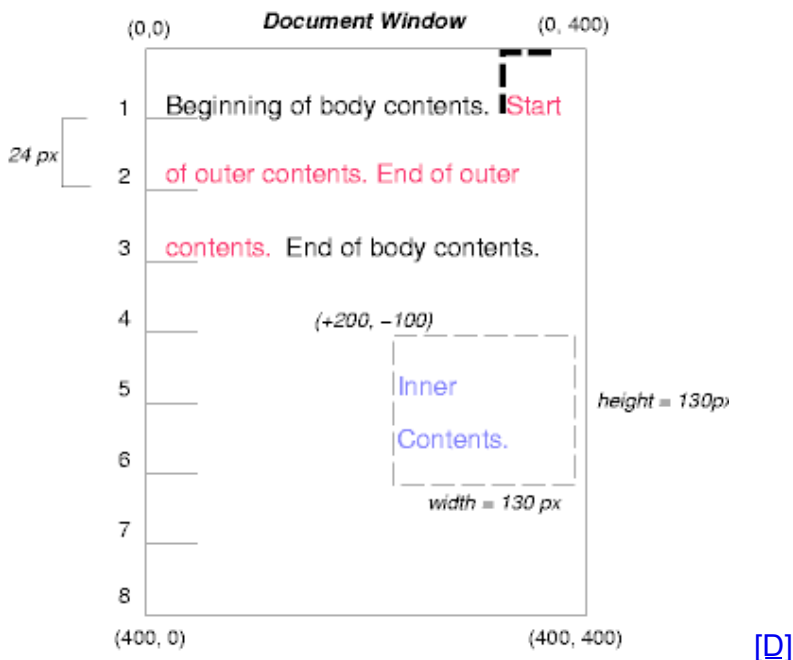
which cause the top of the *outer* box to be positioned with respect to its containing block. The containing block for a positioned box is established by the nearest positioned ancestor (or, if none exists, the [initial containing block](#), as in our example). The top side of the *outer* box is '200px' below 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.



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' means that its box may serve as the containing block for positioned descendants. Since the *outer* box is an inline box that is split across several lines, the first inline box's top and left edges (depicted by thick dashed lines in the illustration below) serve as references for 'top' and 'left' offsets.

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

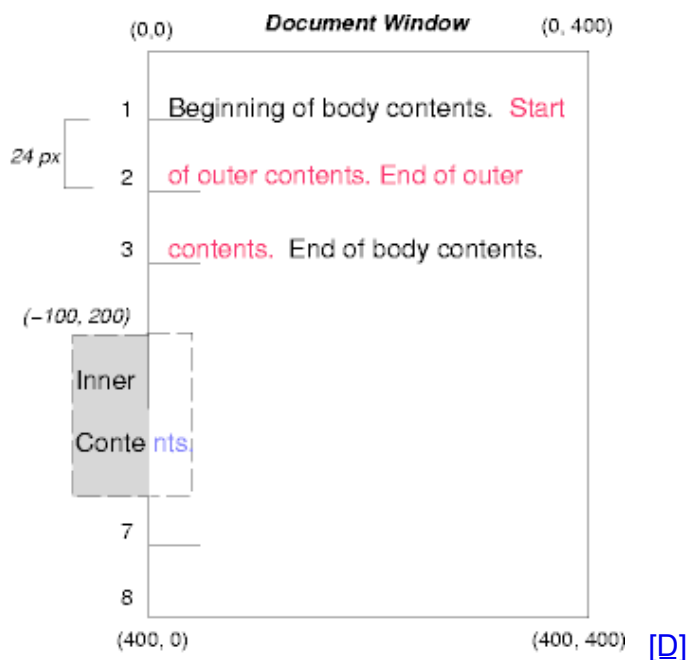
This results in something like the following:



If we do not position the *outer* box:

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

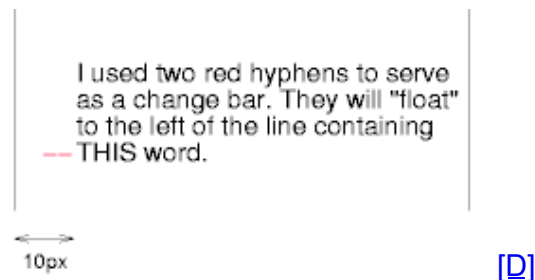
the containing block for *inner* becomes the [initial containing block](#) (in our example). The following illustration shows where the *inner* box would end up in this case.



Relative and absolute positioning may be used to implement change bars, as shown in the following example. The following document:

```
<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: -1em; color: red;">--</SPAN>
word.</P>
```

might result in something like:



First, the paragraph (whose containing block sides are shown in the illustration) is flowed normally. Then it is offset '10px' from the left edge of the containing block (thus, a right margin of '10px' has been reserved in anticipation of the offset). The two hyphens acting as change bars are taken out of the flow and positioned at the current line (due to 'top: auto'), '-1em' from the left edge of its containing block (established by the P in its final position). The result is that the change bars seem to "float" to the left of the current line.

9.9 Layered presentation

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

In CSS2, each box has a position in three dimensions. In addition to their horizontal and vertical positions, boxes lie along a "z-axis" and are formatted one on top of the other. Z-axis positions are particularly relevant when boxes overlap visually. This section discusses how boxes may be positioned along the z-axis.

Each box belongs to one *stacking context*. Each box in a given stacking context has an integer *stack level*, which is its position on the z-axis relative to other boxes in the same stacking context. Boxes with greater stack levels are always formatted in front of boxes with lower stack levels. Boxes may have negative stack levels. Boxes with the same stack level in a stacking context are stacked bottom-to-top according to document tree order.

The [root](#) element creates a *root stacking context*, but other elements may establish *local stacking contexts*. Stacking contexts are inherited. A local stacking context is atomic; boxes in other stacking contexts may not come between any of its boxes.

An element that establishes a local stacking context generates a box that has two stack levels: one for the stacking context it creates (always '0') and one for the stacking context to which it belongs (given by the '[z-index](#)' property).

An element's box has the same stack level as its parent's box unless given a different stack level with the '[z-index](#)' property.

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

'z-index'

<i>Value:</i>	auto <u><integer></u> <u>inherit</u>
<i>Initial:</i>	auto
<i>Applies to:</i>	positioned elements
<i>Inherited:</i>	no
<i>Percentages:</i>	N/A
<i>Media:</i>	<u>visual</u>

For a positioned box, the 'z-index' property specifies:

1. The stack level of the box in the current stacking context.
2. Whether the box establishes a local stacking context.

Values have the following meanings:

<integer>

This integer is the stack level of the generated box in the current stacking context. The box also establishes a local stacking context in which its stack level is '0'.

auto

The stack level of the generated box in the current stacking context is the same as its parent's box. The box does not establish a new local stacking context.

In the following example, the stack levels of the boxes (named with their "id" attributes) are: "text2"=0, "image"=1, "text3"=2, and "text1"=3. The "text2" stack level is inherited from the root box. The others are specified with the 'z-index' property.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//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 id="image" class="pile"
        src="butterfly.gif" alt="A butterfly image"
        style="z-index: 1">

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

      <DIV id="text2">
        This text will be beneath everything.
      </DIV>

      <DIV id="text3" class="pile"
        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](#).

9.10 Text direction: the 'direction' and 'unicode-bidi' properties

The characters in certain scripts are written from right to left. In some documents, in particular those written with the Arabic or Hebrew script, and in some mixed-language contexts, text in a single (visually displayed) block may appear with mixed directionality. This phenomenon is called *bidirectionality*, or "bidi" for short.

The Unicode standard ([\[UNICODE\]](#), section 3.11) defines a complex algorithm for determining the proper directionality of text. The algorithm consists of an implicit part based on character properties, as well as explicit controls for embeddings and overrides. CSS2 relies on this algorithm to achieve proper bidirectional rendering. The 'direction' and 'unicode-bidi' properties allow authors to specify how the elements and attributes of a document language map to this algorithm.

If a document contains right-to-left characters, and if the user agent displays these characters (with appropriate glyphs, not arbitrary substitutes such as a question mark, a hex code, a black box, etc.), the user agent must apply the bidirectional algorithm. This seemingly one-sided requirement reflects the fact that, although not every Hebrew or Arabic document contains mixed-directionality text, such documents are much more likely to contain left-to-right text (e.g., numbers, text from other languages) than are documents written in left-to-right languages.

Because the directionality of a text depends on the structure and semantics of the document language, these properties should in most cases be used only by designers of document type descriptions (DTDs), or authors of special documents. If a default style sheet specifies these properties, authors and users should not specify rules to override them. A typical exception would be to override bidi behavior in a user agent if that user agent transliterates Yiddish (usually written with Hebrew letters) to Latin letters at the user's request.

The HTML 4.0 specification ([\[HTML40\]](#), section 8.2) defines bidirectionality behavior for HTML elements. [Conforming](#) HTML user agents may therefore ignore the 'direction' and 'unicode-bidi' properties in author and user style sheets. The style sheet rules that would achieve the bidi behavior specified in [\[HTML40\]](#) are given in [the sample style sheet](#). The HTML 4.0 specification also contains more information on bidirectionality issues.

'direction'

<i>Value:</i>	ltr rtl <u>inherit</u>
<i>Initial:</i>	ltr
<i>Applies to:</i>	all elements, but see prose
<i>Inherited:</i>	yes
<i>Percentages:</i>	N/A
<i>Media:</i>	<u>visual</u>

This property specifies the base writing direction of blocks and the direction of embeddings and overrides (see 'unicode-bidi') for the Unicode bidirectional algorithm. In addition, it specifies the

direction of [table](#) column layout, the direction of horizontal [overflow](#), and the position of an incomplete last line in a block in case of 'text-align: justify'.

Values for this property have the following meanings:

ltr

Left-to-right direction.

rtl

Right-to-left direction.

For the '[direction](#)' property to have any effect on inline-level elements, the '[unicode-bidi](#)' property's value must be 'embed' or 'override'.

Note. The '[direction](#)' property, when specified for table column elements, is not inherited by cells in the column since columns don't exist in the document tree. Thus, CSS cannot easily capture the "dir" attribute inheritance rules described in [\[HTML40\]](#), section 11.3.2.1.

'unicode-bidi'

Value: normal | embed | bidi-override | [inherit](#)

Initial: normal

Applies to: all elements, but see prose

Inherited: no

Percentages: N/A

Media: [visual](#)

Values for this property have the following meanings:

normal

The element does not open an additional level of embedding with respect to the bidirectional algorithm. For inline-level elements, implicit reordering works across element boundaries.

embed

If the element is inline-level, this value opens an additional level of embedding with respect to the bidirectional algorithm. The direction of this embedding level is given by the '[direction](#)' property. Inside the element, reordering is done implicitly. This corresponds to adding a LRE (U+202A; for 'direction: ltr') or RLE (U+202B; for 'direction: rtl') at the start of the element and a PDF (U+202C) at the end of the element.

bidi-override

If the element is inline-level or a block-level element that contains only inline-level elements, this creates an override. This means that inside the element, reordering is strictly in sequence according to the '[direction](#)' property; the implicit part of the bidirectional algorithm is ignored. This corresponds to adding a LRO (U+202D; for 'direction: ltr') or RLO (U+202E; for 'direction: rtl') at the start of the element and a PDF (U+202C) at the end of the element.

The final order of characters in each block-level element is the same as if the bidi control codes had been added as described above, markup had been stripped, and the resulting character sequence had been passed to an implementation of the Unicode bidirectional algorithm for plain text that produced the same line-breaks as the styled text. In this process, non-textual entities such as images are treated as neutral characters, unless their '[unicode-bidi](#)' property has a value other than 'normal', in which case they are treated as strong characters in the '[direction](#)' specified for the element.

Please note that in order to be able to flow inline boxes in a uniform direction (either entirely left-to-right or entirely right-to-left), more inline boxes (including anonymous inline boxes) may have to be created, and some inline boxes may have to be split up and reordered before flowing.

Because the Unicode algorithm has a limit of 15 levels of embedding, care should be taken not to use 'unicode-bidi' with a value other than 'normal' unless appropriate. In particular, a value of 'inherit' should be used with extreme caution. However, for elements that are, in general, intended to be displayed as blocks, a setting of 'unicode-bidi: embed' is preferred to keep the element together in case display is changed to inline (see example below).

The following example shows an XML document with bidirectional text. It illustrates an important design principle: DTD designers should take bidi into account both in the language proper (elements and attributes) and in any accompanying style sheets. The style sheets should be designed so that bidi rules are separate from other style rules. The bidi rules should not be overridden by other style sheets so that the document language's or DTD's bidi behavior is preserved.

In this example, lowercase letters stand for inherently left-to-right characters and uppercase letters represent inherently right-to-left characters:

```
<HEBREW>
  <PAR>HEBREW1 HEBREW2 english3 HEBREW4 HEBREW5</PAR>
  <PAR>HEBREW6 <EMPH>HEBREW7</EMPH> HEBREW8</PAR>
</HEBREW>
<ENGLISH>
  <PAR>english9 english10 english11 HEBREW12 HEBREW13</PAR>
  <PAR>english14 english15 english16</PAR>
  <PAR>english17 <HE-QUO>HEBREW18 english19 HEBREW20</HE-QUO></PAR>
</ENGLISH>
```

Since this is XML, the style sheet is responsible for setting the writing direction. This is the style sheet:

```
/* Rules for bidi */
HEBREW, HE-QUO {direction: rtl; unicode-bidi: embed}
ENGLISH        {direction: ltr; unicode-bidi: embed}

/* Rules for presentation */
HEBREW, ENGLISH, PAR {display: block}
EMPH                 {font-weight: bold}
```

The HEBREW element is a block with a right-to-left base direction, the ENGLISH element is a block with a left-to-right base direction. The PARs are blocks that inherit the base direction from their parents. Thus, the first two PARs are read starting at the top right, the final three are read starting at the top left. Please note that HEBREW and ENGLISH are chosen as element names for explicitness only; in general, element names should convey structure without reference to language.

The EMPH element is inline-level, and since its value for 'unicode-bidi' is 'normal' (the initial value), it has no effect on the ordering of the text. The HE-QUO element, on the other hand, creates an embedding.

The formatting of this text might look like this if the line length is long:

```
5WERBEH 4WERBEH english3 2WERBEH 1WERBEH
8WERBEH 7WERBEH 6WERBEH
english9 english10 english11 13WERBEH 12WERBEH
```

english14 english15 english16

english17 20WERBEH english19 18WERBEH

Note that the HE-QUO embedding causes HEBREW18 to be to the right of english19.

If lines have to be broken, it might be more like this:

2WERBEH 1WERBEH
-EH 4WERBEH english3
5WERB

-EH 7WERBEH 6WERBEH
8WERB

english9 english10 en-
glish11 12WERBEH
13WERBEH

english14 english15
english16

english17 18WERBEH
20WERBEH english19

Because HEBREW18 must be read before english19, it is on the line above english19. Just breaking the long line from the earlier formatting would not have worked. Note also that the first syllable from english19 might have fit on the previous line, but hyphenation of left-to-right words in a right-to-left context, and vice versa, is usually suppressed to avoid having to display a hyphen in the middle of a line.

[previous](#) [next](#) [contents](#) [properties](#) [index](#)