Lab 02 CSS - Cascade Style Sheet

Version: 0.1
Author: Ovidiu Drugan

Table of contents

Table of contents 1
Introduction 3
Definition Methods 5
  Declaration 5
  Style sheets inside style attributes 5
  Linked and embeded style sheets 5
Units, colors and keywords 8
  Units and keyworks 8
  Colors 8
Selectors 10
  Syntax 10
    Grouping 10
  Basic selectors and grouping 10
    Universal selectors 10
    Element type selectors 11
    Descendant selectors 11
    Child selectors 11
    Adiacent sibling selector 11
  Class selector, attribute selectors and id selectors 11
    Class and attribute selectors 11
    Id selector 12
  Pseudo-classes and pseudo-elements 13
    Pseudo-classes 14
    Pseudo-elements 15
Assigning Property Values, Cascading and Inheritance 16
  Specified, computed, and actual values 16
    Specified values 16
    Computed values 16
    Actual values 16
  Inheritance 16
  The @import rule 17
  The cascade 17
Media types 18
  Media-dependent style sheets 18
  Recognized media types 18
Box model 19
  Dimension 19
  Margin properties 20
  Padding properties 21
  Border properties 22
    Border width 22

Border color 23
Border style 23
Border shorthand properties 24
Visual formatting model 26
Visual effects 27
Overflow and clipping 27
The overflow property 27
The clipping property 28
Visibility 28
Generated content, automatic numbering, and lists 30
The :before and :after pseudo-elements 30
The ‘content’ property 30
Quotation marks 31
Automatic counters 32
Nested counters and scope 32
Counter styles 33
Counters in elements with display: none 34
Lists 34
Paged media 37
Colors and Backgrounds 38
Foreground color 38
The background 38
Properties 39
Fonts 43
Matching Algorithm 43
Properties 43
Text 48
Indentation 48
Alignment 48
Decoration 48
Letter and word spacing 49
Capitalization 50
Tables 51
The CSS table model 51
Column selectors 51
Tables in the visual formatting model 52
Visual layout of table contents 52
Table layers and transparency 53
Table width 53
Table height 54
Borders 54
Collapsing border model 55
Borders styles 56
Bibliography 57
Appendix 58

Introduction

The basic idea is to separate the presentation from the content. CSS works to assist the display of HTML. CSS means the following (I handle terms in logical order):

- *S*tyle = special style declarations and rules used to specify the presentation of elements (I handle them later thoroughly).
- *S*tyle *S*heets = style declarations and rules are collected into sheets. A single style sheet is the location of one or more real or logical rules. These rules specify the presentation of a document (the specifications have some inaccurate definitions, which I handle in the Footnote 1).
- *C*ascading = style sheets can be connected and cascaded - thus the reference to *C*ascading *S*tyle *S*heets.

CSS combines style declarations and rules from different sources. That's why the term is constructed as a plural (sheets). Styles have different weights compared to each others.

CSS is not however an independent language, because it can only be used to assist other languages (primarily HTML, XML and XSL). The term "formatting" refers primarily to visual formatting, but CSS can be used to format documents in other ways as well. CSS exists solely for the formatting of documents.

CSS is similar to scripting languages like JavaScript and ECMAScript, in that it can be used as small fragments inside the main document. Also, CSS syntax has some similarities to these scripting languages even if CSS is not a programming language.

CSS1 is a simple style sheet mechanism that allows authors and readers to attach style (e.g. fonts, colors and spacing) to HTML documents. It is not an actual layout language but rather very limited formatting language.

Instead CSS2 can be call as an almost fully featured layout language, because it can create most layout features, which are needed in web documents. Indeed CSS2 has some missing features. These are added into CSS3).

If a browser were to have full implementation of CSS2, it would at change the character of HTML documents at a fundamental level because the formatting features of most HTML elements could be redefined. Then HTML itself would not be used just at all as a formatting language. CSS cannot, however, exchange the real order of elements (it can hide elements from the screen or reposition them - how it happens, I will explain later).

With it's methods of formatting and layout, CSS aims to:

1. separate presentation from content.
2. partially separate structure from content. CSS doesn't separate these two completely, because CSS can't exchange the real order of elements.
3. give a more exact and much richer presentation to elements as was done in HTML 3.2.
4. assist in making files more degradable.
5. give basic documents simple structure.
6. allow one document to have a different presentation in different devices. It is even possible to make aural style sheets for speech synthesizer and raised letter versions to braille and emboss device and create tactile media.
7. be used in a variety of documents, including XML documents.

HTML 4.0 with CSS offers the possibility to return to the original idea of HTML - to go back to the roots! Extreme HTML 4.0 documents don't have any presentational attributes. All elements and attributes concern structure and linking and possibly embedded elements. Elements and attributes, which only describe presentation are not used (for example, elements FONT, B, I, U

and attributes align, bgcolor). Some of them (for example B) are allowed, but not recommended, because the meaning of elements is primary to describe the structure and semantics, not the presentation.

All possible definitions that affect document presentation are commonly linked into separate files using format `<LINK rel="stylesheet" type="text/css" href="stylesheet.css">` as in this example document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<LINK rel="stylesheet" type="text/css" href="stylesheet.css">
<TITLE>Title text</TITLE>
</HEAD>
<BODY>
<H2>The header</H2>
<P>The first paragraph</P>
<P>Second paragraph. <A href="html_document.html">Link</A >. The
only necessary attributes are href and possible linked objects
like images.</P>
</BODY>
</HTML>
```

Definition Methods

The basic idea of HTML 4.0 and CSS is to separate presentation and contents so, that the whole site would be easy to modify afterwards.

Declaration

Style sheets are like collections of presentational attributes. In that mean CSS works at attribute level. Presentational definitions inside style sheets are not however called as attributes but *properties.*

One CSS-property can replace several HTML-attributes. There is also situations, where replacing one HTML-attribute must define several CSS-properties even so, that they must define for several elements.

Properties create *declarations*, which means **one property with its value or a value set**.

Structure of the declaration:

```
name {property1: value1; property2 value2; … }
```

or

```
name {property: value1 value2 …;}
```

Example:

```
body
{background-color: white;
background-image: url(background_image.gif);
background-repeat: repeat-y;
background-position: 2px 0px;}

body {background: white url(background_image.gif) repeat-y 2px
0px;}
```

Style sheets inside style attributes

Stile sheets can be defined at the level of HTML elememnt. If style sheets have been used inside elements, thy have been defined inside style attributes. In this connection declaration blocks don't have curly brackets. This method is called also as using *inline style sheets* or just *inline styles*.

Example:

```
<TD style="border:1px solid #660033">
```

The usage of *inline style sheets* is **not recommended**.

Linked and embeded style sheets

It is always sensible to create external CSS-files, if you have three or more documents, where you want to use CSS-properties. **Define to the separate file basic styles, which you want to use in your whole site.** They are caled: *site-level style sheets*.

Properties in external files have the same function as style attributes.

Example

```
p {color:blue; border:1px solid blue;}
```

has the same effect as

```
<P style="color:blue; border:1px solid blue;">.
```

The latter definition is not however given directly to the element, but indirectly using an external file.

Properties inside a style attribute concerns only one element .

When you use external files, give to the file name like basic_stylesheet.css and link it to your HTML-document.

```
<LINK rel="stylesheet" type="text/css" href="styleSheetFile.css">
```

The code must be in the HEAD-part of the document. You can link to your own folders by using relative references or use for example *core stylesheets* of W3C.

```
<LINK rel="stylesheet" type="text/css"
href="http://www.w3.org/stylesheets/Core/Modernist">
```

Embedded style sheets mean sheets, which are "embedded" to the same document before the element BODY between <HEAD></HEAD> tags using the element STYLE. We can call them as document-level style sheets.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0 Transitional//EN">
<HTML>
<HEAD>
<TITLE>Title</TITLE>
<LINK rel="stylesheet" type="text/css"
href="external_stylesheet.css">
<STYLE type="text/css">
<!-- /* these definitions are inside HTML comments that's why
that old browsers could pass them */
body {background: yellow url(new_image.gif) no-repeat center;} /*
I explain in other connection, how to define background
properties */
-->
</STYLE>
</HEAD>
<BODY>
```

By using embedded or external style sheets it is possible to use the the import rule:

```
@import url( …);
```

```
<LINK rel="stylesheet" type="text/css" href="stylesheet.css">
<STYLE type="text/css">
<!--
@import url(external_stylesheet.css); /* note that at the end of
the rule is the sign ; - without this sign the browser doesn't
handle further definitions */
/* definitions, which belongs only to one document */
body
{font-family: Verdana, Arial, Helvetica, sans-serif; font-size:x-
small;
padding-top: 10px;
padding-left: 32px;
padding-right: 10px;
padding-bottom: 10px;
margin: 0px;}
h2
```

```
{color: #660033;font-family:Verdana, Arial, Helvetica, sans-
serif;} /* remember, that this definition doesn't cancel imported
definitions in other parts than again definite properties */
td
{font-family: Verdana, Arial, Helvetica, sans-serif; font-size:x-
small;}
-->
</STYLE>
```

Remember always, when you use the import rule, that the import rule is handled before other definitions. If there are several import rules they are handled in order starting from top. The basic idea of the import rule is that the browser reads imported style sheets as if they have been directly written in the place, where the import rules exist.

Units, colors and keywords

Units and keyworks

In the size definitions of CSS-properties use either numeric units or certain keywords, which replace numeric units. Keywords are different to various properties and I handle in this page only common units and and keywords to font definitions.

Text:

- xx-small,
- x-small,
- small,
- medium,
- large,
- x-large,
- xx-large

Example:

```
{font-size:x-small;}
```

Concerning numerical units, some of them are **relative** and some **absolute**.

Relative units:
The unit *em* is relative to the default font size of the surrounding element. The same kind of unit is *ex*, and *px* which the specification says:

- em: the ‘font-size’ of the relevant element
- ex: the 'x-height' of the relevant font.
- px: pixels, relative to the viewing device

```
h1 { margin: 0.5em }      /* em */
h1 { margin: 1ex }        /* ex */
p  { font-size: 12px }    /* px */
```

Absolute units:
It is possible to get better control by using absolute units. Following units are according to the CSS2 specification absolute (possible decimals are separated with a dot):

- in = inches
- cm = centimeters
- mm = millimeters
- pt = points - 1 point = 1/72 inch = 0,353 mm (printed, but not necessary on the screen, where it might be for example 1/96 depending the platform and screen resolution).
- pc = pica - 1 pica = 12 points = 4,24mm

```
h1 { margin: 0.5in }      /* inches  */
h2 { line-height: 3cm }   /* centimeters */
h3 { word-spacing: 4mm }  /* millimeters */
h4 { font-size: 12pt }    /* points */
h4 { font-size: 1pc }     /* picas */
```

Colors

It is possible to give colors with many ways in the following case:

- Named colors as keywords (for example strong {color:red;}).

However that even common used 16 color names (`aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white and yellow`) are platform dependent.

- So-called hexadecimal values, which have in front of the definition sign # like p.special {color:#070e0e;}. Six digits' hexadecimal values correspond 24 bits (about 16,7 million colors) RGB-values (red-green-blue).

```
em {color: rgb(255,0,0) } /* RGB colors 0-255, it is possible to
use always three digits, in this case 255,000,000 */
em {color: rgb(100%,0%,0%) } /* RGB colors 0-255 with nearest
percentage values - note, that this system is not as exact as
previous method */
em { color: red; } /* corresponding keyword */
```

Selectors

The basic method on CSS is to apply CSS-properties to elements according to certain principles. The browser must be able to identify elements. Excluding direct style attributes, the browser creates from element identifiers like patterns. That's why in CSS is written about pattern matching. In CSS and and documents, which use CSS must have matching pairs.

Selectors have different weights compared to each others, so they create the **cascading order**.

Syntax

A simple selector is either a type selector or universal selector followed immediately by zero or more attribute selectors, ID selectors, or pseudo-classes, in any order. The simple selector matches if all of its components match.

A selector is a chain of one or more simple selectors separated by combinator's. Combinator's are: white-space, ">", and "+".

The elements of the document tree that match a selector are called subjects of the selector. A selector consisting of a single simple selector matches any element satisfying its requirements.

Rule:

```
selector { property: value; … }
```

Grouping

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

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

```
h1 { font-family: sans-serif }
h2 { font-family: sans-serif }
h3 { font-family: sans-serif }
```

is equivalent to:

```
h1, h2, h3 { font-family: sans-serif }
```

Basic selectors and grouping

Universal selectors

Using the universal selector means, that defined properties are applied to any element. The browser doesn't try to find any specific element from the document. It just finds all possible elements, which CSS can be applied. Universal selectors have the lowest priority from the selectors

The universal selector, written "*", matches the name of any element type. It matches any single element in the document tree.

If the universal selector is not the only component of a simple selector, the "*" may be omitted. For example:

```
*[lang=fr] and [lang=fr] are equivalent.
*.warning and .warning are equivalent.
*#myid and #myid are equivalent.
```

Element type selectors

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

Example:

```
h1 { font-family: sans-serif }
```

Descendant selectors

The selectors are able to match an element that is the descendant of another element in the document tree if the used descendant selectors.

```
h1 { color: red }
em { color: red }
h1 em { color: blue }
```

Child selectors

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

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

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

Adiacent sibling selector

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

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

```
math + p { text-indent: 0 }
```

Class selector, attribute selectors and id selectors

The matching criterion can be specified by using class and *id* (= identification) attributes and corresponding selectors: *class selector* or *id selector*.

The class selector is designed to many elements in the same document. The basic idea of using class selectors is to give the possibility to avoid the limitation of element type identifiers (element names). Using class selectors you can create as many "new elements" as you need.

You can give quite freely names to class and id-selectors, but there is however some limitations: The name must start with a normal letter (a-z,A-Z). Accepting the underscore (_) has been afterward added into CSS2

Class and attribute selectors

Authors can specify rules that match attributes defined in the source document.

Attribute selectors

Attribute selectors may match in four ways:

- [att] - Match when the element sets the "att" attribute, whatever the value of the attribute.
- [att=val] - Match when the element's "att" attribute value is exactly "val".
- [att~=val] - Match when the element's "att" attribute value is a space-separated list of "words", one of which is exactly "val". (the words in the value must not contain spaces).

- [att|=val] - Match when the element's "att" attribute value is a hyphen-separated list of "words", beginning with "val". The match always starts at the beginning of the attribute value.

Attribute values must be identifiers or strings. The case-sensitivity of attribute names and values in selectors depends on the document language.

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

```
h1[title] { color: blue; }
```

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

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

Multiple attribute selectors can be used to refer to several attributes of an element, or even several times to the same attribute.

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

The following selectors illustrate the differences between "=" and "~=". The first selector will match, for example, the value "copyright copyleft copyeditor" for the "rel" attribute. The second selector will only match when the "href" attribute has the value "http://www.w3.org/".

```
a[rel~="copyright"]
a[href="http://www.w3.org/"]
```

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

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

Class selectors

Working with HTML, authors may use the period `(.)` notation as an alternative to the `~=` notation when representing the class attribute. Thus, for HTML, `div.value` and `div[class~=value]` have the same meaning. The attribute value must immediately follow the "period" `(.)`.

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

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

or just

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

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

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

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

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

Id selector

Document languages may contain attributes that are declared to be of type ID. What makes attributes of type ID special is that no two such attributes can have the same value; whatever the

document language, an ID attribute can be used to uniquely identify its element. In HTML all ID attributes are named "id"; XML applications may name ID attributes differently, but the same restriction applies.

The ID attribute of a document language allows authors to assign an identifier to one element instance in the document tree. CSS ID selectors match an element instance based on its identifier. A CSS ID selector contains a "#" immediately followed by the ID value.

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

```
h1#chapter1 { text-align: center }
```

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

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

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

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

ID selectors have a higher specificity than attribute selectors.

Pseudo-classes and pseudo-elements

These classifications are created, because any definitions don't match to any particular part of the document tree. That's why is created two classifications, which are not depending on the document tree at the same way as the element type selector. Pseudo-classes and pseudo-elements are assistance concepts. They behave like class or element type selectors. Because they don't use the class attribute or they don't have corresponding elements, they are in a way fictional. That's why they are called as pseudo-classes and pseudo-elements.

Pseudo-class is an assistant concept, which gives the possibility to **classify elements like by using classes**, but in some cases relating to outstanding features of the HTML-document (for example if the link has been followed or not).

Pseudo-element is an assistant concept, which is related to the **presentational sub-part of the elements** (like the first letter or row) resembling the way, how CSS is related with element type selectors.

Neither pseudo-elements nor pseudo-classes appear in the document source or document tree.

Pseudo-classes are allowed anywhere in selectors while pseudo-elements may only appear after the subject of the selector.
Pseudo-element and pseudo-class names are case-insensitive.

Pseudo-classes

Simple element type selectors are not tied with any particular attribute inside the element. Pseudo-classes are always bound with real or fictional attributes basically exactly at the same mechanism as an element is bound with the class attribute (remember that class selectors are also bound with certain attributes). Pseudo-classes offer the possibility to express the same element differently in various states.

Especially links can have several dictions in different states and you find pseudo-classes used commonly in link presentations. Pseudo-classed `:link` and `:visited` gives the possibility to express the states of links without using HTML-attributes.

Link related pseudo-classes are link pseudo-classes `:link` and `:visited` and dynamic link pseudo-classes `:hover`, `:active` and `:focus`. Dynamic pseudo-classes affect wider than link pseudo-classes. They concern also other elements than A and they are related only to different states of elements.

:first-child pseudo-class
The `:first-child` pseudo-class matches an element that is the first child of some other element.

```
p:first-child em { font-weight : bold }
```

:link and :visited pseudo-class
User agents commonly display unvisited links differently from previously visited ones. CSS provides the pseudo-classes ':link' and ':visited' to distinguish them:

- The :link pseudo-class applies for links that have not yet been visited.
- The :visited pseudo-class applies once the link has been visited by the user.

```
a:link { color: red}
a.external:visited {color: blue}
```

:hover, :active and :focus dinamic pseudo-classes
Interactive user agents sometimes change the rendering in response to user actions. CSS provides three pseudo-classes for common cases:

- The `:hover` pseudo-class applies while the user designates an element (with some pointing device), but does not activate it.
- The :active pseudo-class applies while an element is being activated by the user.
- The :focus pseudo-class applies while an element has the focus (accepts keyboard events or other forms of text input).

An element may match several pseudo-classes at the same time.

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

:lang language pseudo-class
If the document language specifies how the human language of an element is determined, it is possible to write selectors in CSS that match an element based on its language.

```
html:lang(fr) { quotes: '« ' ' »' }
html:lang(de) { quotes: '»' '«' '\2039' '\203A' }
:lang(fr) > Q { quotes: '« ' ' »' }
:lang(de) > Q { quotes: '»' '«' '\2039' '\203A' }
```

Pseudo-elements

Pseudo-elements don't use any special attributes as their identifiers. They work together with other selectors. If pseudo-elements concern only a certain element type it is not necessary to add to them other selectors. Because in CSS2 `:first-letter` and `:first-line` concern any block level elements it is necessary to define also the element type, which they concern.

:first-line pseudo-element

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

```
p.first-list { text-transform: upercase }
```

:first-letter pseudo-element

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

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

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

:before and :after pseudo-elements

They can be used to insert generated content before or after an element's content.

```
h1:before {content: counter(chapno, upper-roman) ".”}

p.special:before {content: “Special! “}
p.special:first-letter {color: #ffd800}
```

Assigning Property Values, Cascading and Inheritance

Specified, computed, and actual values

Once a user agent has parsed a document and constructed a document tree, it must assign, for every element in the tree, a value to every property that applies to the target media type.
The final value of a property is the result of a three-step calculation: the value is determined through specification (the "specified value"), then resolved into an absolute value if necessary (the "computed value"), and finally transformed according to the limitations of the local environment (the "actual value").

Specified values

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

- If the cascade results in a value, use it.
- Otherwise, if the property is inherited and the element is not the root of the document tree, use the computed value of the parent element.
- Otherwise use the property's initial value. The initial value of each property is indicated in the property's definition.

Since it has no parent, the root of the document tree cannot use values from the parent element; in this case, the initial value is used if necessary.

Computed values

Specified values may be absolute (i.e., they are not specified relative to another value, as in 'red' or '2mm') or relative (i.e., they are specified relative to another value, as in 'auto', '2em', and '12%'). For absolute values, no computation is needed to find the computed value.

Relative values, on the other hand, must be transformed into computed values: percentages must be multiplied by a reference value (each property defines which value that is), values with relative units (em, ex, px) must be made absolute by multiplying with the appropriate font or pixel size, 'auto' values must be computed by the formulas given with each property, certain keywords ('smaller', 'bolder', 'inherit') must be replaced according to their definitions.

When the specified value is not 'inherit', the computed value of a property is determined as specified by the Computed Value line in the definition of the property. See the section on inheritance for the definition of computed values when the specified value is 'inherit'.

The computed value exists even when the property doesn't apply, as defined by the 'Applies To' [add reference] line. However, some properties may define the computed value of a property for an element to depend on whether the property applies to that element.

Actual values

A computed value is in principle ready to be used, but a user agent may not be able to make use of the value in a given environment. For example, a user agent may only be able to render borders with integer pixel widths and may therefore have to approximate the computed width. The actual value is the computed value after any approximations have been applied.

Inheritance

Some values are inherited by the children of an element in the document tree. Each property defines if it is inherited or not.
When inheritance occurs, elements inherit computed values. The computed value from the parent element becomes both the specified value and the computed value on the child.

```
body { font-size: 10pt }
h1 { font-size: 120% }
```

Each property may also have a specified value of 'inherit', which means that, for a given element, the property takes the same computed value as the property for the element's parent. The 'inherit' value can be used to strengthen inherited values, and it can also be used on properties that are not normally inherited.

```
body {
  color: black !important;
  background: white !important;
}

* {
  color: inherit !important;
  background: transparent !important;
}
```

The @import rule

The '*@import'* rule allows users to import style rules from other style sheets. Any @import rules must precede all rule sets in a style sheet. The '@import' keyword must be followed by the URI of the style sheet to include. A string is also allowed; it will be interpreted as if it had *url(...)* around it.

```
@import "mystyle.css";
@import url("mystyle.css");
```

The cascade

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

- **Author**. The author specifies style sheets for a source document according to the conventions of the document language.
- **User**: The user may be able to specify style information for a particular document.
- **User agent**: Conforming user agents must apply a default style sheet (or behave as if they did) prior to all other style sheets for a document. Style sheets from these three origins will overlap in scope, and they interact according to the cascade.

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

By default, rules in author style sheets have more weight than rules in user style sheets. Precedence is reversed, however, for "!important" rules. All user and author rules have more weight than rules in the UA's default style sheet.

Rules specified in a given style sheet override rules of the same weight imported from other style sheets. Imported style sheets can themselves import and override other style sheets, recursively, and the same precedence rules apply.

Media types

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

Media-dependent style sheets

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

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

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

- Specify the target medium within the document language.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
   <HEAD>
      <TITLE>Link to a target medium</TITLE>
      <LINK REL="stylesheet" TYPE="text/css"
       MEDIA="print, handheld" HREF="foo.css">
   </HEAD>
   <BODY>
      <P>The body...
   </BODY>
</HTML>
```

Recognized media types

The names chosen for CSS media types reflect target devices for which the relevant properties make sense. The names of media types are normative. In the following list of CSS media types, the parenthetical descriptions are not normative. Likewise, the "Media" field in the description of each property is informative.

- **all** - Suitable for all devices.
- **braille** - Intended for braille tactile feedback devices.
- **embossed** - Intended for paged braille printers.
- **handheld** - Intended for handheld devices (typically small screen, limited bandwidth).
- **print** - Intended for paged material and for documents viewed on screen in print preview mode.
- **projection** - Intended for projected presentations, for example projectors. Please consult the section on paged media for information about formatting issues that are specific to paged media.
- **screen** - Intended primarily for color computer screens.
- **speech** (aural in CSS2)- Intended for speech synthesizers.
- **tty** - Intended for media using a fixed-pitch character grid (such as teletypes, terminals, or portable devices with limited display capabilities). Authors should not use pixel units with the "tty" media type.
- **tv** - Intended for television-type devices (low resolution, color, limited-scrollability screens, sound available).

Media type names are case-insensitive

Box model

Describe rectangular boxes which are generated for elements in the document tree and laid out to the visual formatting model.

Dimension

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 margin, border, and padding can be broken down into top, right, bottom, and left 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.
- 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 top, right, bottom, and left 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 width - given by the sum of the left and right margins, border, and padding, and the content width.
- box height - given by the sum of the top and bottom margins, border, and padding, and the content height.
- background style of the content, padding, and border areas of a box is specified by the 'background' property of the generating element. Margin backgrounds are always transparent.

Example:

```
<!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: yellow;
        margin: 12px 12px 12px 12px;
        padding: 3px 3px 3px 3px;
                                    /* No borders set */
      }
      LI {
        color: white;               /* text color is white */
```

```
        background: blue;            /* Content, padding will be
blue */
        margin: 12px 12px 12px 12px;
        padding: 12px 0px 12px 12px; /* Note 0px padding right */
        list-style: none             /* no glyphs before a list
item */
                                     /* No borders set */
      }
      LI.withborder {
        border-style: dashed;
        border-width: medium;        /* sets border width on all
sides */
        border-color: lime;
      }
    </STYLE>
  </HEAD>
  <BODY>
    <UL>
      <LI>First element of list
      <LI class="withborder">Second element of list is longer
           to illustrate wrapping.
    </UL>
  </BODY>
</HTML>
```

Margin properties

Margins 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. These properties apply to all elements, but vertical margins will not have any effect on non-replaced inline elements.

<**margin-width**> can be:

- <**length**>: fixed width
- <**percentage**>: with respect to the width of the generated box

auto

'margin-top', 'margin-bottom'

Value:	<margin-width> | inherit
Initial:	0
Applies to:	all elements but inline, non-replaced elements
Inherited:	no
Percentages:	refer to width of containing block
Media:	visual
Computed value:	absolute length

'margin-right', 'margin-left'

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

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:	see individual properties
Applies to:	all elements
Inherited:	no
Percentages:	refer to width of containing block
Media:	visual
Computed value:	see individual properties

Examples:

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

Padding properties

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.

<padding-width> value type:

- <**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'.

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

Value:	<padding-width> | inherit
Initial:	0
Applies to:	all elements
Inherited:	no
Percentages:	refer to width of containing block
Media:	visual
Computed value:	absolute length

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

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

'padding'

Value:	<padding-width>{1,4} | inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentages:	refer to width of containing block
Media:	visual
Computed value:	see individual properties

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.

```
h1 {
  background: white;
  padding: 1em 2em;
}
```

Border properties

The border properties specify the width, color, and style of the border area of a box.

Border width

The border width properties specify the width of the border area.
<border-width> value type:

- thin - a thin border.
- medium - a medium border.
- thick - a thick border.
- <length> - the border's thickness has an explicit value.

'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
Computed value:	absolute length; '0' if the border style is 'none' or 'hidden'

'border-width'

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

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

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

```
h1 { border-width: thin }              /* thin thin thin thin */
h1 { border-width: thin thick }        /* thin thick thin thick */
h1 { border-width: thin thick medium }/* thin thick medium thick
*/
```

Border color

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

<border-color> value types:

- <color> - specifies a color value.
- transparent - the border is transparent (though it may have width).

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

Value:	<color> | transparent | inherit
Initial:	the value of the 'color' property
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	when taken from the 'color' property, the computed value of 'color'; otherwise, as specified

'border-color'

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

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

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

Border style

The border style properties specify the line style of a box's border (solid, double, dashed, etc.).
<border-style> value type:

- none - no border.
- 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 box look as though it were embedded in the canvas.
- outset - the opposite of 'inset': the border makes the 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 border color properties, but UA's may choose their own algorithm to calculate the actual colors used.

'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
Computed value:	as specified

'border-style'

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

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 }
```

Border shorthand properties

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

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

'border'

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

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
}

blockquote {
  border-color: red;
  border-left: double;
  color: black
}
```

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.

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.

Note: For more information on this mater please se the CSS 2.1 specification.

Visual effects

Overflow and clipping

Generally, the content of a block box is confined to the content edges of the box. In certain cases, a box may overflow, meaning its content lies partly or entirely outside of the box, e.g.:

- A line cannot be broken, causing the line box to be wider than the block box.
- A block-level box is too wide for the containing block.
- An element's height exceeds an explicit height assigned to the containing block.
- A descendent box is positioned absolutely, partly outside the box.
- A descendent box has negative margins, causing it to be positioned partly outside the box.

Whenever overflow occurs, the 'overflow' property specifies whether a box is clipped to its content box, and if so, whether a scrolling mechanism is provided to access any clipped out content.

The overflow property

This property specifies whether the content of a block-level element is clipped when it overflows the element's box (which is acting as a containing block for the content).
<overflow> value types:

- visible - this value indicates that content is not clipped
- hidden - this value indicates that the content is clipped and that no scrolling mechanism should be provided to view the content outside the clipping region;
- scroll - this value indicates that the content is clipped and that if the user agent uses a scrolling mechanism that is visible on the screen (such as a scroll bar or a panner), that mechanism should be displayed for a box whether or not any of its content is clipped.
- auto - the behavior of the 'auto' value is user agent-dependent, but should cause a scrolling mechanism to be provided for overflowing boxes.

'overflow'

Value:	visible | hidden | scroll | auto | inherit
Initial:	visible
Applies to:	block-level and replaced elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	as specified

Even if 'overflow' is set to 'visible', content may be clipped to a UA's document window by the native operating environment.

Example:

```
<DIV>
<BLOCKQUOTE>
<P>I didn't like the play, but then I saw
it under adverse conditions - the curtain was up.
<CITE>- Groucho Marx</CITE>
</BLOCKQUOTE>
</DIV>
```

Here is the style sheet controlling the sizes and style of the generated boxes:

```
div { width : 100px; height: 100px;
      border: thin solid red;
      }
```

```
blockquote   { width : 125px; height : 100px;
      margin-top: 50px; margin-left: 50px;
      border: thin dashed black
      }

cite { display: block;
       text-align : right;
       border: none
       }
```

The clipping property

A clipping region defines what portion of an element's border box is visible
The 'clip' property applies only to absolutely positioned elements.
<clip> - value types:

- auto - the clipping region has the same size and location as the element's box(es).
- <shape> - it can be: rect (<top>, <right>, <bottom>, <left>); <top>, <right>, <bottom>, and <left> may either have a <length> value or 'auto'.

'clip'

Value:	<shape> | auto | inherit
Initial:	auto
Applies to:	absolutely positioned elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	For rectangle values, a rectangle consisting of four computed lengths; otherwise, as specified

The element's ancestors may also have clipping regions (e.g. if their 'overflow' property is not 'visible'); what is rendered is the intersection of the various clipping regions.

The following two rules:

```
p { clip: rect(5px, 40px, 45px, 5px); }
p { clip: rect(5px, 55px, 45px, 5px); }
```

Visibility

The 'visibility' property specifies whether the boxes generated by an element are rendered.
<visibilty> - value type:

- visible - the generated box is visible.
- hidden - the generated box is invisible (fully transparent), but still affects layout.
- collapse - please consult the section on dynamic row and column effects in tables.

'visibility'

Value:	visible | hidden | collapse | inherit
Initial:	visble
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

This property may be used in conjunction with scripts to create dynamic effects.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
<HEAD>
<STYLE type="text/css">
<!--
   #container1 { position: absolute;
                 top: 2in; left: 2in; width: 2in }
   #container2 { position: absolute;
                 top: 2in; left: 2in; width: 2in;
                 visibility: hidden; }
-->
</STYLE>
</HEAD>
<BODY>
<P>Choose a suspect:</P>
<DIV id="container1">
   <IMG alt="Al Capone"
        width="100" height="100"
        src="suspect1.jpg">
   <P>Name: Al Capone</P>
   <P>Residence: Chicago</P>
</DIV>

<DIV id="container2">
   <IMG alt="Lucky Luciano"
        width="100" height="100"
        src="suspect2.jpg">
   <P>Name: Lucky Luciano</P>
   <P>Residence: New York</P>
</DIV>

<FORM method="post"
      action="http://www.suspect.org/process-bums">
   <P>
   <INPUT name="Capone" type="button"
          value="Capone"
          onclick='show("container1");hide("container2")'>
   <INPUT name="Luciano" type="button"
          value="Luciano"
          onclick='show("container2");hide("container1")'>
</FORM>
</BODY>
</HTML>
```

Generated content, automatic numbering, and lists

In some cases, authors may want user agents to render content that does not come from the document tree.

The :before and :after pseudo-elements

The :*before* and :*after* pseudo-elements specify the location of content before and after an element's document tree content. The 'content' property, in conjunction with these pseudo-elements, specifies what is inserted.

Example:

```
p.note:before { content: "Note: " }
p.note        { border: solid green }
```

The :before and :after pseudo-elements inherit any inheritable properties from the element in the document tree to which they are attached.

```
Example:
q:before {
  content: open-quote;
  color: red
}
```

In a :before or :after pseudo-element declaration, non-inherited properties take their initial values.
Example:

```
body:after {
    content: "The End";
    display: block;
    margin-top: 2em;
    text-align: center;
}
```

The 'content' property

This property is used with the :before and :after pseudo-elements to generate content in a document.
<content> value type:

- normal - on the :before and :after pseudo-elements, this value is the same as 'none'.
- none – no content is generated.
- <string> - text content (see the section on strings).
- <uri> - the value is a URI that designates an external resource.
- <counter> - counters may be specified with two different functions: 'counter()' or 'counters()'.
- open-quote and close-quote - these values are replaced by the appropriate string from the 'quotes' property.
- no-open-quote and no-close-quote - same as 'none', but increments (decrements) the level of nesting for quotes.
- attr(X) - this function returns as a string the value of attribute X for the subject of the selector.

The 'display' property controls whether the content is placed in a block, inline, or marker box.

'content'

Value:	[<string> | <counter> | attr(<identifier>) | open-quote | close-quote | no-open-quote | no-close-quote]+ | inherit
Initial:	empty string

Applies to:	:before and :after pseudo-elements
Inherited:	no
Percentages:	N/A
Media:	all
Computed value:	for URI values, the absolute URI; for attr() values, the resulting string; otherwise as specified

The following rule causes the string "Chapter: " to be generated before each H1 element:

```
H1:before {
  content: "Chapter: ";
  display: inline;
}
```

Quotation marks

In CSS 2.1, authors may specify, in a style-sensitive and context-dependent manner, how user agents should render quotation marks. The 'quotes' property specifies pairs of quotation marks for each level of embedded quotation. The 'content' property gives access to those quotation marks and causes them to be inserted before and after a quotation.

<quotes>
This property specifies quotation marks for any number of embedded quotations. Values have the following meanings:

- none - the 'open-quote' and 'close-quote' values of the 'content' property produce no quotation marks.

[<string> <string>]+ - Values for the 'open-quote' and 'close-quote' values of the 'content' property are taken from this list of pairs of quotation marks (opening and closing).

'quotes'

Value:	[<string> <string>]+ | none | inherit
Initial:	depends on user agent
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

For example, applying the following style sheet:

```
/* Specify pairs of quotes for two levels in two languages */
q:lang(en) { quotes: '"' '"' "'" "'" }
q:lang(no) { quotes: "«" "»" '"' '"' }

/* Insert quotes before and after Q element content */
q:before { content: open-quote }
q:after  { content: close-quote }
to the following HTML fragment:
<HTML lang="en">
  <HEAD>
  <TITLE>Quotes</TITLE>
  </HEAD>
  <BODY>
    <P><Q>Quote me!</Q>
  </BODY>
</HTML>
```

Quotation marks are inserted in appropriate places in a document with the 'open-quote' and 'close-quote' values of the 'content' property. and inserts a single closing quote at the end:

```
blockquote p:before     { content: open-quote }
blockquote p:after      { content: no-close-quote }
blockquote p.last:after { content: close-quote }
```

Automatic counters

Automatic numbering in CSS2 is controlled with two properties, 'counter-increment' and 'counter-reset'. The counters defined by these properties are used with the counter() and counters() functions of the the 'content' property.

'counter-reset'

Value:	[<identifier> <integer>?]+ | none | inherit
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Media:	all
Computed value:	as specified

'counter-increment'

Value:	[<identifier> <integer>?]+ | none | inherit
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Media:	all
Computed value:	as specified

The 'counter-increment' property accepts one or more names of counters (identifiers), each one optionally followed by an integer. The integer indicates by how much the counter is incremented for every occurrence of the element. The default increment is 1. Zero and negative integers are allowed.

The 'counter-reset' property also contains a list of one or more names of counters, each one optionally followed by an integer. The integer gives the value that the counter is set to on each occurrence of the element. The default is 0.

```
H1:before {
    content: "Chapter " counter(chapter) ". ";
    counter-increment: chapter;  /* Add 1 to chapter */
    counter-reset: section;      /* Set section to 0 */
}
H2:before {
    content: counter(chapter) "." counter(section) " ";
    counter-increment: section;
}
```

Nested counters and scope

Counters are "self-nesting", in the sense that re-using a counter in a child element automatically creates a new instance of the counter.

Thus, the following suffices to number nested list items. The result is very similar to that of setting 'display:list-item' and 'list-style: inside' on the LI element:

```
OL { counter-reset: item }
LI { display: block }
LI:before { content: counter(item) ". "; counter-increment: item
}
```

The self-nesting is based on the principle that every element that has a 'counter-reset' for a counter X, creates a fresh counter X, the scope of which is the element, its preceding siblings, and all the descendants of the element and its preceding siblings.

```
<OL>               <!-- (set item[0] to 0          -->
  <LI>item         <!--  increment item[0] (= 1)   -->
  <LI>item         <!--  increment item[0] (= 2)   -->
    <OL>           <!--  (set item[1] to 0         -->
      <LI>item     <!--   increment item[1] (= 1)  -->
      <LI>item     <!--   increment item[1] (= 2)  -->
      <LI>item     <!--   increment item[1] (= 3)  -->
        <OL>       <!--   (set item[2] to 0        -->
          <LI>item <!--    increment item[2] (= 1) -->
        </OL>      <!--   )                        -->
        <OL>       <!--   (set item[3] to 0        -->
          <LI>     <!--    increment item[3] (= 1) -->
        </OL>      <!--   )                        -->
      <LI>item     <!--   increment item[1] (= 4)  -->
    </OL>          <!--  )                         -->
  <LI>item         <!--  increment item[0] (= 3)   -->
  <LI>item         <!--  increment item[0] (= 4)   -->
</OL>              <!-- )                         -->
<OL>               <!-- (reset item[4] to 0       -->
  <LI>item         <!--  increment item[4] (= 1)   -->
  <LI>item         <!--  increment item[4] (= 2)   -->
</OL>              <!-- )                         -->
```

The 'counters()' function generates a string composed of the values of all counters with the same name, separated by a given string.

The following style sheet numbers nested list items as "1", "1.1", "1.1.1", etc.

```
OL { counter-reset: item }
LI { display: block }
LI:before { content: counters(item, "."); counter-increment: item
}
```

Counter styles

By default, counters are formatted with decimal numbers, but all the styles available for the 'list-style-type' property are also available for counters. The notation is:

```
counter(name)
```

for the default style, or:

```
counter(name, 'list-style-type')
```

All the styles are allowed, including 'disc', 'circle', 'square', and 'none'.

```
H1:before        { content: counter(chno, upper-latin) ". " }
H2:before        { content: counter(section, upper-roman) " - " }
BLOCKQUOTE:after { content: " [" counter(bq, hebrew) "]" }
DIV.note:before  { content: counter(notecntr, disc) " " }
P:before         { content: counter(p, none) }
```

Counters in elements with display: none

An element that is not displayed ('display' set to 'none') cannot increment or reset a counter.

```
H2.secret {counter-increment: count2; display: none}
```

Lists

An element with 'display: list-item' generates a principal box for the element's content and an optional marker box as a visual indication that the element is a list item.

The list properties describe basic visual formatting of lists:

- They allow style sheets to specify the marker type (image, glyph, or number), and the marker position with respect to the principal box (outside it or within it before content).
- They do not allow authors to specify distinct style (colors, fonts, alignment, etc.) for the list marker or adjust its position with respect to the principal box.

The background properties apply to the principal box only; an 'outside' marker box is transparent.

'list-style-type'

Value:	disc | circle | square | decimal | decimal-leading-zero | lower-roman | upper-roman | lower-latin | upper-latin | none | inherit
Initial:	disc
Applies to:	elements with 'display: list-item'
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

This property specifies appearance of the list item marker if 'list-style-image' has the value 'none' or if the image pointed to by the URI cannot be displayed. The value 'none' specifies no marker, otherwise there are three types of marker: glyphs, numbering systems, and alphabetic systems.

Numbering systems are specified with:

- Decimal: Decimal numbers, beginning with 1.
- decimal-leading-zero: Decimal numbers padded by initial zeros (e.g., 01, 02, 03, ..., 98, 99).
- lower-roman: Lowercase roman numerals (i, ii, iii, iv, v, etc.).
- upper-roman: Uppercase roman numerals (I, II, III, IV, V, etc.).

Alphabetic systems are specified with:

- lower-latin or lower-alpha: Lowercase ascii letters (a, b, c, ... z).
- upper-latin or upper-alpha: Uppercase ascii letters (A, B, C, ... Z).

Example:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
   <HEAD>
     <TITLE>Lowercase latin numbering</TITLE>
     <STYLE type="text/css">
          ol { list-style-type: lower-roman }
     </STYLE>
  </HEAD>
  <BODY>
    <OL>
      <LI> This is the first item.
```

```
      <LI> This is the second item.
      <LI> This is the third item.
    </OL>
  </BODY>
</HTML>
```

'list-style-image'

Value:	<uri> | none | inherit
Initial:	none
Applies to:	elements with 'display: list-item'
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	absolute URI or 'none'

This property sets the image that will be used as the list item marker. When the image is available, it will replace the marker set with the 'list-style-type' marker.

```
ul { list-style-image: url("http://png.com/ellipse.png") }
```

'list-style-position'

Value:	inside | outside | inherit
Initial:	outside
Applies to:	elements with 'display: list-item'
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

This property specifies the position of the marker box in the principal block box. Values have the following meanings:

- outside - The marker box is outside the principal block box.
- inside - The marker box is the first inline box in the principal block box.

Example:

```
<HTML>
  <HEAD>
    <TITLE>Comparison of inside/outside position</TITLE>
    <STYLE type="text/css">
      ul         { list-style: outside }
      ul.compact { list-style: inside }
    </STYLE>
  </HEAD>
  <BODY>
    <UL>
      <LI>first list item comes first
      <LI>second list item comes second
    </UL>

    <UL class="compact">
      <LI>first list item comes first
      <LI>second list item comes second
    </UL>
  </BODY>
</HTML>
```

'list-style'

Value:	['list-style-type' || 'list-style-position' || 'list-style-image'] | inherit
Initial:	see individual properties
Applies to:	elements with 'display: list-item'
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	see individual properties

The 'list-style' property is a shorthand notation for setting the three properties 'list-style-type', 'list-style-image', and 'list-style-position' at the same place in the style sheet

```
ul { list-style: upper-roman inside }  /* Any "ul" element */
ul > li > ul { list-style: circle outside } /* Any "ul" child
                                               of an "li" child
                                               of a "ul" element */
```

The following rules look similar, but the first declares a descendant selector and the second a (more specific) child selector.

```
ol.alpha li   { list-style: lower-alpha } /* Any "li" descendant
of an "ol" */
ol.alpha > li { list-style: lower-alpha } /* Any "li" child of an
"ol" */
```

Authors who use only the descendant selector may not achieve the results they expect. Consider the following rules:

```
<HTML>
  <HEAD>
    <TITLE>WARNING: Unexpected results due to cascade</TITLE>
    <STYLE type="text/css">
      ol.alpha li  { list-style: lower-alpha }
      ul li        { list-style: disc }
    </STYLE>
  </HEAD>
  <BODY>
    <OL class="alpha">
      <LI>level 1
      <UL>
         <LI>level 2
      </UL>
    </OL>
  </BODY>
</HTML>
```

Paged media

Paged media (e.g., paper, transparencies, pages that are displayed on computer screens, etc.) differ from continuous media in that the content of the document is split into one or more discrete pages.

To handle page breaks, CSS2 extends the visual formatting model as follows:

- The page box extends the box model to allow authors to specify page margins.
- The page model extends the visual formatting model to account for page breaks.

The CSS 2.1 page model specifies how a document is formatted within the page box. The page box does not necessarily correspond to the real sheet where the document will ultimately be rendered (paper, transparency, screen, etc.). The user agent is responsible for transferring the page box to the sheet. Transfer possibilities include:

- Transferring one page box to one sheet (e.g., single-sided printing).
- Transferring two page boxes to both sides of the same sheet (e.g., double-sided printing).
- Transferring N (small) page boxes to one sheet (called "n-up").
- Transferring one (large) page box to N x M sheets (called "tiling").
- Creating signatures. A signature is a group of pages printed on a sheet, which, when folded and trimmed like a book, appear in their proper sequence.
- Printing one document to several output trays.
- Outputting to a file.

Colors and Backgrounds

CSS properties allow authors to specify the foreground color and background of an element. Backgrounds may be colors or images. Background properties allow authors to position a background image, repeat it, and declare whether it should be fixed with respect to the viewport or scrolled along with the document.

Foreground color

'color'

Value:	<color> | inherit
Initial:	depends on user agent
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

This property describes the foreground color of an element's text content. There are different ways to specify red:

```
em { color: red }              /* predefined color name */
em { color: rgb(255,0,0) }     /* RGB range 0-255   */
```

The background

Authors may specify the background of an element (i.e., its rendering surface) as either a color or an image. In terms of the box model, "background" refers to the background of the content, padding and border areas. Border colors and styles are set with the border properties. Margins are always transparent.

Background properties are not inherited, but the parent box's background will shine through by default because of the initial 'transparent' value on 'background-color'.

The background of the box generated by the root element covers the entire canvas.

Note: It is recommended that authors specify the background for the BODY element rather than the HTML element.

According to these rules, the canvas underlying the following HTML document will have a "marble" background:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <HEAD>
    <TITLE>Setting the canvas background</TITLE>
    <STYLE type="text/css">
       BODY { background: url("http://example.com/marble.png") }
    </STYLE>
  </HEAD>
  <BODY>
    <P>My background is marble.
  </BODY>
</HTML>
```

Properties

'background-color'

Value:	<color> | transparent | inherit
Initial:	transparent
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	as specified

This property sets the background color of an element, either a <color> value or the keyword 'transparent', to make the underlying colors shine through.

```
h1 { background-color: #F00 }
```

'background-image'

Value:	<uri> | none | inherit
Initial:	none
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	absolute URI

This property sets the background image of an element. When setting a background image, authors should also specify a background color that will be used when the image is unavailable. When the image is available, it is rendered on top of the background color. (Thus, the color is visible in the transparent parts of the image).
Values for this property are either <uri>, to specify the image, or 'none', when no image is used.

```
body { background-image: url("marble.png") }
p { background-image: none }
```

'background-repeat'

Value:	repeat | repeat-x | repeat-y | no-repeat | inherit
Initial:	repeat
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	as specified

If a background image is specified, this property specifies whether the image is repeated (tiled), and how. All tiling covers the content, padding and border areas of a box.

Values have the following meanings:

- Repeat - the image is repeated both horizontally and vertically.
- repeat-x - the image is repeated horizontally only.
- repeat-y -the image is repeated vertically only.
- no-repeat - the image is not repeated: only one copy of the image is drawn.

```
body {
  background: white url("pendant.png");
  background-repeat: repeat-y;
```

```
  background-position: center;
}
```

One copy of the background image is centered, and other copies are put above and below it to make a vertical band behind the element.

'background-attachment'

Value:	scroll | fixed | inherit
Initial:	scroll
Applies to:	all elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	as specified

If a background image is specified, this property specifies whether it is fixed with regard to the viewport ('fixed') or scrolls along with the containing block ('scroll').

Even if the image is fixed, it is still only visible when it is in the background, padding or border area of the element. Thus, unless the image is tiled ('background-repeat: repeat'), it may be invisible.

```
body {
  background: red url("pendant.png");
  background-repeat: repeat-y;
  background-attachment: fixed;
}

body {
  background: white url(paper.png) scroll; /* for all UAs */
  background: white url(ledger.png) fixed; /* for UAs that do
fixed backgrounds */
}
```

'background-position'

Value:	[[<percentage> | <length> | top | center | bottom] || [<percentage> | <length> | left | center | right]] | inherit
Initial:	0% 0%
Applies to:	block-level and replaced elements
Inherited:	no
Percentages:	refer to the size of the box itself
Media:	visual
Computed value:	for <length> the absolute value, otherwise a percentage

If a background image has been specified, this property specifies its initial position. Values have the following meanings:

- <percentage> <percentage>
  With a value pair of '0% 0%', the upper left corner of the image is aligned with the upper left corner of the box's padding edge. A value pair of '100% 100%' places the lower right corner of the image in the lower right corner of padding area.
- <length> <length>
  With a value pair of '2cm 1cm', the upper left corner of the image is placed 2cm to the right and 1cm below the upper left corner of the padding area.
- top left and left top

Same as '0% 0%'.

- top, top center, and center top
  Same as '50% 0%'.
- right top and top right
  Same as '100% 0%'.
- left, left center, and center left
  Same as '0% 50%'.
- center and center center
  Same as '50% 50%'.
- right, right center, and center right
  Same as '100% 50%'.
- bottom left and left bottom
  Same as '0% 100%'.
- bottom, bottom center, and center bottom
  Same as '50% 100%'.
- bottom right and right bottom
  Same as '100% 100%'.

If only one percentage or length value is given, it sets the horizontal position only, and the vertical position will be 50%. If two values are given, the horizontal position comes first. Combinations of length and percentage values are allowed, (e.g., '50% 2cm'). Negative positions are allowed. Keywords cannot be combined with percentage values or length values (all possible combinations are given above).

```
body { background: url("banner.jpeg") right top } /* 100%   0% */
body { background: url("banner.jpeg") top center }/*  50%   0% */
body { background: url("banner.jpeg") center }    /*  50%  50% */
body { background: url("banner.jpeg") bottom }    /*  50% 100% */

body {
  background-image: url("logo.png");
  background-attachment: fixed;
  background-position: 100% 100%;
  background-repeat: no-repeat;
}
```

'background'

Value:	['background-color' || 'background-image' || 'background-repeat' || 'background-attachment' || 'background-position'] | inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	no
Percentages:	allowed on 'background-position'
Media:	visual
Computed value:	see individual properties

The 'background' property is a shorthand property for setting the individual background properties (i.e., 'background-color', 'background-image', 'background-repeat', 'background-attachment' and 'background-position') at the same place in the style sheet.

Given a valid declaration, the 'background' property first sets all the individual background properties to their initial values, then assigns explicit values given in the declaration.

```
BODY { background: red }
P { background: url("chess.png") gray 50% repeat fixed }
```

Fonts

Matching Algorithm

Because there is no accepted, universal taxonomy of font properties, matching of properties to font faces must be done carefully. The properties are matched in a well-defined order to insure that the results of this matching process are as consistent as possible across UAs (assuming that the same library of font faces is presented to each of them).

1. The User Agent makes (or accesses) a database of relevant CSS 2.1 properties of all the fonts of which the UA is aware. If there are two fonts with exactly the same properties, the user agent selects one of them.
2. At a given element and for each character in that element, the UA assembles the font properties applicable to that element. Using the complete set of properties, the UA uses the 'font-family' property to choose a tentative font family. The remaining properties are tested against the family according to the matching criteria described with each property. If there are matches for all the remaining properties, then that is the matching font face for the given element.
3. If there is no matching font face within the 'font-family' being processed by step 2, and if there is a next alternative 'font-family' in the font set, then repeat step 2 with the next alternative 'font-family'.
4. If there is a matching font face, but it doesn't contain a glyph for the current character, and if there is a next alternative 'font-family' in the font sets, then repeat step 2 with the next alternative 'font-family'.
5. If there is no font within the family selected in 2, then use a UA-dependent default 'font-family' and repeat step 2, using the best match that can be obtained within the default font. If a particular character cannot be displayed using this font, then the UA has no suitable font for that character. The UA should map each character for which it has no suitable font to a visible symbol chosen by the UA, preferably a "missing character" glyph from one of the font faces available to the UA.

(The above algorithm can be optimized to avoid having to revisit the CSS 2.1 properties for each character.)

The per-property matching rules from (2) above are as follows:

a. 'font-style' is tried first. 'italic' will be satisfied if there is either a face in the UA's font database labeled with the CSS keyword 'italic' (preferred) or 'oblique'. Otherwise the values must be matched exactly or font-style will fail.
b. 'font-variant' is tried next. 'normal' matches a font not labeled as 'small-caps'; 'small-caps' matches (1) a font labeled as 'small-caps', (2) a font in which the small caps are synthesized, or (3) a font where all lowercase letters are replaced by upper case letters. A small-caps font may be synthesized by electronically scaling uppercase letters from a normal font.
c. 'font-weight' is matched next, it will never fail. (See 'font-weight' below.)
d. 'font-size' must be matched within a UA-dependent margin of tolerance. (Typically, sizes for scalable fonts are rounded to the nearest whole pixel, while the tolerance for bitmapped fonts could be as large as 20%.) Further computations, e.g. by 'em' values in other properties, are based on the computed value of 'font-size'.

Properties

'font-family'

Value: [[<family-name> | <generic-family>] [, <family-name>| <generic-family>]*] | inherit

Initial:	depends on user agent
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

The value is a prioritized list of font family names and/or generic family names. Unlike most other CSS properties, values are separated by a comma to indicate that they are alternatives:

```
body { font-family: Gill, Helvetica, sans-serif }
```

Although many fonts provide the "missing character" glyph, typically an open box, as its name implies this should not be considered a match for characters that cannot be found in the font. (It should, however, be considered a match for U+FFFD, the "missing character" character's code point).

There are two types of font family names:

- <**family-name**> - the name of a font family of choice. In the last example, "Gill" and "Helvetica" are font families.
- <**generic-family**> - in the example above, the last value is a generic family name. The following generic families are defined:
  - **'serif'** (e.g. Times)
  - **'sans-serif'** (e.g. Helvetica)
  - **'cursive'** (e.g. Zapf-Chancery)
  - **'fantasy'** (e.g. Western)
  - **'monospace'** (e.g. Courier)

If an unquoted font family name contains parentheses, brackets, and/or braces, they must still be either balanced or escaped per CSS grammar rules. Similarly, quote marks, semicolons, exclamation marks and commas within unquoted font family names must be escaped. Font names containing any such characters or whitespace should be quoted:

```
body { font-family: "New Century Schoolbook", serif }
```

in HTML

```
<BODY STYLE="font-family: 'My own font', fantasy">
```

'font-style'

Value:	normal | italic | oblique | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

The 'font-style' property selects between normal (sometimes referred to as "roman" or "upright"), italic and oblique faces within a font family.

Fonts with Oblique, Slanted or Incline in their names will typically be labeled 'oblique' in the UA's font database. Fonts with Italic, Cursive or Kursiv in their names will typically be labeled 'italic'.

```
h1, h2, h3 { font-style: italic }
h1 em { font-style: normal }
```

'font-variant'

Value:	normal | small-caps | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

Another type of variation within a font family is the small-caps. In a small-caps font the lower case letters look similar to the uppercase ones, but in a smaller size and with slightly different proportions. The 'font-variant' property selects that font.

```
h3 { font-variant: small-caps }
em { font-style: oblique }
```

'font-weight'

Value:	normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900 | inherit
Initial:	normal
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	see text

The 'font-weight' property selects the weight of the font. The values '100' to '900' form an ordered sequence, where each number indicates a weight that is at least as dark as its predecessor. The keyword 'normal' is synonymous with '400', and 'bold' is synonymous with '700'. Keywords other than 'normal' and 'bold' have been shown to be often confused with font names and a numerical scale was therefore chosen for the 9-value list.

```
p { font-weight: normal }   /* 400 */
h1 { font-weight: 700 }     /* bold */
```

The 'bolder' and 'lighter' values select font weights that are relative to the weight inherited from the parent:

```
strong { font-weight: bolder }
```

Child elements inherit the resultant weight, not the keyword value.

'font-size'

Value:	<absolute-size> | <relative-size> | <length> | <percentage> | inherit
Initial:	medium
Applies to:	all elements
Inherited:	yes
Percentages:	refer to parent element's font size
Media:	visual
Computed value:	absolute length

The font size corresponds to the em square, a concept used in typography. Note that certain glyphs may bleed outside their em squares. Values have the following meanings:

- <absolute-size> font sizes computed and kept by the UA. Possible values are: [xx-small | x-small | small | medium | large | x-large | xx-large]. Different media may need different scaling factors.

- <relative-size> - relative to the table of font sizes and the font size of the parent element. Possible values are: [larger | smaller].

Examples:

```
p { font-size: 16px; }
@media print {
      p { font-size: 12pt; }
}
blockquote { font-size: larger }
em { font-size: 150% }
em { font-size: 1.5em }
```

'font'

Value:	[['font-style' || 'font-variant' || 'font-weight']? 'font-size' [/ 'line-height']? 'font-family'] | caption | icon | menu | message-box | small-caption | status-bar | inherit
Initial:	see individual properties
Applies to:	all elements
Inherited:	yes
Percentages:	allowed on 'font-size' and 'line-height'
Media:	visual
Computed value:	see individual properties

The 'font' property is, except as described below, a shorthand property for setting 'font-style', 'font-variant', 'font-weight', 'font-size', 'line-height' and 'font-family' at the same place in the style sheet. The syntax of this property is based on a traditional typographical shorthand notation to set multiple properties related to fonts.

All font-related properties are first reset to their initial values, including those listed in the preceding paragraph. Then, those properties that are given explicit values in the 'font' shorthand are set to those values. For a definition of allowed and initial values, see the previously defined properties.

```
p { font: 12px/14px sans-serif }
p { font: 80% sans-serif }
p { font: x-large/110% "New Century Schoolbook", serif }
p { font: bold italic large Palatino, serif }
p { font: normal small-caps 120%/120% fantasy }
```

The following values refer to system fonts:

- caption - the font used for captioned controls (e.g., buttons, drop-downs, etc.).
- icon - the font used to label icons.
- menu – the font used in menus (e.g., dropdown menus and menu lists).
- message-box - the font used in dialog boxes.
- small-caption - the font used for labeling small controls.
- status-bar - the font used in window status bars.

System fonts may only be set as a whole; that is, the font family, size, weight, style, etc. are all set at the same time.

That is why this property is "almost" a shorthand property: system fonts can only be specified with this property, not with 'font-family' itself, so 'font' allows authors to do more than the sum of its subproperties. However, the individual properties such as 'font-weight' are still given values taken from the system font, which can be independently varied.

```
button { font: 300 italic 1.3em/1.7em "FB Armada", sans-serif }
button p { font: menu }
```

```
button p em { font-weight: bolder }

button p { font: 600 9px Charcoal }

button p {
  font-family: Charcoal;
  font-style: normal;
  font-variant: normal;
  font-weight: 600;
  font-size: 9px;
  line-height: normal;
}
```

Text

Indentation

'text-indent'

Value:	<length> | <percentage> | inherit
Initial:	0
Applies to:	block-level elements
Inherited:	yes
Percentages:	refer to width of containing block
Media:	visual
Computed value:	absolute length

This property specifies the indentation of the first line of text in a block. More precisely, it specifies the indentation of the first box that flows into the block's first line box. The box is indented with respect to the left (or right, for right-to-left layout) edge of the line box. User agents should render this indentation as blank space.
Values have the following meanings:

- <**length**> - The indentation is a fixed length.
- <**percentage**> - The indentation is a percentage of the containing block width.

The value of 'text-indent' may be negative, but there may be implementation-specific limits. If the value of 'text-indent' is negative, the value of 'overflow' will affect whether the text is visible.

```
p { text-indent: 3em }
```

Alignment

'text-align'

Value:	left | right | center | justify | inherit
Initial:	depends on user agent and writing direction
Applies to:	block-level elements and table cells
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

This property describes how inline content of a block is aligned. Values have the following meanings:

- left, right, center, justify - left, right, center, and justify text, respectively.

A block of text is a stack of line boxes. In the case of 'left', 'right' and 'center', this property specifies how the inline boxes within each line box align with respect to the line box's left and right sides; alignment is not with respect to the viewport.

```
div.important { text-align: center }
```

Decoration

'text-decoration'

Value:	none | [underline || overline || line-through || blink] | inherit
Initial:	none
Applies to:	all elements
Inherited:	no (see prose)

Percentages: N/A
Media: visual
Computed value: as specified

This property describes decorations that are added to the text of an element.

Values have the following meanings:

- none - Produces no text decoration.
- underline - Each line of text is underlined.
- overline - Each line of text has a line above it.
- line-through - Each line of text has a line through the middle.
- blink - Text blinks (alternates between visible and invisible).

The color(s) required for the text decoration must be derived from the 'color' property value. This property is not inherited, but descendant boxes of a block box should be formatted with the same decoration (e.g., they should all be underlined). The color of decorations should remain the same even if descendant elements have different 'color' values.

```
a:visited,a:link { text-decoration: underline }
```

Letter and word spacing

'letter-spacing'

Value: normal | <length> | inherit
Initial: normal
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual
Computed value: 'normal' or absolute length

This property specifies spacing behavior between text characters. Values have the following meanings:

- normal - The spacing is the normal spacing for the current font. This value allows the user agent to alter the space between characters in order to justify text.
- <length> - This value indicates inter-character space in addition to the default space between characters. Values may be negative, but there may be implementation-specific limits.

Character spacing algorithms are user agent-dependent. Character spacing may also be influenced by justification (see the 'text-align' property).

```
blockquote { letter-spacing: 0.1em }
In the following example, the user agent is not permitted to
alter inter-character space:
blockquote { letter-spacing: 0cm }   /* Same as '0' */
```

'word-spacing'

Value: normal | <length> | inherit
Initial: normal
Applies to: all elements
Inherited: yes
Percentages: N/A
Media: visual
Computed value: for 'normal' the value '0'; otherwise the absolute length

This property specifies spacing behavior between words. Values have the following meanings:

- normal - The normal inter-word space, as defined by the current font and/or the UA.
- <length> - This value indicates inter-word space in addition to the default space between words.

```
h1 { word-spacing: 1em }
```

Capitalization

'text-transform'

Value:	capitalize | uppercase | lowercase | none | inherit
Initial:	none
Applies to:	all elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

This property controls capitalization effects of an element's text. Values have the following meanings:

- capitalize - Puts the first character of each word in uppercase.
- uppercase - Puts all characters of each word in uppercase.
- lowercase - Puts all characters of each word in lowercase.
- none - No capitalization effects.

```
h1 { text-transform: uppercase }
```

Tables

The CSS table model

The CSS table model is based on the HTML 4.0 table model, in which the structure of a table closely parallels the visual layout of the table. In this model, a table consists of an optional caption and any number of rows of cells.

The following 'display' values assign table formatting rules to an arbitrary element:

- **table** (In HTML: TABLE) - Specifies that an element defines a block-level table: it is a rectangular block that participates in a block formatting context.
- **inline-table** (In HTML: TABLE) - Specifies that an element defines an inline-level table: it is a rectangular block that participates in an inline formatting context).
- **table-row** (In HTML: TR) - Specifies that an element is a row of cells.
- **table-row-group** (In HTML: TBODY) - Specifies that an element groups one or more rows.
- **table-header-group** (In HTML: THEAD) - Like 'table-row-group', but for visual formatting, the row group is always displayed before all other rows and rowgroups and after any top captions.
- **table-footer-group** (In HTML: TFOOT) - Like 'table-row-group', but for visual formatting, the row group is always displayed after all other rows and rowgroups and before any bottom captions.
- **table-column** (In HTML: COL) - Specifies that an element describes a column of cells.
- **table-column-group** (In HTML: COLGROUP) - Specifies that an element groups one or more columns.
- **table-cell** (In HTML: TD, TH) - Specifies that an element represents a table cell.
- **table-caption** (In HTML: CAPTION) - Specifies a caption for the table.

The default style sheet for HTML 4.0 in the appendix illustrates the use of these values for HTML 4.0:

```
table    { display: table }
tr       { display: table-row }
thead    { display: table-header-group }
tbody    { display: table-row-group }
tfoot    { display: table-footer-group }
col      { display: table-column }
colgroup { display: table-column-group }
td, th   { display: table-cell }
caption  { display: table-caption }
```

Column selectors

Table cells may belong to two contexts: rows and columns. However, in the source document cells are descendants of rows, never of columns. Nevertheless, some aspects of cells can be influenced by setting properties on columns.

The following properties apply to column and column-group elements:

- **'border'** - The various border properties apply to columns only if 'border-collapse' is set to 'collapse' on the table element.
- **'background'** - The background properties set the background for cells in the column, but only if both the cell and row have transparent backgrounds.
- **'width'** - The 'width' property gives the minimum width for the column.
- **'visibility'** - If the 'visibility' of a column is set to 'collapse', none of the cells in the column are rendered, and cells that span into other columns are clipped.

```
col   { border-style: none solid }
table { border-style: hidden }
col.totals { background: blue }
table { table-layout: fixed }
col.totals { width: 5em }
```

Tables in the visual formatting model

In terms of the visual formatting model, a table may behave like a block-level or replaced inline-level element. Tables have content, padding, borders, and margins.

'caption-side'

Value:	top | bottom | left | right | inherit
Initial:	top
Applies to:	'table-caption' elements
Inherited:	yes
Percentages:	N/A
Media:	visual
Computed value:	as specified

This property specifies the position of the caption box with respect to the table box. Values have the following meanings:

- **top** - Positions the caption box above the table box.
- **bottom** - Positions the caption box below the table box.

Captions above or below a 'table' element are formatted very much as if they were a block element before or after the table, except that (1) they inherit inheritable properties from the table, and (2) they are not considered to be a block box for the purposes of any 'compact' or 'run-in' element that may precede the table.

```
caption { caption-side: bottom;
          width: auto;
          text-align: left }
```

Visual layout of table contents

Like other elements of the document language, internal table elements generate rectangular boxes with content and borders. Cells have padding as well. Internal table elements do not have margins.

The visual layout of these boxes is governed by a rectangular, irregular grid of rows and columns. Each box occupies a whole number of grid cells, determined according to the following rules. These rules do not apply to HTML 4.0.

```
<TABLE>
<TR><TD>1 <TD rowspan="2">2 <TD>3 <TD>4
<TR><TD colspan="2">5
</TABLE>

<TABLE>
<ROW><CELL>1 <CELL rowspan="2">2 <CELL>3 <CELL>4
<ROW><CELL colspan="2">5
</TABLE>
```

Table layers and transparency

For the purposes of finding the background of each table cell, the different table elements may be thought of as being on six superimposed layers. The background set on an element in one of the layers will only be visible if the layers above it have a transparent background.

The edges of the rows, columns, row groups and column groups in the collapsing borders model coincide with the hypothetical grid lines on which the borders of the cells are centered. In the separated borders model, the edges coincide with the border edges of cells.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<HTML>
  <HEAD>
    <STYLE type="text/css">
      TABLE { background: #ff0; border-collapse: collapse }
      TD    { background: red; border: double black }
    </STYLE>
  </HEAD>
  <BODY>
    <P>
    <TABLE>
      <TR>
        <TD> 1
        <TD rowspan="2"> 2
        <TD> 3
        <TD> 4
      </TR>
      <TR><TD></TD></TR>
    </TABLE>
  </BODY>
</HTML>
```

Table width

CSS does not define an "optimal" layout for a table since, in many cases, what is optimal is a matter of taste.

'table-layout'

Value:	auto | fixed | inherit
Initial:	auto
Applies to:	'table' and 'inline-table' elements
Inherited:	no
Percentages:	N/A
Media:	visual
Computed value:	as specified

The 'table-layout' property controls the algorithm used to lay out the table cells, rows, and columns. Values have the following meaning:

- fixed - Use the fixed table layout algorithm
- auto - Use any automatic table layout algorithm

The two algorithms are described below.

Fixed table layout

With this (fast) algorithm, the horizontal layout of the table does not depend on the contents of the cells; it only depends on the table's width, the width of the columns, and borders or cell spacing.

The table's width may be specified explicitly with the 'width' property. A value of 'auto' (for both 'display: table' and 'display: inline-table') means use the automatic table layout algorithm.

```
table { table-layout: fixed;
        margin-left: 2em;
        margin-right: 2em }
```

The width of the table is then the greater of the value of the 'width' property for the table element and the sum of the column widths (plus cell spacing or borders). If the table is wider than the columns, the extra space should be distributed over the columns.
In this manner, the user agent can begin to lay out the table once the entire first row has been received. Cells in subsequent rows do not affect column widths.

Automatic table layout
In this algorithm (which generally requires no more than two passes), the table's width is given by the width of its columns (and intervening borders).

Table height

The height of a table is given by the 'height' property for the 'table' or 'inline-table' element. A value of 'auto' means that the height is the sum of the row heights plus any cell spacing or borders. Any other value specifies the height explicitly; the table may thus be taller or shorter than the height of its rows.

The height of a 'table-row' element's box is calculated once the user agent has all the cells in the row available: it is the maximum of the row's specified 'height' and the minimum height (MIN) required by the cells. A 'height' value of 'auto' for a 'table-row' means the computed row height is MIN. MIN depends on cell box heights and cell box alignment (much like the calculation of a line box height).

The 'vertical-align' property of each table cell determines its alignment within the row. Each cell's content has a baseline, a top, a middle, and a bottom, as does the row itself. In the context of tables, values for 'vertical-align' have the following meanings:

- baseline - The baseline of the cell is put at the same height as the baseline of the first of the rows it spans (see below for the definition of baselines of cells and rows).
- top - The top of the cell box is aligned with the top of the first row it spans.
- bottom - The bottom of the cell box is aligned with the bottom of the last row it spans.
- middle - The center of the cell is aligned with the center of the rows it spans.
- sub, super, text-top, text-bottom - These values do not apply to cells; the cell is aligned at the baseline instead.

Borders

There are two distinct models for setting borders on table cells in CSS. One is most suitable for so-called separated borders around individual cells, the other is suitable for borders that are continuous from one end of the table to the other.

'border-collapse'

Value:	collapse | separate | inherit
Initial:	separate
Applies to:	'table' and 'inline-table' elements
Inherited:	yes
Percentages:	N/A
Media:	visual

Computed value: as specified

This property selects a table's border model. The value 'separate' selects the separated borders border model. The value 'collapse' selects the collapsing borders model.

'border-spacing' - separate borders model

Value: <length> <length>? | inherit
Initial: 0
Applies to: 'table' and 'inline-table' elements
Inherited: yes
Percentages: N/A
Media: visual
Computed value: two absolute lengths

The lengths specify the distance that separates adjacent cell borders. If one length is specified, it gives both the horizontal and vertical spacing. If two are specified, the first gives the horizontal spacing and the second the vertical spacing. Lengths may not be negative.

```
table      { border: outset 10pt;
             border-collapse: separate;
             border-spacing: 15pt }
td         { border: inset 5pt }
td.special { border: inset 10pt }  /* The top-left cell */
```

'empty-cells'

Value: show | hide | inherit
Initial: show
Applies to: 'table-cell' elements
Inherited: yes
Percentages: N/A
Media: visual
Computed value: as specified

In the separated borders model, this property controls the rendering of borders and backgrounds around cells that have no visible content. Empty cells and cells with the 'visibility' property set to 'hidden' are considered to have no visible content. Visible content includes " " and other whitespace except ASCII CR ("\0D"), LF ("\0A"), tab ("\09"), and space ("\20").

```
table { empty-cells: show }
```

Collapsing border model

In the collapsing border model, it is possible to specify borders that surround all or part of a cell, row, row group, column, and column group.

Borders are centered on the grid lines between the cells. User agents must find a consistent rule for rounding off in the case of an odd number of discrete units (screen pixels, printer dots).

*row-width = (0.5 * border-width0) + padding-left1 + width1 + padding-right1 + border-width1 + padding-left2 +...+ padding-rightn + (0.5 * border-widthn)*

The following example illustrates the application of these precedence rules. This style sheet:

```
table          { border-collapse: collapse;
                 border: 5px solid yellow; }
```

```
*#col1         { border: 3px solid black; }
td             { border: 1px solid red; padding: 1em; }
td.solid-blue  { border: 5px dashed blue; }
td.solid-green { border: 5px solid green; }
```

with this HTML source:

```
<P>
<TABLE>
<COL id="col1"><COL id="col2"><COL id="col3">
<TR id="row1">
    <TD> 1
    <TD> 2
    <TD> 3
</TR>
<TR id="row2">
    <TD> 4
    <TD class="solid-blue"> 5
    <TD class="solid-green"> 6
</TR>
<TR id="row3">
    <TD> 7
    <TD> 8
    <TD> 9
</TR>
<TR id="row4">
    <TD> 10
    <TD> 11
    <TD> 12
</TR>
<TR id="row5">
    <TD> 13
    <TD> 14
    <TD> 15
</TR>
</TABLE>
```

Borders styles

Some of the values of the 'border-style' have different meanings in tables than for other elements. In the list below they are marked with an asterisk.

- none - No border.
- *hidden - Same as 'none', but in the collapsing border model, also inhibits any other border (see the section on border conflicts).
- 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.
- groove - The border looks as though it were carved into the canvas.
- ridge - The opposite of 'grove': the border looks as though it were coming out of the canvas.
- *inset - In the separated borders model, the border makes the entire box look as though it were embedded in the canvas.
- *outset - In the separated borders model, the border makes the entire box look as though it were coming out of the canvas.

Bibliography

Appendix

The following CSS example can be used as a basis for development.

```
address,
blockquote,
body, dd, div,
dl, dt, fieldset, form,
frame, frameset,
h1, h2, h3, h4,
h5, h6, noframes,
ol, p, ul, center,
dir, hr, menu, pre   { display: block }
li               { display: list-item }
head             { display: none }
table            { display: table }
tr               { display: table-row }
thead            { display: table-header-group }
tbody            { display: table-row-group }
tfoot            { display: table-footer-group }
col              { display: table-column }
colgroup         { display: table-column-group }
td, th           { display: table-cell; }
caption          { display: table-caption }
th               { font-weight: bolder; text-align: center }
caption          { text-align: center }
body             { padding: 8px; line-height: 1.12em }
h1               { font-size: 2em; margin: .67em 0 }
h2               { font-size: 1.5em; margin: .75em 0 }
h3               { font-size: 1.17em; margin: .83em 0 }
h4, p,
blockquote, ul,
fieldset, form,
ol, dl, dir,
menu             { margin: 1.12em 0 }
h5               { font-size: .83em; margin: 1.5em 0 }
h6               { font-size: .75em; margin: 1.67em 0 }
h1, h2, h3, h4,
h5, h6, b,
strong           { font-weight: bolder }
blockquote       { margin-left: 40px; margin-right: 40px }
i, cite, em,
var, address     { font-style: italic }
pre, tt, code,
kbd, samp        { font-family: monospace }
pre              { white-space: pre }
button, textarea,
input, object,
select, img { display:inline-block; }
big              { font-size: 1.17em }
small, sub, sup { font-size: .83em }
sub              { vertical-align: sub }
sup              { vertical-align: super }
s, strike, del   { text-decoration: line-through }
hr               { border: 1px inset }
```

```
ol, ul, dir,
menu, dd        { margin-left: 40px }
ol              { list-style-type: decimal }
ol ul, ul ol,
ul ul, ol ol    { margin-top: 0; margin-bottom: 0 }
u, ins          { text-decoration: underline }
br:before       { content: "\A" }
center          { text-align: center }
abbr, acronym   { font-variant: small-caps; letter-spacing: 0.1em
}
:link, :visited { text-decoration: underline }
:focus          { outline: thin dotted invert }

/* Begin bidirectionality settings (do not change) */
BDO[DIR="ltr"]  { direction: ltr; unicode-bidi: bidi-override }
BDO[DIR="rtl"]  { direction: rtl; unicode-bidi: bidi-override }

*[DIR="ltr"]    { direction: ltr; unicode-bidi: embed }
*[DIR="rtl"]    { direction: rtl; unicode-bidi: embed }

@media print {
  h1            { page-break-before: always }
  h1, h2, h3,
  h4, h5, h6    { page-break-after: avoid }
  ul, ol, dl    { page-break-before: avoid }
}
```