Abstract

Reference guide for Cascading Style Sheets 2.0, a language for describing the style of Web pages.

This publication is available in Web form¹ and also as a PDF document². Please forward any comments to tcc-doc@nmt.edu.

Table of Contents

1. What are Cascading Style Sheets and why should you use them? ................................................ 3
2. Connecting your page to a style sheet ........................................................................................ 3
3. Overall structure of a style sheet ............................................................................................... 4
4. At-rules ................................................................................................................................... 4
5. How to write CSS rules ............................................................................................................ 5
6. Common value types in CSS ..................................................................................................... 5
   6.1. Dimensions ................................................................................................................... 5
   6.2. Specifying colors ........................................................................................................... 6
   6.3. String constants ............................................................................................................. 6
   6.4. Universal resource identifiers (URIs) .............................................................................. 7
   6.5. Counters ....................................................................................................................... 7
   6.6. Specifying angles ........................................................................................................... 7
   6.7. Times ........................................................................................................................... 7
   6.8. Frequencies ................................................................................................................... 8
7. Selectors .................................................................................................................................. 8
   7.1. Element type selectors ................................................................................................... 8
   7.2. Selecting elements by class ............................................................................................ 8
   7.3. Selecting elements by their context ............................................................................... 9
   7.4. Child selection ............................................................................................................. 9
   7.5. Adjacent element selection ............................................................................................ 9
   7.6. Selecting by attribute values ......................................................................................... 10
   7.7. Selecting specific single elements by ID ......................................................................... 10
   7.8. Pseudo-classes ............................................................................................................. 10
   7.9. Pseudo-elements .......................................................................................................... 11
   7.10. The universal selector ................................................................................................. 12
8. When rules collide .................................................................................................................. 12
   8.1. Cascading ................................................................................................................... 12
   8.2. Specificity: Which selector applies? ............................................................................... 13
   8.3. Inheritance .................................................................................................................. 13
9. Declarations ........................................................................................................................... 13

¹ http://www.nmt.edu/tcc/help/pubs/css/
1. What are Cascading Style Sheets and why should you use them?

CSS (Cascading Style Sheets) defines a style sheet language that you write to describe the presentation you want. This language can be included in your page or stored in a separate file.

Assuming that you want all your pages to share the same basic style, the best approach is to use a text editor to write your CSS rules in a separate text file and then use a special `<link>` tag on each of your pages to point to that file.

2. Connecting your page to a style sheet

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>My Page</title>
    <link rel="stylesheet" type="text/css" href="styles.css">
</head>
<body>
    <!-- Your content here -->
</body>
</html>
```

Useful online resources:

- The CSS standard is defined by the W3 Consortium, the umbrella organization for Web-related standards. See the W3 CSS homepage for a variety of resources: tutorials, standards, and books.
- This document is based on Cascading Style Sheets, level 2 revision 1: CSS 2.1 Specification. Some of the rules governing page makeup are extremely tricky; refer to this document for all the fine points.

3. Setting the dimensions and font size

```html
<style>
    body { font-family: Arial, sans-serif; font-size: 16px; line-height: 1.5; }
</style>
```

New Mexico Tech Computer Center  Styling Web pages with CSS-2  3
For example, you could write your CSS rules into a file called mystyle.css and then place this tag somewhere inside the head element of each of your HTML pages:

```html
<link rel='stylesheet' href='mystyle.css' type='text/css'>
```

The href='...' attribute will accept any URI, just like the href attribute of the HTML <a> tag.

Here's a small, complete Web page with a stylesheet link:

```html
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
  <title>Small example page</title>
  <link rel="stylesheet" href="mystyle.css" type="text/css">
</head>
<body>
  <h1>Title</h1>
  <p>Sample paragraph.</p>
</body>
</html>
```

### 3. Overall structure of a style sheet

To write a style sheet, use a text editor to create a file whose name ends in .css.

A style sheet is a sequence of statements. Each statement can be either:

- A rule that describes the styling of page elements. See Section 5, “How to write CSS rules” (p. 5).
- An at-rule, starting with “@”. See Section 4, “At-rules” (p. 4).

#### Note

Because HTML element names are case-insensitive, CSS stylesheets used to mark up HTML pages are also case-insensitive, so that for example element names "DIV" and "div" are considered equivalent. However, because XML element names are case-sensitive, names in a CSS stylesheet associated with an XML file are also case-sensitive. So, for example, XML element names "castle" and "CASTLE" are not the same.

### 4. At-rules

Besides regular style rules, your CSS can include “at-rules.” These directives all start with an at-sign (@).

- Section 21, “The @import rule: Importing another stylesheet” (p. 43).
- Section 22, “The @media rule: Tuning for different rendering platforms” (p. 44).
- Section 23, “The @page rule: Paged media” (p. 45).
5. How to write CSS rules

Most of the statements in a CSS stylesheet are rules that specify how certain elements of your page should appear.

Here is an example of a typical rule:

```css
h1 { color: blue; text-align: center; }
```

This rule says that all h1 headings should be rendered using blue letters and centered on the page.

In general, a CSS rule has this structure:

```css
selector { declaration; declaration; ... }
```

where the selector describes which kinds of elements of your Web page are affected, and each declaration describes how those elements should appear. If there are multiple declarations, separate them with semicolons (";").

The declaration part has this format:

```css
property: value
```

where the property is a keyword that specifies what aspect of the element you are changing, and the value says what you are changing it to. In the example rule above, the selector is h1; color and text-align are properties; and blue and center are values.

You can add comments by enclosing them between "/*" and "*/" characters. For example:

```css
/* Render level 3 and 4 headings in red, centered, and using sans-serif italic set 13/15 */
h3, h4 { color: red; text-align: center; font-family: sans-serif; font-style: italic; font-size: 13pt; line-heading: 15pt }
```

See Section 7, “Selectors” (p. 8) and Section 9, “Declarations” (p. 13).

6. Common value types in CSS

Before reviewing the many properties in CSS, we need to survey some of the common types of values used by properties.

6.1. Dimensions

Wherever a property can be set to a value representing a physical dimension, such as a height or a width, there are several ways to specify them.

These units are used in property definitions:

<table>
<thead>
<tr>
<th>Unit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>mm</td>
<td>Millimeters</td>
</tr>
<tr>
<td>Unit</td>
<td>Description</td>
</tr>
<tr>
<td>------</td>
<td>-------------</td>
</tr>
<tr>
<td>cm</td>
<td>Centimeters</td>
</tr>
<tr>
<td>in</td>
<td>Inches</td>
</tr>
<tr>
<td>pt</td>
<td>Printer’s points, about 1/72”</td>
</tr>
<tr>
<td>pc</td>
<td>Printer’s pica, about 1/6”</td>
</tr>
<tr>
<td>em</td>
<td>An em is the current font size. For example, if the font size is 14pt, two ems is 28pt.</td>
</tr>
<tr>
<td>px</td>
<td>One pixel on the display.</td>
</tr>
</tbody>
</table>

You can also specify dimensions as a percentage. For example, specifying the left margin as 10% would mean the left margin would be a tenth of the total screen width.

**Note**

Whenever possible, prefer relative units (ems and percentages) over fixed units. Consider the plight of readers whose eyes are aging and not as sharp as they were as a youngster. They can read your page if they increase the font size. Percentage units can help your layout adjust to different screen sizes.

### 6.2. Specifying colors

There are several ways of describing colors in Cascading Style sheets.

- There are sixteen predefined color names: aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, yellow, and white.
- You can specify colors as percentages in the RGB (red-green-blue) color system. For example, this rule would display all h1 headings in cyan:

  ```css
  h1 { color: rgb(0%, 100%, 100%); }
  ```

  where the first number is red, the second green, and the third blue.
- The rgb() notation also allows numbers in the range 0–255. For example, rgb(255, 255, 0) is pure yellow (pure red plus pure green).
- You can also define the colors using a string of the form #rgb or #rrggbb, using hexadecimal digits to specify the relative amount of red, green and blue. For example, color #ff5500 has full red, a little green, and no blue.

  Any color in the #rgb form is the same as if each of the three digits were doubled. For example, #ff50 is the same as #ff5500.

### 6.3. String constants

Most property values in CSS are keywords. However, sometimes it is necessary to specify a string constant. You can use either double-quotes "..." or single-quotes ‘...’ to enclose string constants.

For example, to import a stylesheet named "basic.css" into your stylesheet, you would use this at-rule:

```css
@import "basic.css";
```
To include a quote symbol inside a string, precede it with a backslash, e.g., 'don\'t' is the same string as "don't".

You can include any character value inside a string by encoding it as \"hh\", where hh is the character's code value in hexadecimal. For example, since the single-quote character has code hex 27, string "don't" could also be represented as 'don\27t'.

6.4. Universal resource identifiers (URIs)

A URI describes some resource on the World Wide Web. The best way to identify such a resource is to use the url function, which has this format:

```javascript
url("URI")
```

where URI is the resource's URI.

For example, suppose you have a background image named lawn.jpg in the same directory as all your pages. This rule would place it as the page background:

```css
body { background-image: url("lawn.jpg"); }
```

You can, of course, use a full URI to refer to any image on the Web.

6.5. Counters

If you need to assign serial numbers to some sequence of elements, such as bullets in a bullet list, sections in a chapter, or similar applications, CSS allows you to declare counters that hold such numbers.

Counters are created by using the counter-reset property. For details on the creation and application of counters, see Section 16, “The content property: Specifying content in pseudo-elements” (p. 29).

6.6. Specifying angles

To describe an angle, use a signed number immediately followed by one of the three angle specifiers:

- deg for degrees.
- grad for grads.
- rad for radians.

For example, audio playback defines an azimuth property that specifies the apparent source of a sound in the stereo image. Azimuth zero is straight ahead; 90° is purely in the right ear; -90° is in the left ear; and so on. So, to specify that the azimuth of a spoken h1 heading is halfway between straight ahead and the left ear, you would use this rule:

```css
h1 { azimuth: -45deg; }
```

6.7. Times

CSS time intervals are specified as a number followed immediately by either ms for milliseconds or s for seconds.

For example, if you want the speech synthesizer to insert a fifty-millisecond delay before speaking an h3 heading, use this rule:
6.8. Frequencies

Frequencies in CSS are specified as a number followed immediately by suffix Hz for Hertz, or kHz for kiloHertz. (Note that CSS is case-insensitive, so you don't have to capitalize them as carefully as they are shown here.)

For example, to specify that a speech synthesizer should produce a voice with a frequency around 180 Hertz for title elements, you would use this rule:

```css
title { pitch: 180Hz; }
```

7. Selectors

The selector part of a CSS rule can take several forms.

7.1. Element type selectors

To make a rule apply to all occurrences of a given HTML element, use the element (tag) name as a selector. For example, to indent every HTML `<p>…</p>` (paragraph) element half an inch:

```css
p { text-indent: 0.5in; }
```

You can also make the rule apply to multiple element types by giving a list of tag names separated by commas, for example:

```css
h1, h2, h3 { color: red; }
```

This rule would render all three heading levels as red: h1, h2, and h3.

7.2. Selecting elements by class

You can make a rule apply only to certain occurrences of an element. To do this:

1. Invent a class name (using letters, digits, and hyphens).
2. Add an attribute `class='C'` to the elements you want to affect, where C is the class name you have invented. Any HTML start tag can contain a `class='C'` attribute.
3. Then, in your style sheet, use a selector of this form:

   ```css
element.C
   ```

For example, suppose you want to designate certain paragraphs as “key paragraphs.” You might tag all those paragraphs using:

```html
<p class='key'> ... </p>
```

If you want key paragraphs set in purple type, add this rule:

```css
p.key { color: purple; }
```
You can use the HTML `div` and `span` elements to attach class names to sections of your documents. These tags don’t affect the HTML; they exist only as places to attach class names. Use `div` for block-level elements, meaning that this element always starts a new line on the page and ends with a new line. Use the `span` element to mark part of a line or paragraph.

### 7.3. Selecting elements by their context

Sometimes you want to apply a style rule to a particular element only in certain contexts. To do this, use a selector of the form

```
 e1 e2
```

where `e1` is the containing element and `e2` is the contained element. The associated rule then applies only to cases where element `e1` is an ancestor of `e2`—that is, either `e2` is a child of `e1`, or a child of a child, et cetera.

For example, to use maroon letters for emphasized text (the `em` tag) anywhere within a level 1 heading (the `h1` tag), you would use a CSS rule like this:

```
 h1 em { color: maroon; }
```

You can stack any number of element names in a rule like this. For example, the selector “`ul ul ul`” would apply to third-level bullet lists (that is, bullet lists within bullet lists within bullet lists).

### 7.4. Child selection

Another way to select elements by context is to specify that a certain element must be an immediate child of some parent element. Use a selector of this form:

```
 e1 > e2
```

where `e1` and `e2` are element names. In this form, the rule applies only to `e2` elements that are the direct children of `e1` elements.

For example, the selector “`h1 > em`” would apply to an `em` element whose parent element is `h1`, but it would not apply to an `em` element inside a `cite` element inside an `h1` element.

### 7.5. Adjacent element selection

You can also write a selector that applies to an element only when it is preceded by a certain other type of element.

The general form is

```
 e1 + e2
```

where `e1` and `e2` are element names. In this form, the rule applies only to any `e2` elements that immediately follows an `e1` element.

For example, the selector “`div.q+div.a`” would apply to any `<div class="a">...</div>` element that immediately follows a `<div class="q">...</div>` element.
7.6. Selecting by attribute values

You can write rules that apply only to elements with certain attributes. There are several forms of this rule.

\([att]\]

Applies only to elements that have an attribute named \(att\).

For example, the selector

```
    h2[rating]
```

would apply only to h2 elements that have a rating attribute.

\([att=value]\]

Applies only to elements that have an attribute named \(att\) whose value is \(value\).

For example, this selector

```
    p[role="panic"]
```

would apply only to paragraphs starting with a tag like "<p role="panic">".

\([att~=value]\]

Applies to elements that have an attribute named \(att\) whose value is a space-separated list of values including the given \(value\).

For example, a paragraph that started with tag `<p phobias='acro claustro arachno agora'>` would be affected by a selector "p[phobias~="arachno"]".

\([att|=value]\]

Applies to elements that have an attribute \(att\) whose value starts with \(value\), optionally followed by a hyphen and other characters. This particular selector was intended for style markup that applies only to specific languages. For example, this rule

```
    *[lang|="en"]
```

would apply to any element that has a lang attribute that starts with "en". Thus it would apply to the various English variants such as lang="en-us" (U. S. English), lang="en-uk" (British English), and so forth.

7.7. Selecting specific single elements by ID

You can also make a rule apply to one specific element. Tag the element with an attribute `id='I'` where \(I\) is some unique identifier made of letters, digits, and hyphens. As with the `class='...'` attribute, any HTML element can have an `id='...'` attribute. Then use a stylesheet rule of the form:

```
    element#I
```

where \(I\) is the identifier.

7.8. Pseudo-classes

You can use a `pseudo-class` in a selector. Unlike the selectors we’ve already discussed, pseudo-classes are used to select parts of the content by their state or position or other qualities that are not related to what elements or tags they are part of.
Pseudo-class names are always preceded by a colon (:). Pseudo-classes include:

- **:active**
  Affects an element during the time the user is clicking on it.

- **:focus**
  Affects the element that has focus, that is, the element that would currently receive any keyboard input.

- **:hover**
  Affects the appearance of an element while the mouse is on top of it, but not clicked.

- **:lang(code)**
  Affects only elements with a language that matches the given code. For example, this rule would display any French elements in blue:

  ```css
  *:lang(fr) { color: blue; }
  ```

- **:link**
  Affects the appearance of links that have not been visited. For example, this rule would make unvisited links appear in purple:

  ```css
  a:link { color: purple; }
  ```

- **:visited**
  Affects the appearance of links that have been visited.

### 7.9. Pseudo-elements

A pseudo-element is a selector that refers to a specific area of the page. There are two types of pseudo-elements:

- The :first-letter and :first-line pseudo-elements refer to specific parts of a paragraph. There is no other way to refer to the first line of a paragraph, since we only know what is in that line when the page is actually rendered.

- The :before and :after pseudo-elements allow you to add content before and after some other element.

Here are the pseudo-elements in CSS2.

- **:after**
  Displays specific content after an element. Use the content property of the rule to specify what text is displayed after an element.

  For example, this rule would display two closing square brackets after any level-2 heading:

  ```css
  h2:after { content: "\]" }
  ```

- **:before**
  Like :after, but this selector displays specific content before an element. Use the content property of the rule to specify what text is displayed after an element.

  For example, this rule would display two opening square brackets before any level-2 heading:

  ```css
  h2:before { content: "[" }
  ```

One common use of this selector is to insert generated content, such as section numbers. See Section 16, “The content property: Specifying content in pseudo-elements” (p. 29).
**:first-letter**
Affects the appearance of the first letter of an element. For example, this rule:

```
p:first-letter { font-size: large }
```
would display the first letter of each paragraph in a large size. In combination with other properties, you can use this pseudo-class to get a “drop cap” effect (starting a paragraph with one large letter).

**:first-line**
Affects the appearance of the first line of a block element (such as a text paragraph). For example, this rule:

```
p:first-line { text-transform: capitalize}
```
would display the first line of every paragraph all in capital letters.

### 7.10. The universal selector
The selector “*” applies to all elements. For example, to make text green everywhere:

```
* { color: green; }
```
This selector is the least specific selector, so any more specific selector will override it. See Section 8.2, “Specificity: Which selector applies?” (p. 13).

### 8. When rules collide
Style rules may come from multiple sources, and each source may have more than one rule that might apply to a given element of your page. How are these collisions resolved?

- There are three possible sources for stylesheet rules: see Section 8.1, “Cascading” (p. 12).
- When more than rule from a given source might apply, see Section 8.2, “Specificity: Which selector applies?” (p. 13).
- Not every property is specified for every element. In such cases, many properties are inherited from other rules. See Section 8.3, “Inheritance” (p. 13).

#### 8.1. Cascading
Up to three style sheets may apply to a page.

- The user can provide their own stylesheet.
- The author of the page may attach a style sheet to the page.
- The browser may have a built-in style sheet that applies if neither the user nor the author has provided one.

Normally, the page author’s style sheet has the highest priority, followed by the user’s, with the browser’s style last.

However, a rule can override rules from other, higher-priority style sheets by including the special keyword “!important” after the property value. For example, suppose this rule appears in the user’s style sheet:

```
p.admon { font-style: italic !important; }
```
Paragraphs with class='admon' would be set in italics, even if the author’s style sheet specifies a different font-style value.

Here is the ranking of all possible sources with and without the !important keyword, from highest to lowest priority:

- User stylesheet with !important.
- Author stylesheet with !important. (This has changed since the CSS-1 specification, which gave author rules precedence over user rules.)
- Browser stylesheet with !important.
- Author stylesheet without !important.
- User stylesheet without !important.
- Browser stylesheet without !important.

8.2. Specificity: Which selector applies?

When more than one selector applies to a given element, the more specific selector is used. Here are the rules for calculating specificity:

- Selectors that refer to a specific element by its ID value (see Section 7.7, “Selecting specific single elements by ID” (p. 10)) are the most specific.
- Selectors that use non-ID attributes and pseudo-classes are less specific.
- Selectors containing element names are less specific still. The fewer element names, the less specific. So, for example, selector ‘em’ is less specific than “h1 em”.
- The universal selector “*” is least specific.

8.3. Inheritance

Most properties inherit from the properties of parent elements. That is, a property that applies to an element generally applies to all of the elements inside it.

For example, the HTML body element is the parent element for the content of a web page. So a rule such as

```
body { color: green; }
```

would apply to the entire page unless overruled by a more specific rule at a lower level.

Some properties, however, do not inherit. For example, the background property determines what appears behind an element. The default value of this property is transparent, so if there is no specific rule applying to the background of an element, its background is invisible and you see the background of the parent element behind it.

9. Declarations

The declaration part of a CSS rule is one of the parts inside the curly braces { … } following the selector. Each declaration has the form

```
property: value
```

where the property is some aspect of page rendering, and the value specifies how that property should be rendered.

Following sections discuss the various properties and the values they take.
10. Font properties

First, let's define some common terms:

- A type family refers to a related group of typefaces. For example, the Times family came from newspaper practice.
- A typeface refers to all of the characters that have the same style (weight, width, posture, and name). Example: Bodoni Bold Extended.
- A font refers to a typeface in a particular size, such as Bodoni Bold Extended 12pt.

Attributes of a font include:

- General class: serif, sans-serif, script.
- Proportional (different characters may have different widths) or monospaced (all characters have the same width).

One of the problems of Web page design is that you can't assume that any given font exists on any given reader's system. For this reason, it is best to specify a set of fonts, listing your preferred fonts first, but also providing more generic alternatives for systems without your favorites.

The section below lists some of the CSS font properties.

10.1. The font-family property

This property enumerates the font family or families that you want. You can use specific font names or one of the generic font names serif, sans-serif, monospace, cursive, or fantasy. Example:

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

Note that family names containing spaces must be enclosed in quotes. This rule says to use Garamond if available; use Times if Garamond is not available; and so on, using the generic family serif of none of the named fonts are available.
10.2. The **font-style** property

Allowable values are *normal* for vertical text (the default), *italic* for italics, or *oblique* for fonts that look like regular text, only slanted.

10.3. The **font-variant** property

The default for this attribute is *normal*, but you may specify a value of *small-caps* to get a caps-and-small-caps font.

10.4. The **font-weight** property

This property specifies how heavy or bold the font is. The default value is *normal*, but you can instead give values of *bold*, *bolder* (meaning a bit bolder than the parent's weight), or *lighter* (again, relative to the parent's weight). You can also specify the weight as one of the values **100, 200, ..., 900**, where 100 is the lightest weight, 400 is normal, and 700 is the same as *bold*. Not all values are available in any given font.

10.5. The **font-size** property

You can specify the size of a font in four general ways:

- As an absolute size. Permissible values range through **xx-small** (extra extra small), *x-small*, *small*, *medium*, *large*, *x-large*, and **xx-large**. The difference between each size is about a factor of 1.5.

  We discourage using absolute sizes because they make it impossible for large-print users to resize the fonts.

- As a relative size, compared to the parent font. Two values are supported: *larger* and *smaller*. Example:

  ```
  body { font-size: medium; }
  h2 { font-size: larger; }
  p { font-size: smaller; }
  ```

  In the example above, an h2 element inside a body element would be displayed in a larger font than medium, and a p element would use a smaller font.

- By naming a specific font size. The units allowed are discussed elsewhere; see Section 6.1, “Dimensions” (p. 5). Here is an example:

  ```
  pre.special { font-size: 14pt; }
  ```

  This rule would set in 14-point type all elements of class special inside pre elements.

- As a percentage of the parent font size. Example:

  ```
  em { font-size: 120%; }
  ```

  This rule would set text inside an em element 20% larger than the parent font.

10.6. The **font** property

You can combine all the font properties together in a single property called **font**. Here is an example:
11. The **display** property: What kind of box is this?

Certain tags such as `p` and `blockquote` are always set as *block-level elements*, which means that they are always preceded and followed by a line break.

Other tags, such as `em` and `a` are *inline elements*: the beginning and ending of these elements does not force a line break.

Still other tags, such as an `li` element inside a `ul` element, are displayed as *list items*, preceded by a bullet (or number, as inside an `ol` element).

You can change the way a given element is displayed by using a rule that sets its `display` property to one of these values:

- **block**
  - Force the element to be displayed as a block element, with line breaks before and after it.

- **inline**
  - Display the element inline. This element can be put on the same line with other inlines, and may be broken over multiple lines.

- **list-item**
  - Display the element as a list item. Each item is a principal box, optionally preceded by a marker.

- **none**
  - Don’t display this element at all. Don’t even leave space for it in the rendering.
  
    For example, the rule
    
    ```css
    p.optional { display: none; }
    ```
    
    would cause an element of the form `p class='optional'` to disappear altogether.

- **marker**
  - This box type is used with the `:before` and `:after` pseudo-elements to display generated content like the bullets or item numbers before list items.

- **run-in**
  - If this element is followed by a block element, treat this element as the first inline box of that block.
  - If not followed by a block element, treat this element as a block element.

- **compact**
  - Render this element in the left margin. If its content fits entirely in that margin, place it there, and any following block will appear to its right.
  - If the content doesn’t fit in the margin, render it as a separate block.

- **table**
  - Render this element as if it were table.

- **inline-table**
  - Render this element as a table, but make the entire table act as an inline box.

- **table-row, table-row-group, table-header-group, table-footer-group, table-column, table-column-group, table-cell, table-caption**
  - Specify one of these values to make the element act as if it were part of a table. For more information, see Section 18, “Tables” (p. 32).
12. Other text properties

In addition to font selection, CSS has a number of other properties for controlling the presentation of text.

12.1. The line-height property

The line-height property controls the “leading,” that is, the vertical distance between the baselines of two adjacent lines.

The property can take any of these values:

- normal: Set the line height to the default size.
- A number (with or without a decimal) to specify the multiple of normal line height. For example, line-height: 2 specifies double-spacing; line-height: 1.2 would give you 20% more than normal spacing.
- A number followed by a percent sign gives you that percent of the normal leading. For example, line-height: 150% specifies 1.5 times the normal leading.
- A dimension (see Section 6.1, “Dimensions” (p. 5)) sets the leading to that dimension. For example, line-height: 12pt gives you 12-point leading.

12.2. The text-indent property

Use the text-indent property to control the indentation of text paragraphs (and text in block elements generally). You can use two kinds of values for this property:

- Use a dimension (see Section 6.1, “Dimensions” (p. 5)) to specify the size of the indentation directly.
- A number followed by a percent sign sets the indent size to that percentage of the width of the browser window. For example, text-indent: 10% specifies that paragraphs should be indented one tenth of the width of the browser window.

12.3. The text-align property

Normally, text is set “ragged right”, with a straight left margin. You can get different paragraph shapes by setting the text-align property to one of these values:

- left for a straight left margin and a ragged right margin, the default appearance;
- center to set the text with both left and right margins ragged;
- right for straight right and ragged left margins; or
- justify for straight left and right margins.

12.4. The text-decoration property

This property is used to add certain effects to text:

- underline adds an underline below the text.
- overline places a line over the text.
- line-through sets the text in “strike-out” type, with a line through the text. This style is often used to show that is being deleted in a particular revision.
- blink makes the text blink on a Web page. This effect can be extremely annoying and should be used sparingly, if at all.
12.5. The **text-transform** property

These properties can change the case of text:

- The **capitalize** value capitalizes the first letter of each word.
- The **uppercase** value changes all letters to capitals.
- The **lowercase** value changes all letters to lowercase.

12.6. The **white-space** property

You can control how white space (spaces and tab characters) are treated by setting the **white-space** property to one of these values:

- **normal** treats whitespace in the usual way for its enclosing element.
- **nowrap** collapses each region of whitespace within a line to a single space and ignores line breaks, setting the entire content as one long line.
- **pre** treats text as in the **pre** element: all whitespace and line breaks are displayed exactly as they appear in the HTML source file.

12.7. The **letter-spacing** property

For extra emphasis, you can use the **letter-spacing** property to display text with extra spaces between the letters. The value can be:

- **normal**: this cancels any letter-spacing.
- A dimension (see Section 6.1, “Dimensions” (p. 5)) that says how much extra space you want between letters.

For example, this rule:

```
    h1.shout { letter-spacing: 1pt; }
```

would add one point of horizontal space between the letters of any heading element `<h1 class='shout'>...<h1>.

12.8. The **word-spacing** property

Use the **word-spacing** property to get extra spacing between words. Values can be:

- **normal**: this cancels any word-spacing.
- A dimension (see Section 6.1, “Dimensions” (p. 5)) that says how much extra space you want between words.

12.9. The **vertical-align** property: Shifting the baseline

This property allows you to shift text vertically relative to its normal baseline. Values may be any of:

- **baseline**: This text’s baseline is the same as the baseline of the block that contains it. This is the default value.
- **middle**: The vertical center of this text is aligned with the center of the containing block. The center of a line of text is above the baseline by an amount half the x-height of the text. The x-height of a font is the height of a character with no ascenders or descendents, such as “x”.
sub
Shift this text down as if it were a subscript. Note that this does not change the size of the text.

super
Shift this text up as if it were a superscript. This property does not change the size of the text.

text-top
The top of this text will be vertically aligned with the top of text in the containing block.

text-bottom
The bottom of this text will be vertically aligned with the bottom of text in the containing block.

int%
Raise this text an amount specified as the percentage of the current line-height. For example, if the current line-height is 12 points, this declaration

```css
vertical-align: 25%;
```

would raise the text 3 points.

dimension
Raise the text by the given dimension. For example, this declaration

```css
vertical-align: -2mm;
```

would lower the text by two millimeters. See Section 6.1, “Dimensions” (p. 5).

12.10. The quotes property: Specifying quote characters

The quotes property is used to declare one or more pairs of quote symbols to be used in setting off quoted text. The default is to use double quotes “…” for the first level of quotation, and single quotes ‘…’ for quotes within quotes.

The value for this property may be either of:

• A list of character strings to be used as quote symbols. The first two strings are used as the first-level open and close quote characters. The next two characters are the second-level open and close quote characters.

• The keyword none, if you don’t want any quote symbols to appear.

For example, this declaration would set up the British convention of single quotes at the outermost level and double quotes at the second level:

```css
quotes: "'" "'" "'" "'
```

The quote symbols you set up with this declaration are invoked when the content property contains open-quote and close-quote values. For more on this property, see Section 16, “The content property: Specifying content in pseudo-elements” (p. 29).

13. The color property

This property sets the color of the text. The value can be any one of the methods for specifying colors (see Section 6.2, “Specifying colors” (p. 6)). For example, the rule

```css
h3 { color: maroon; }
```
would display all level h3 headings with maroon letters.

14. The background properties

These properties are used to display background images or colors behind other elements.

**background-color**
Sets the background color. Values may be:
- A color name. See Section 6.2, “Specifying colors” (p. 6).
- **transparent** to make the background transparent, so it displays whatever is behind it.

**background-image**
Displays an image as the background. Use the `url()` function to select the image; see Section 6.4, “Universal resource identifiers (URIs)” (p. 7).

**background-repeat**
When you use a background image and the image is not large enough to fill the space, the default behavior is to tile the image, that is, to repeat it in rows and columns.

You can control what happens in this case by setting the `background-repeat` property to one of these values:
- **repeat**: This gets you the default behavior.
- **repeat-x**: If you use this value, the image will be repeated horizontally, but not vertically.
- **repeat-y**: The image will be repeated vertically, but not horizontally.
- **no-repeat**: The image will not be repeated at all.

**background-attachment**
By default, when you scroll a page, the background image scrolls along with the content. To change that, set the `background-attachment` property to one of these values:
- **scroll**: this is the default behavior.
- **fixed**: the background stays in the same position in the browser window, as the the content scrolls.

**background-position**
Normally, a background image is positioned in the top left corner of the page. To change that, set the `background-position` property to two values from this list; the first value sets the x (horizontal) position, and the second the y (vertical) position:
- Use a dimension (see Section 6.1, “Dimensions” (p. 5)) to specify the offset from the edge of the page: this is the distance from the left side for the x position and the distance from the top for the y position.
- Use a number followed by a percent sign (%) to specify how far the image is shifted relative to the top left corner. A value of 0% aligns the image at the top or left edge; a value of 100% aligns the right or bottom of the image with the right or bottom of the page; and other percentages are placed linearly between these extremes.
- **center**: If used for the x position, the image is centered horizontally; for the y position, it is centered vertically.
- **left**: This value should be used only for the x position. It has the effect of moving the image all the way to the left.
- **right**: Moves the image all the way to the right.
- **top**: This value should be used only for the y position. It has the effect of moving the image all the way up.
- **bottom**: This value should be used only for the y position. It has the effect of moving the image all the way down.
For example, this rule

```
body { background-position: 50% 0.5in; }
```

would center the image horizontally, and move it down half an inch from the top edge.

**background**

You can specify all of the above properties in a single rule with the `background` property. The list of values for the property can include any of the values shown above for `background-color`, `background-image`, `background-repeat`, `background-attachment`, or `background-position`.

For example:

```
body { background: 50% 0.5in scroll repeat-y maroon url(http://clip-arf.dog/schnauzer.gif); }
```

### 15. Designing with box elements

The next important set of properties deals with the appearance of a generic “box element.” Box elements are block-level elements, so they are always preceded and followed by a line break.

An ordinary text paragraph element `<p>`... is a box element, but you can make your own box elements using the `<div>`... element.

You should use `<div>` instead of `<p>` whenever your box elements are part of something other than a regular text paragraph, e.g., as part of a complex page layout. This allows style sheets to manipulate regular text paragraphs with selectors like `<p>` and `<p>someClass>` without affecting other box elements.

Here is an illustration of CSS's general box model. From inside out, you can specify:

- The background of the block's `content` may be a color or an image set either by the `background` property of the block, or inherited from some containing block.

- `Padding` separates the box's content from what is outside. It also shows the background color or image, if any. The default padding size is zero.

- An optional `border` can be drawn around the outside of the padding area. This can be a solid, dashed, or dotted line of any color or thickness. The default is to have no border.
• The margin is an area that is supposed to be kept clear. The margin area is always transparent, so whatever is behind it will show through.

When two such boxes are stacked vertically adjacent, their margin areas may overlap, but never their borders, padding, or content.

In this diagram, distance \( d \) is either the bottom margin of box A, or the top margin of box B, whichever is larger. (In some cases, margins may not overlap. Refer to the standard for pages of mind-numbing detail.)

For each box property, you can specify different values for top, bottom, left, and right. For example, the same box can have a thick black solid top border, a thin red dotted bottom border, an inch-thick dotted left border, and no right border at all.

Box properties fall into several groups:

- Section 15.5, “The padding properties” (p. 23).
- Section 15.6, “The border properties” (p. 24).
- Section 15.7, “The margin properties” (p. 25).
- Shape properties describe the size of the box, if necessary. The default size of a box is the full width of the browser window with just enough height to accommodate the contents. These properties include height and width. See Section 15.2, “The height and width properties” (p. 23).
- Positioning properties specify where the browser is allowed to place the box relative to other elements. They include clear and float. See Section 15.3, “The clear property” (p. 23) and Section 15.4, “The float property” (p. 23).

15.1. Side lists

A number of box properties allow you to specify properties of the four sides of a box independently. A side list is a list of one, two, three or four items that describes properties of all four sides. The table below shows how the values are translated to the four sides:

Table 1. Interpretation of side lists

<table>
<thead>
<tr>
<th>Side list</th>
<th>Top</th>
<th>Right</th>
<th>Bottom</th>
<th>Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
<td>a</td>
</tr>
</tbody>
</table>
15.2. The height and width properties

You can specify the height of a box, its width, both, or neither.

Both these properties allow the same three ways of specifying the values:

- **auto**: This attribute makes the box its natural size.
- As a dimension (see Section 6.1, “Dimensions” (p. 5)).
- As a number followed by a percent sign. The number is interpreted as a percentage of the containing block width, both for height and width.

Four more properties allow you to constrain the size of a box:

- **min-width**: Minimum box width.
- **max-width**: Maximum box width.
- **min-height**: Minimum box height.
- **max-height**: Maximum box height.

These four properties take the same values as the height and width properties, with one exception. You can specify a value of none for max-height or max-width, to tell the browser that there is no upper limit on those sizes.

15.3. The clear property

Use this property to tell the browser whether you want to allow the box to be placed to the side of previous elements, or whether to move it down below any previous elements. Values are:

- **both**: Move the box below any floating elements that precede it.
- **left**: Move the box below any preceding element that is floating to its left.
- **right**: Move the box below any preceding element that is floating to its right.
- **none**: Allow this box to be positioned beside any preceding floating elements.

15.4. The float property

You can specify a float value for a box to tell the browser that it can allow following elements to flow around it or be positioned to its sides. Values are:

- **none**: Don’t allow following elements to flow around or float beside the box.
- **left**: Position this box to the left side of the available space and allow other elements to flow around it or be placed on its right.
- **right**: Position this box to the right side and allow other elements on its left.

15.5. The padding properties

Padding is extra space added around the content of a box. The default padding is none at all.

The amount of padding can be specified in two ways:
• As a dimension (see Section 6.1, “Dimensions” (p. 5)). For example, `padding-right: 2pc` would add two picas (about a third of an inch) of space between the contents of the box and its right side.
• As a number followed by a percent sign (%). The number is interpreted as a percentage of the width of the browser window, even for `padding-top` and `padding-bottom`.

The usual way of specifying padding is to specify the amount of padding on all four sides in one rule like this:

```
padding: side-list;
```

where the `side-list` is as described in Section 15.1, “Side lists” (p. 22).

For example, to specify an extra 1.5 picas of padding:

```
padding: 1.5pc;
```

To specify one em at the top and bottom, and two ems of padding on the sides:

```
padding: 1em 2em;
```

You can also specify the padding for only one side with these properties:

• `padding-top`
• `padding-left`
• `padding-bottom`
• `padding-right`

## 15.6. The `border` properties

The default box rendering is not to have a border at all. To make a border appear around a box, you must specify at least a border width and a border style.

Each side of the border can be any of these values:

• thin for a hairline border; medium for the normal border; or thick for a slightly thicker border.
• The line thickness as a dimension (see Section 6.1, “Dimensions” (p. 5)).

Here are the border properties and their values.

**border-width**

The simplest way to specify a border width is with the `border-width` property. The value of this property is a side list; see Section 15.1, “Side lists” (p. 22).

For example, this rule would render paragraphs of class `warning` with a border 0.2 ems wide on the top and left, and 0.3 ems wide on the bottom and right:

```
border-width: 0.2em 0.3em 0.2em 0.3em;
```

To specify the border widths independently, use these properties:

• `border-top-width`
• `border-right-width`
• `border-bottom-width`
• `border-left-width`

**border-style**

To set the style of all four sides of the border at once, use the `border-style` property. The value of this property is a side list (see Section 15.1, “Side lists” (p. 22)).

Available border styles include:
• none: This is the default style—no visible border.
• solid: The border is a single solid line.
• double: A double solid line.
• dashed: Displays a dashed border.
• dotted: A border of dots.
• inset: A 3-d border that makes the contents look like they're set more deeply into the page.
• outset: A 3-d border that makes the contents look raised against the background.
• groove: A 3-d effect that makes the border look inset.
• ridge: A 3-d effect that makes the border look outset.

You can also set the styles of the four sides of the borders individually using these properties:
• border-top-style
• border-left-style
• border-bottom-style
• border-right-style

**border-color**

By default, borders are black. To specify a different border color, use the `border-color` property. The value of this property is a side list (see Section 15.1, “Side lists” (p. 22). See Section 6.2, “Specifying colors” (p. 6)) for the various ways of specifying colors.

You can specify the color of each side of the border individually using these properties:
• border-top-color
• border-left-color
• border-bottom-color
• border-right-color

**border**

You can specify multiple properties for any side of the border using these property names:
• border-top
• border-right
• border-bottom
• border-left

You can set each of the above properties to a list containing a border thickness (as thin, medium, thick, or a dimension), border style (using any of the values legal for `border-style`), or border color (as in `border-color`).

You can specify all of the border properties of a box element by using the `border` property. Set the value of this property to a list containing any of the values allowed for `border-style`, `border-color`, or `border-width`.

For example, this rule

```css
div.orchid { border: 1px maroon dashed; }
```

would display `<div class='orchid'>…</div>` elements with a one-pixel thick, maroon, dashed border.

### 15.7. The margin properties

Use the `margin` property of a box to add extra space outside the box (and outside the border, if there is one).

Values can be any of:
• auto: Specifies the default value, no extra spacing.
• A dimension (see Section 6.1, “Dimensions” (p. 5)). For example, margin-left: 10mm would add an extra 10 millimeters of margin outside the left side of the box.
• A number followed by a percent sign (%), interpreted as a percentage of the browser window width (even for the top and bottom margin values).

Most commonly, all four margins are set at once with the margin property. The value of this property is a side list (see Section 15.1, “Side lists” (p. 22)).

For example, this rule specifies a one-em margin around all div elements:

```css
div { margin: 1em; }
```

To specify the margin on a specific side of the box, use one of:

• margin-top
• margin-right
• margin-bottom
• margin-left

15.8. The overflow property: What if it doesn’t fit?

This property allows you to specify what happens when content won’t fit inside a given block. These values are allowed:

visible
   Display all the block’s content, even if it is outside the block.

hidden
   Clip (hide) any content that won’t fit inside the block.

scroll
   Assuming that this content is being rendered in a continuous medium (as opposed to fixed-size pages), always provide vertical and horizontal scrollbars so that the user can scroll to see the full content.

auto
   If the content fits within the box, display it all without scrollbars. If the content overflows the box, then provide scrollbars.

See the related property, Section 15.9, “The clip property: Specify a clipping rectangle” (p. 26).

15.9. The clip property: Specify a clipping rectangle

Use this property if you want to specify how far content can extend outside the box before it is clipped. A separate property controls whether or not clipping occurs: see Section 15.8, “The overflow property: What if it doesn’t fit?” (p. 26).

The clip property takes either of these values:

auto
   Use the size of the box to clip overflowing content. This is the default value.

rect(top, right, bottom, left)
   Use this value if you want to specify exactly where the clipping rectangle is relative to the containing box. The four dimensions specify how far the clip rectangle is inside the top, right, bottom, and left sides of the containing box, respectively. Each value is a dimension (see Section 6.1, “Dimen-
15.10. The **visibility** property: Can we see the content?

This property allows you to reserve space in the layout for some element, but without rendering it. Values may be either of:

- **visible**
  - Display the block normally. This is the default value.

- **hidden**
  - Leave space for the block, but don't display any of its content.

15.11. The **position** property: Positioning boxes

You can control where a box appears by setting its `position` property. Values include:

- **static**
  - This is the default option: the box appears as part of the regular flow of boxes.

- **relative**
  - The box's position is shifted relative to what would be its normal position. To specify how far it is shifted, see Section 15.12, “The box offset properties: top, bottom, left, and right” (p. 27).

- **absolute**
  - The box's position is shifted relative to the box that contains it. To specify how far it is shifted, see Section 15.12, “The box offset properties: top, bottom, left, and right” (p. 27).

- **fixed**
  - When the content is rendered in continuous media, its position is fixed relative to the viewport. On paged media, its position is fixed relative to the page box.

15.12. The box offset properties: top, bottom, left, and right

When a box's `position` property is not **static** (see Section 15.11, “The position property: Positioning boxes” (p. 27)), these four properties specify where the box's edges are relative to some other box:

- **top**
- **bottom**
- **left**
- **right**

Each property takes one of these values:

- **dimension**
  - Shift the box's edge inward by the value of this dimension; see Section 6.1, “Dimensions” (p. 5).
  - A negative value shifts the box's edge outwards.

  For example, these declarations would shift a box 5mm to the right:
Shift the box's edge as a percentage of some value. For relatively positioned boxes, this is a percentage of the box to be rendered. For absolutely positioned boxes, it is a percentage of the size of the containing box. For example, these declarations move a box to the right 10% of its width:

```css
position: relative;
right: -5mm;
```

The default value: do not shift this box edge.

**15.13. The z-index property: Stacking order**

When elements overlap each other, you can use the `z-index` property to specify which elements are in front and which are in back.

The `z-index` property is a number that specifies the stacking order of a box. Higher numbers are “in front” and may obscure other elements with lower `z-index` values. The base value is zero, and values may be negative.

The `z-index` property may take any of these values:

- **auto**
  - The box has the same `z-index` as its containing box.

- **int**
  - When you specify an integer value, the element is stacked at that given level. Also, this element becomes the base for a new local stacking context; see the explanation below.

Boxes are stacked according to their `z-index` value, with higher numbers in front of lower numbers.

However, each box that uses a specific `z-index` value establishes a local stacking context for boxes that are inside of it. This means that any internal boxes are stacked relative to each other according to their `z-index` values, but they are all stacked in the same group as the parent box, regardless of how their `z-index` values compare with boxes other than the parent box.

Here’s an example. First, a fragment of HTML:

```html
<div id="d1">
  <img src="i1.jpg">
</div>
<div id="d2">
  <img src="i2.jpg">
  <div id="d3">
    <img src="i3.jpg">
  </div>
</div>
```

And here’s a stylesheet for that:

```css
div
{
  position: absolute;
  top: 0in;
  left: 0in;
}
```
The first rule says that all div elements are absolutely positioned at the top left corner of the page. The next three rules assign z-index values to the div elements in the HTML.

There are two different stacking contexts here. Block d1 and block d2 are in the outermost (global) stacking contexts, so block d1, with a z-index value of 5, is in front of d2 with a z-index of 3.

Block d3, however, is inside block d2's local stacking context, so block d2 and everything inside it are stacked together. Block d3's z-index value of 15 only influences how it stacks relative to any other elements inside of block d2.

Therefore, image i3.jpg, with a z-index of 15 inside that local stacking context, stacks in front of image i2.jpg, which has the base z-index of 0 inside that local context.

However, on the final page, image i1.jpg still appears in front of i3.jpg, because it has a higher z-index in the global stacking context.

16. The content property: Specifying content in pseudo-elements

Use the content property inside the :before or :after pseudo-elements to add content before or after some other element.

The value of this property consists of one or more items from this list, concatenated.

string
The literal string is inserted into generated content. For the form of literal strings, see Section 6.3, “String constants” (p. 6).

url("URI")
Insert the content (e.g., an image) from the specified URI. See Section 6.4, “Universal resource identifiers (URIs)” (p. 7).

counter(name)
Insert the value of the counter with the given name.

counter(name, style)
Insert the named counter, but present its value using the given style. The style codes are discussed in Section 17.1, “The list-style-type property” (p. 31).

counters(name, string)
Insert all the counters with the given name, separated by the given string. This is used for hierarchical section numbers such as “3.1.4”; you use the same counter name at each level, and specify “.” as the separator character.

counters(name, string, style)
To change the counter style from the default presentation as an integer, specify a style code, such as lower-roman for lowercase Roman numerals. The complete list of style codes is given in Section 17.1, “The list-style-type property” (p. 31).

open-quote
Adds the open quote character for the current level of nesting. By default, double quote (“”) is used at the outer level, and single quote (‘) for a quote inside a quote. If you want to specify different quotes, see Section 12.10, “The quotes property: Specifying quote characters” (p. 19).
Use of this value increments the quote-nesting level.

**close-quote**
Adds the closing quote character for the current level of nesting. Use of this value decrements the quote-nesting level.

**no-open-quote**
Like `open-quote`, this value increments the quote-nesting level, but it does not generate any content.

**no-close-quote**
Like `close-quote`, this value decrements the quote-nesting level, but generates no content.

**attr(a)**
Generates the value of the `a` attribute of the element to which this rule applies.

For example, suppose your document's `div` elements of class `dog` have an attribute `dogname`. This rule would insert the value of that attribute, in parentheses, as content after the rendering of the `div`:

```css
div.dog
{
    :after
    { " ( attr(dogname) )" }
}
```

These properties are used only inside generated content:

- Section 16.1, “The `counter-reset` property” (p. 30).
- Section 16.2, “The `counter-increment` property” (p. 30).

### 16.1. The `counter-reset` property

Use this property inside a `:before` or `:after` pseudo-element to a counter to some specific value. Counters are used for tasks such as numbering sections of a document. You will pick a name for each counter; names must conform to the usual rules for HTML and XML elements.

The `counter-reset` property may have any of these values:

**name**
The named counter is reset to zero.

**name int**
The named counter is set to the value specified by the integer `int`.

Examples:

```css
counter-reset: chapter-no;
counter-reset: section-no 10;
```

The first example sets the counter named `chapter-no` to zero. The second example sets counter `section-no` to ten.

### 16.2. The `counter-increment` property

This property adds some number to a counter. Values may have either of two forms:
**name**

The named counter is incremented by one.

**name int**

The given integer is added to the named counter.

Examples:

```css
counter-increment: chapter-no;
counter-increment: section-no 10;
```

The first example adds one to the counter named `chapter-no`. The second example adds ten to the counter named `section-no`.

### 17. The list properties

You can customize the appearance of HTML `ul` (unnumbered or “bullet” list) and `ol` (ordinal or numbered list) elements with the CSS properties below.

#### 17.1. The `list-style-type` property

Some of the choices below apply to unnumbered lists and some to numbered lists:

- **circle**: Display an empty circle before each list element.
- **disc**: Display a filled circle before each list element.
- **square**: Display a square before each list element.
- **decimal**: Display an item number in decimal numerals before each list element: 1, 2, 3, ....
- **lower-alpha**: Items in a list are numbered as a, b, c, ....
- **upper-alpha**: Items in a list are numbered as A, B, C, ....
- **lower-roman**: Items in a list are numbered as lowercase Roman numerals: i, ii, iii, iv, ....
- **upper-roman**: Items in a list are numbered as uppercase Roman numerals: I, II, III, IV, ....

In addition to the options shown above, there are options specific to Hebrew, Georgian, Armenian, Chinese, Japanese, and Korean languages. Refer to the standard[^1] for full details.

#### 17.2. The `list-style-image` property

You can use the `list-style-image` property to display an image of your choice in place of the bullet in a bullet list, or suppress the bullet altogether. This property takes precedence over the `list-style-type` property described above.

- **none**: If you set `list-style-image: none` for a list item, no bullet will be displayed in front of that list item.
- **url(u)**: Display the image located at URI `u` in place of the bullet.

#### 17.3. The `list-style-position` property

Normally, the second and following lines of a list item will start lined up vertically with the text on the first line, that is, after the bullet or section number. The bullet will appear in its own vertical space outside the list text.

[^1]: [http://www.w3.org/Style/CSS/](http://www.w3.org/Style/CSS/)
This behavior is selected by setting `list-style-position: outside`.

However, if you set `list-style-position: inside`, the second and following lines will start lined up with the left edge of the bullet (or section number); that is, the bullet or section number will be run into the text.

17.4. The `list-style` property

You can set more than one of the above list properties in a single rule by setting the `list-style` property to a list of values allowed for the `list-style-position`, `list-style-image`, and `list-style-type` properties.

For example, this rule

```css
ul.pterry { list-style: disc inside; }
```

would display any list tagged as `ul class='pterry'` using disc-shaped bullets run in to the text.

17.5. The `marker-offset` property

In a list, the item that appears before the first line of the list is called the `marker`. You can use the `marker-offset` property to specify the horizontal distance between the marker and the first line. The value of this property is a dimension (see Section 6.1, “Dimensions” (p. 5)). For example, the declaration `marker-offset: 2em;` would insure that two ems of space separate the marker from the first line of the list item.

18. Tables

CSS allows you to format elements as a table. A table consists of an optional caption and one or more `rows` and `columns`. Each item in the grid is called a `cell`. Additionally, you can combine multiple rows into `row groups` that share style features, and you can combine multiple columns can into `column groups`.

You can designate one or more rows at the top of the table as a `table header group`. You can also designate one or more rows at the bottom as a `table footer group`. When a table has a table header group or table footer group, and the table is rendered on paged media, the header group and footer group will be repeated at the top and bottom of each page, so the reader can see what the columns mean.

To present some element of your document act as one of these parts of a table, set the `display` property for that element to one of these values:

- `table`
  Renders the element as a block containing a table.

- `inline-table`
  Renders the element as a table, but encloses the entire table in an inline element, so it can be placed on a line with other inlines.

- `table-row`
  The element acts as one row of a table.

- `table-row-group`
  The element is presented as a row group.

- `table-header-group`
  The element is a group of header lines.
**table-footer-group**
The element is a group of footer lines.

**table-column**
A rule for a `table-column` does not actually render any content. Such a rule can, however, carry properties that can be inherited by the cells in that column.

**table-column-group**
Like `table-column`, a rule with this `display` value does not render content, but can carry properties to be inherited by columns in the group.

**table-cell**
The item is rendered as one cell of a table.

**table-caption**
The item is presented as a table caption.

---

**Note**
A minimal table consists of a `table` or `inline-table` containing at least one `table-row`, which in turn contains at least one `table-cell`.

You don’t have to supply all these parts to create a table. Any missing parts will be added automatically. For example, if you render a series of objects as table cells, an anonymous table row and table will be created to contain them. Similarly, if you render some objects as table rows, but they aren’t inside a table element, an anonymous table will be created to contain the rows.

The various parts of a table render as if they were stacked in this order (from front to back): cells, rows, row groups, columns, column groups, and the table itself. This means, for example, that if the background of a cell is transparent, but its row has a background color, that color will show as the background color of the cell.

### 18.1. Table column properties

Any elements you render with a `display` type of `table-column` or `table-column-group` may have these properties:

**background**
If you set up a background for a column or column group, that background will be visible for transparent rows and cells.

**border**
A column or column group’s `border` property will apply to all cells in the affected columns, but only if the table’s `border-collapse` property has the `collapse` property. For more information on this property, see Section 18.3, “Table border properties” (p. 34).

**width**
The `width` property of a column or column group sets the minimum size of the column. For table sizing in general, see Section 18.2, “How table size is computed” (p. 34).

**visibility**
If you set the `visibility` property of a column or column group to the `collapse` value, that column or group will disappear as if it had never existed.
18.2. How table size is computed

The first property that affects table size and layout is the `caption-side` property. This property can take any of these values:

- **top** is the default caption position. The caption appears above the table.
- **bottom** places the caption below the table.
- **left** or **right** places the caption to the left or right of the table.

The width of a table depends on the `table-layout` property. Select one of these two values:

**fixed**

This layout method starts rendering the table as soon as the first row is complete, so it is a good choice if you have a huge table and you don't want the reader to have to wait for the entire thing to be rendered.

The agent looks first at each column to see if it has a fixed width attribute, and uses that width if it is specified. Otherwise it looks at the first row's cell in that column to see if that cell has a fixed width. Failing that, it distributes the table's width over the cells that don't declare a fixed width.

The final table's width is the value of the table's width property, or the sum of the column widths, whichever is greater.

In the second and succeeding rows, any cells that are wider than the computed column width will not cause the column to get any wider. The `overflow` property of the cell determines what happens to the extra content.

**auto**

Rendering a table with `table-layout` set to **auto** is more flexible, but a table can't be rendered completely until the agent looks at all the rows.

In this method, the agent computes the minimum and maximum width of the contents of each cell in a column. For example, if a cell contains text, the minimum width is the width of the longest line if the content were broken into the maximum number of lines, and the maximum width is how wide that cell would be if it were rendered all on one line.

The final width of the table depends on the table's width property (which is **auto** by default.

- If the table's width is **auto**, and the total of the maximum column sizes is less than the width of the block containing the table, that total is used. Otherwise, the table's width is the sum of the minimum column sizes, or the width of the containing block, whichever is larger.

- If the table's width is a specific value, the rendered width is either that value, or the sum of the minimum column sizes, whichever is larger.

The height of a given row in the table is the height of the tallest cell in the row, or the `height` property of the row, whichever is larger.

The height of a given cell is the height of its content, or the value of its `height` property, whichever is larger.

18.3. Table border properties

The borders that appear in a table depend on the value you set for the table's `border-collapse` property. Select one of:
**collapse**

This is the default. A cell may have border properties set on itself, its row, its row group, its column, its column group, or the table itself.

The actual border rendered at any given spot follows these rules:

1. If any `border-style` property that affects the cell has the value `hidden`, there will be no border.
2. If any of the relevant `border-style` properties are `none` but others are not `none`, the `none` property will be ignored.
3. Wider borders have precedence over narrower borders.
4. If all the relevant border properties have the same width, the border with the higher-priority style wins. From highest to lowest priority, the styles are `double`, `solid`, `dashed`, `dotted`, `ridge`, `outset`, `groove`, and `inset`.
5. If relevant border properties differ only by their color, closer elements win over more distant elements. From closer to further, the elements are cell, row, row group, column, column group, and the table itself.

**separate**

With this option, every cell has its own border. Any border properties on rows, row groups, columns, or column groups are ignored.

Two additional properties control the appearance with the `separate` option:

**border-spacing**

Set this property to one or two dimensions (see Section 6.1, “Dimensions” (p. 5)). If you supply one value, it will be used for both vertical and horizontal spacing between cells. If you supply two values, the first gives the horizontal spacing and the second the vertical spacing.

**empty-cells**

This property controls the appearance of cells that don’t have any content. (You can use the non-breaking space character, `&nbsp;`, to force the appearance of content when there is none.) The default value of this property is `show`, which forces empty cells to have borders. Set this property to `hide` if you want empty cells to be displayed as if there were no cell there. If you select `hide` and all the cells in the row are empty, the entire row will disappear.

### 18.4. The `speak-header` property: Aural rendering of tables

When a table is being rendered by a speech synthesizer, the `speak-header` property controls when table header information (row titles and column titles) is presented. Values may be:

**once**

Speak header information only when it changes. For example, a row header would be spoken only at the beginning of each row. This is the default value.

**always**

Speak header information for each cell.

### 19. User interface options

CSS allows you to control several features of the user interface:

- Section 19.1, “The `cursor` property” (p. 36).
- Section 19.2, “Selecting colors to match UI components” (p. 36).
19.1. The cursor property

You can select what kind of cursor appears when the pointer is over an element by using the `cursor` property. Values may be any of:

- **uri**
  Retrieve the cursor image from a URI.

- **auto**
  Use a cursor appropriate for the type of the element.

- **crosshair**
  A crosshair cursor, something like: ⬤

- **default**
  The default cursor, usually an arrow.

- **pointer**
  The cursor that identifies a link.

- **move**
  A cursor showing something that is being moved, something like: ☹️

- **n-resize | ne-resize | e-resize | se-resize | s-resize | sw-resize | w-resize | nw-resize**
  Produces a cursor for resizing a window in one of the eight compass directions. For example, **s-resize** is for resizing the bottom (south side) of a window, and might look like this: 🖇️

- **text**
  A cursor for selecting text, something like this: 🔧

- **wait**
  The cursor urging the reader to be patient, typically a watch or hourglass image such as: ⏳

19.2. Selecting colors to match UI components

In addition to the colors described in Section 6.2, “Specifying colors” (p. 6), you can use any of the special color names from this list to select a color from the color scheme of the agent’s user interface context.

<table>
<thead>
<tr>
<th>Color Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ActiveBorder</td>
<td>The border around an active window.</td>
</tr>
<tr>
<td>ActiveCaption</td>
<td>The caption of an active window.</td>
</tr>
<tr>
<td>AppWorkspace</td>
<td>In an interface with multiple documents, this is the background color behind those documents’ windows.</td>
</tr>
<tr>
<td>Background</td>
<td>The desktop background.</td>
</tr>
<tr>
<td>ButtonFace</td>
<td>The foreground color for three-dimensional display elements.</td>
</tr>
<tr>
<td>ButtonHighlight</td>
<td>The highlight color on the sides of 3-d elements that are facing the apparent light source.</td>
</tr>
</tbody>
</table>
### 19.3. Dynamic outlines

CSS allows you to display an outline around an element. These outlines are different from borders: they don’t affect the spacing of other elements, and they may not necessarily be rectangular.

There are three outline properties and one combination property:

- **outline-color**
  
  Gives the color of the dynamic outline. In addition to the colors described in Section 6.2, “Specifying colors” (p. 6), you can also specify the value invert, which shows the outline as the inverse of the existing color, so it will show up regardless of the color scheme.

- **outline-width**
  
  Specifies the width of a dynamic outline. The values are the same as those for the border-width property described in Section 15.6, “The border properties” (p. 24).

- **outline-style**
  
  Specifies the style of the dynamic outline. The values are the same as those for the border-style properties described in Section 15.6, “The border properties” (p. 24), except that the hidden value is not allowed.
**outline**

This is a combination property that allows you to specify any of the color, width, and style values at once. For example, this rule sets up a wide dashed red border around the `div` element whose `id` is `n23`:

```css
div#n23 { outline: red dashed wide; }
```

### 20. Aural stylesheets

Use the rules in this section to control the presentation of your material in audio form. This is a good idea to improve accessibility for blind readers, but may be useful in other situations where your readers are actually listeners.

#### 20.1. Spatial presentation: the azimuth property

These properties control the apparent location of the speaker in space. Any stereo rendering will be able to render the spatial position anywhere between the left and right channels. The standard also defines apparent positions behind the listener, as well as positions above and below the plane of the listener's ear, for more advanced rendering systems that can place the sounds in those positions.

The horizontal position property is called `azimuth`. A sound directly in front of the listener, equidistant between left and right channels, is the 0° reference; the right channel is azimuth 90°, and the left channel is azimuth 270° or -90°—these angles are equivalent. A source behind the listener is at azimuth 180°.

The values of the `azimuth` property may be any of:

- The azimuth in degrees. See Section 6.6, “Specifying angles” (p. 7).
- One of the keywords in the diagram below, such as `left-side` for azimuth 270° or `center-left` behind for 200°.
- `leftwards` to place the source 20° clockwise relative to the azimuth inherited from the parent element, that is, the parent's azimuth minus 20°.
- `rightwards` to place the source 20° counterclockwise relative to the azimuth inherited from the parent element, that is, the parent's azimuth plus 20°.
This figure shows the various azimuth keywords and the equivalent angles.

If the rendering agent has speakers above and below the listener, we can also specify the apparent vertical position of the source.

Set the elevation property to one of:

- **angle**
  The angle of the source relative to the horizontal; see Section 6.6, “Specifying angles” (p. 7). For example, the declaration “elevation: -20deg;” would place the source twenty degrees below the horizon.

- **above**
  The source comes from the zenith, straight up.

- **below**
  The source comes from the nadir, straight down.

- **higher**
  The source is placed 10° higher than the elevation property of the parent element.

- **lower**
  The source is placed 10° lower than the elevation property of the parent element.

### 20.2. Voice properties

### 20.3. The volume property

This property controls the loudness of the aural presentation. Permissible values include:

- **integer**
  An integer from 0 to 100 selects the loudness. A value of 0 does not mean no sound at all: it means the softest audible level. A value of 100 is the loudest comfortable level. It is the responsibility of
the user agent, not CSS, to determine what these numbers represent, taking into account the background noise level and the dynamic range of the audio system.

**silent**
Turn the sound off altogether.

**x-soft**
Extra-soft.

**soft**
A relatively soft volume.

**medium**
A medium volume. This is the default value.

**loud**
A moderately loud volume.

**x-loud**
Extra-loud volume.

### 20.4. The `speak`, `speak-punctuation`, and `speak-numeral` properties: spelling it out

The `speak` property has three options:

**normal**
Render the material in the normal, default way.

**none**
Don't speak this element, and don't take any time for it either.

Note that other elements contained inside this element can override this property with a `speak` rule of their own. If you want to suppress rendering of contained elements, set the `display` property to “none”.

**spell-out**
Instead of speaking the words, spell them letter by letter. This may help people understand acronyms that are not easy to pronounce, such as PNAMBC (Pay No Attention to the Man Behind the Curtain).

The `speak-punctuation` property controls how punctuation marks are rendered:

**code**
Pronounce the name of the punctuation mark. For example, you might hear a voice synthesizer say “calm comma calm comma calm period.”

**none**
Render punctuation marks as pauses of appropriate lengths.

Finally, the `speak-numeral` property specifies how numbers are rendered. Values:

**digits**
Speak each digit individually, e.g., 869 would be rendered “eight six nine”.

**continuous**
Render the numeral in the customary way for the language, e.g., “eight hundred sixty-nine.”
20.5. General voice qualities: voice-family, pitch, pitch-range, stress, and richness

The voice-family property selects a general voice type or specific personality. Values may include:

- **female**
  A generic female voice.
- **male**
  A generic male voice.
- **child**
  A generic child’s voice.
- **personality**
  A given voice synthesizer may have any number of specific named voices, somewhat like type font families. Examples: gary owens, talullah bankhead, bullwinkle, lounge singer.

You can provide a comma-separated list of voice names as the value of the voice-family property, and the agent will try them in the given order until it finds one that is actually available. Just as you can't assume any specific font will be available to any reader's browser, it's safest to provide one of the generic voices as a fail-safe at the end of the list. Example:

```
voice-family: madeline kahn, carol kane, female;
```

The pitch property controls whether a voice is low or high. You may specify the value as a frequency (see Section 6.8, “Frequencies” (p. 8)), or any of the values x-low (extra-low), low, medium (the default), high, or x-high for extra-high.

Use the pitch-range property to control how much the synthesized voice varies in pitch. The value of this property is a number from 0 to 100, the default being 50. A declaration “pitch-range: 0” would render the voice as a flat, uninflected monotone. Values above 50 give you higher amounts of pitch range.

The stress property controls how much the voice varies in stress. The value is a number from 0 to 100. A zero value would suppress all stress variations. The default value is 50. Higher values add more variation in stress, as if the speaker were agitated.

Finally, the richness property is also a number from 0 to 100, defaulting to 50. Lower values produce a smoother voice; higher values produce a voice that carries better in a larger room.

20.6. Timing properties: speech-rate, pause-before, pause-after, and pause

The speech-rate property controls how fast the agent speaks the words. Values may be:

- **integer**
  An integer specifies the speech rate in words per minute.
- **x-slow**
  About 80 words per minute.
- **slow**
  About 120 wpm.
- **medium**
  The default speed, somewhere around 180 to 200 wpm.
fast
   About 300 wpm.

x-fast
   About 500 wpm.

defaster
   About 40 wpm faster than the speech-rate of the parent element.

dslower
   About 40 wpm less than the speech-rate of the parent element.

To add a bit of extra silence before an element, set its pause-before property; to add some after it, set pause-after. Values allowed:

time
   The syntax for specifying times is discussed in Section 6.7, “Times” (p. 7).

percentage
   A number followed by “%” specifies the duration as a percentage of an average word length, as specified by the speech-rate property (see Section 20.2, “Voice properties” (p. 39)).

   For example, if the current speech rate is 180 words per minute, the average time per word is 1/3 of a second, so a pause-before value of “200%” would add a gap of about 2/3 of a second.

There is also a combination pause property that sets both the pause-before and pause-after properties. If pause is followed by one value, both properties are set to that value; if there are two values, pause-before gets the first value and pause-after the second.

For example, this rule inserts a 150-millisecond pause both before and after any h2 element:

   h2 { pause: 150ms; }

20.7. Element cues: cue-before, cue-after, and cue

You can play a sound clip before or after a selected element. The cue-before property plays the clip before the element, and cue-after plays the clip after the element. Values may be:

tui
   To supply a sound file, specify the file’s URI as the value.

none
   Do not use a cue.

You can set both cues at once with the cue combination property. If you supply one value, that clip is played both before and after the element. You may also supply two values, one before and after. For example:

   p.warning { cue: url("reveille.wav") url("retreat.wav"); }

This rule would play reveille.wav before each <p class='warning'>…</p> element, and play retreat.wav after it.

20.8. Audio mixing: play-during

The play-during property causes a recording to be played in the background during the rendering of some element. Values may be:
uri
Retrieve the recording from the given URI. The value may be followed by either of two keywords:

mix
If this keyword is given, and the parent element has also specified a recording with the play-during property, mix the parent's recording with this element's recording. Without this keyboard, the child element's recording would replace that of the parent.

repeat
If given, this keyword tells the agent to repeat the recording as many times as necessary to be heard behind the rendering of the element. The default is to play it only once. If the recording is longer than the rendering of the element, it is cut off when the element is finished rendering.

auto
If the parent element has a play-during recording, continue playing it back as this element is rendered.

none
Do not play a recording during this element.

inherit
If the parent element has a play-during recording, start it over again for the current element.

21. The @import rule: Importing another stylesheet
You can use rules from a different stylesheet in your stylesheet by using the @import at-rule. The general form is either of these:

```css
@import url(string);
@import string;
```

The string is the URI of the stylesheet you want to import.

Warning
If you use an @import, it must precede all other rules in the same stylesheet.

The effect of importing another stylesheet is just as if the other stylesheet were copied into yours at that location—with one important difference. Imported rules have a lower priority than your other rules.

For example, suppose your stylesheet contains this at-rule:

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

Further suppose that your stylesheet has a rule for paragraphs, with selector p, and the basic.css stylesheet also has a rule for paragraphs, with the same selector. In that case, your rule will be used.

You may also specify a comma-separated list of media types after the URL part of the rule. In this case, the importation of the other stylesheet would happen only if the content were being rendered in one of the selected media types.

For example, this rule:

```css
@import "speech.css" aural, tv;
```

would apply only for content rendered by a speech synthesizer (aural) or television (tv).

For more information, see Section 22.1, “Media types” (p. 44).
22. The @media rule: Tuning for different rendering platforms

If you want certain rules to be applied only for certain media types, wrap them in a @media at-rule with this format:

```css
@media type[, ...] { rule ... }
```

where the @media keyword is followed by a comma-separated list of media types, and the rules inside the curly braces are used only when the content is rendered on one of those media types. See Section 22.1, “Media types” (p. 44).

For example, these lines would specify 14-point fonts on a screen, 12-point fonts on paper, and brown type on a yellow background in both cases:

```css
@media screen
{ body
  { font-size: 14pt;
    }
}
@media print
{ body
  { font-size: 12pt;
    }
}
@media print, screen
{ body
  { color: brown;
    background-color: yellow;
    }
}
```

22.1. Media types

Because Web content can be rendered in so many different ways, CSS allows you to define rules that apply only to certain types of media.

- CSS defines a set of media types that describe different ways to render content. Examples: screen for a color display, print for printed pages.
- CSS also defines several media groups that describe different qualities of each media type. For example, different media may be described as paged or continuous depending on whether they are presented on fixed-size pages, or continuously such as in a Web browser with scrollbars.

Here are the media groups. The keyword “both” is allowed in some groups to mean that either choice applies.

**Paged**

Is this medium made of fixed-size pages, or is it continuously scrollable? Values are:
- paged for fixed-size pages.
- continuous for Web browsers and other devices with scrolling capability.

**Senses**

What senses do users need to perceive this rendering? Values are:
- visual for sight.
- aural for hearing.
- tactile for touch.
Grid
Can this device render individual pixels, or is it restricted to a grid of characters? Example: ancient CRT displays that show only 24 rows of 80 characters. Values are:

- **grid** for devices restricted to a character grid.
- **bitmap** for devices that can render individual pixels.

Interactive
Can this medium change in response to user actions? A screen can, but paper can’t. Values are:

- **interactive**
- **static** (non-interactive)

Here is a table showing the current set of media types and the attributes of the various media groups.

<table>
<thead>
<tr>
<th>Type</th>
<th>Paged</th>
<th>Senses</th>
<th>Grid</th>
<th>Interactive</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>aural</td>
<td>continuous</td>
<td>aural</td>
<td>–</td>
<td>interactive</td>
<td>Speech synthesizer.</td>
</tr>
<tr>
<td>braille</td>
<td>continuous</td>
<td>tactile</td>
<td>grid</td>
<td>interactive</td>
<td>A Braille device with tactile feedback.</td>
</tr>
<tr>
<td>embossed</td>
<td>paged</td>
<td>tactile</td>
<td>grid</td>
<td>interactive</td>
<td>A paged Braille printer.</td>
</tr>
<tr>
<td>handheld</td>
<td>both</td>
<td>visual</td>
<td>both</td>
<td>interactive</td>
<td>Handheld devices with small screens, limited resolution, and low bandwidth, such as palmtops.</td>
</tr>
<tr>
<td>print</td>
<td>paged</td>
<td>visual</td>
<td>bitmap</td>
<td>static</td>
<td>Printed documents.</td>
</tr>
<tr>
<td>projection</td>
<td>paged</td>
<td>visual</td>
<td>bitmap</td>
<td>static</td>
<td>A set of overhead projector transparencies.</td>
</tr>
<tr>
<td>screen</td>
<td>continuous</td>
<td>visual</td>
<td>bitmap</td>
<td>both</td>
<td>A color display screen.</td>
</tr>
<tr>
<td>tty</td>
<td>continuous</td>
<td>visual</td>
<td>grid</td>
<td>both</td>
<td>A “dumb terminal.”</td>
</tr>
<tr>
<td>tv</td>
<td>both</td>
<td>visual/aural</td>
<td>bitmap</td>
<td>both</td>
<td>A television with audio.</td>
</tr>
</tbody>
</table>

23. The @page rule: Paged media

You can specify rules that apply only when the content is presented on fixed-size pages such as PDF output. In general, the rule has this format:

```css
@page { ... }
```

The rules inside the braces are applied to paged media.

It is important to distinguish between pages and sheets. For example, “two-up” printing places two page boxes on a physical sheet.

You can also create named page types with a rule of this format:

```css
@page name { ... }
```

See Section 23.6, “The page attribute: Selecting a page type” (p. 48) for more information on named pages.

If you want some rules to apply only to odd pages, even pages, or the first page of the set, use this option to the @page at-rule:

```css
@page page-selector { ... }
```
where the page-selector is one of:

- :right for right-hand pages.
- :left for left-hand pages.
- :first for the first page.

You can use margin properties inside the @page rule to define the margins of the page. For example, this rule would set up a page with one-inch margins all around:

```css
@page { margin: 1in; }
```

You may also use the individual margin properties such as margin-left to set different values for different margins. See Section 15.7, “The margin properties” (p. 25).

The declarations described below may appear only inside an @page rule. They apply only to presentation of the content on fixed-size pages:

- Section 23.1, “The size property for paged media” (p. 46).
- Section 23.2, “Controlling page breaks” (p. 46).
- Section 23.3, “Orphan control” (p. 47).
- Section 23.4, “Widow control” (p. 47).
- Section 23.5, “Crop marks and alignment targets: the marks property” (p. 47).
- Section 23.6, “The page attribute: Selecting a page type” (p. 48).

### 23.1. The size property for paged media

This property allows you to specify the size of a page box. There are several forms:

```css
size: width height;
```

Specifies the absolute size of the page box as two dimensions (see Section 6.1, “Dimensions” (p. 5)).

For example, to specify a sheet 20cm wide by 33cm high:

```css
@page { size: 20cm 33cm; }
```

- `size: auto;` Sets the page size to the sheet size.
- `size: portrait;` Selects portrait orientation, with the longer sides vertical.
- `size: landscape;` Selects landscape orientation, with the longer sides horizontal.

### 23.2. Controlling page breaks

Three properties, allowed only inside @page at-rules, let you control where page breaks fall.

```css
page-break-before
```

Specifies when page breaks can occur before some element. Values allowed:

- `always`: Always start a new page before this element.
- `avoid`: Avoid page breaks before this element.
- `right`: Insure that this element starts on a new, right-hand page.
- `left`: Insure that this element starts on a new, left-hand page.
**page-break-after**
Specifies whether page breaks are allowed after some element. Values are the same as for `page-break-before`: `always` means always start a new page after the element; `avoid` prevents a page break after the element; `right` forces the following element to start on a new right-hand page; and `left` forces the following element to a new left-hand page.

**page-break-inside**
To prevent page breaks inside a given element, use this declaration:

```css
page-break-inside: avoid;
```

### 23.3. Orphan control

An *orphan* is one or more lines by themselves at the bottom of the page. It is often considered bad typesetting practice to have single-line orphans, and some shops don’t even like two-line orphans.

To suppress orphans with *n* lines, use this declaration:

```css
orphans: n;
```

For example, if you want to make sure that a `p` (paragraph) element has no orphans of three lines or fewer:

```css
@page
{ p { orphans: 3; }
}
```

### 23.4. Widow control

A *widow* is the opposite of an orphan (see Section 23.3, “Orphan control” (p. 47)): one or more lines by themselves at the top of a page.

To suppress widows of *n* lines or fewer, use this declaration:

```css
widows: n;
```

For example, to suppress single-line widows inside an `li` element:

```css
@page
{ li { widows: 1; }
}
```

### 23.5. Crop marks and alignment targets: the `marks` property

You can specify two additional decorations with the `marks` property.

**marks: crop;**
Adds *crop marks* to the printed output. These marks show a print shop where the physical edge of the paper will appear.

**marks: cross;**
Adds alignment targets to the printed output. These marks appear on every sheet in the same position, so a print shop can line up sheet images.

You can get both these decorations with this rule:
23.6. The page attribute: Selecting a page type

The page attribute, allowed only inside an @page at-rule, allows you to specify that some element will appear only on a certain page type.

The general form:

```
page: page-name
```

where the page-name is specified just after the “@page” in the at-rule.

For example, suppose you want all table elements to be displayed on a page with landscape orientation. First you define a new page-name that specifies a landscape page:

```
@page land { size: landscape; }
```

Then you connect the table element with that page name using this rule:

```
table { page: land; }
```