OXFORD BROOKES UNIVERSITY

CSS 2.1 Primer

Contents

Appendices

1. Introduction

1.1 Content and Style

Two aspects of any document are content and style. The content gives the **information** to be presented and the **style** defines how the information is presented. Most publishers have a House Style that is a consistent way of presenting information. The same information printed by two publishers may look quite different. This Primer describes the basic features of Cascading Style Sheets (CSS for short) which is the primary styling language used with HTML.

HTML's main role is to define content. What is needed is the ability for publishers to have control of style. In the context of the Web, the publisher can be the organisation that owns the Web site but it can also be the person viewing the information. It is only by having a clear separation between content and style that this can be achieved. So in this Primer we shall use HTML in most of the examples as a means of defining the **content** and CSS to define the **style**.

CSS is a World Wide Web Consortium (W3C) Recommendation. That means that all the W3C Members (which includes all the main browser manufacturers) have agreed to support it in their products. That does not necessarily mean immediately as each has its own production schedule for enhancements to its products but long term all should support all of the features. CSS first became a W3C Recommendation in 1996 (called CSS 1) and there was a significant update in May 1998 (called CSS 2). Not all of the facilities in CSS 2 were implemented by all manufacturers. A new version is under development called CSS 2.1 that corrects some errors found in CSS 2 and either removes or makes optional those features not supported. It also adds some features that people required (particularly for XML) that were not in CSS 2.

With the ever increasing use of the Web by a large variety of devices, there is a need to profile facilities in specifications like CSS so that a device with limited capabilities can support a consistent subset. Towards this goal, CSS 3 will be divided into a set of modules with profiles aimed at devices like mobile phones, PDA, televisions etc. This work was goes under the general heading of CSS 3. Some modules are near completion while others still have significant work to be done on them. It is still under development and will not be considered further in this Primer. This Primer aims to describe the main facilities available in CSS 2.1

HTML allows a Web page to be laid out and there is some guidance to browsers as to how to represent each element. Thus, **h1** should be larger than **h2** and **em** should be emphasised by some method but that is as far as it goes.

The aim of Cascading Style Sheets (CSS) is to give the page developer much more control on how a page should be displayed by **all** browsers.

A **style sheet** is a set of **rules** that controls the formatting of HTML elements on one or more Web pages. Thus, the appearance of a Web page can be changed by changing the style sheet associated with it. There is no need to make detailed changes within the Web page to change how it looks.

Some of the advantages of using style sheets are **accessibility**, different styling can be provided for different users dependent on their requirements. Separating style and content is **good design** and will normally produce a better and more consistent web site. As one style sheet can be used for a whole web site, it normally means that the overall **size** of the web site is smaller and the downloads required for each page can be decreased by up to 40%. Allwoing browsers to make overall decisions on styling often means that the rendering time by the browser is also shorter. We shall see how styling can radically effect a page's layout and this allows important information to appear early in the HTML markup of the page even if the design requires it to appear later. This can be of use to search engines.

A set of Web pages may use a common style sheet. A Web page may have its own style sheet that refines the information in the common style sheet. Readers may define their own style sheet indicating their preferences. Thus style sheets **cascade** and decisions need to be made as to which style sheet is in control when there is a conflict. But more of that later.

A style sheet **rule** consists of two parts. A **selector** that defines which HTML elements are controlled by the rule and a **declaration** that says what the required effect is. Thus a simple rule is:

```
h1 { color: blue }
```

This says that all **h1** elements in a page should be displayed in blue. The selector is **h1**. **color** is the **property** that is to be changed, **blue** is the **value** that the property is changed to and **color: blue** is the declaration.

1.2 HTML Structure

HTML elements in the body of the document fall into two main classes:

1. Block-level
2. Inline (sometimes called character-level)

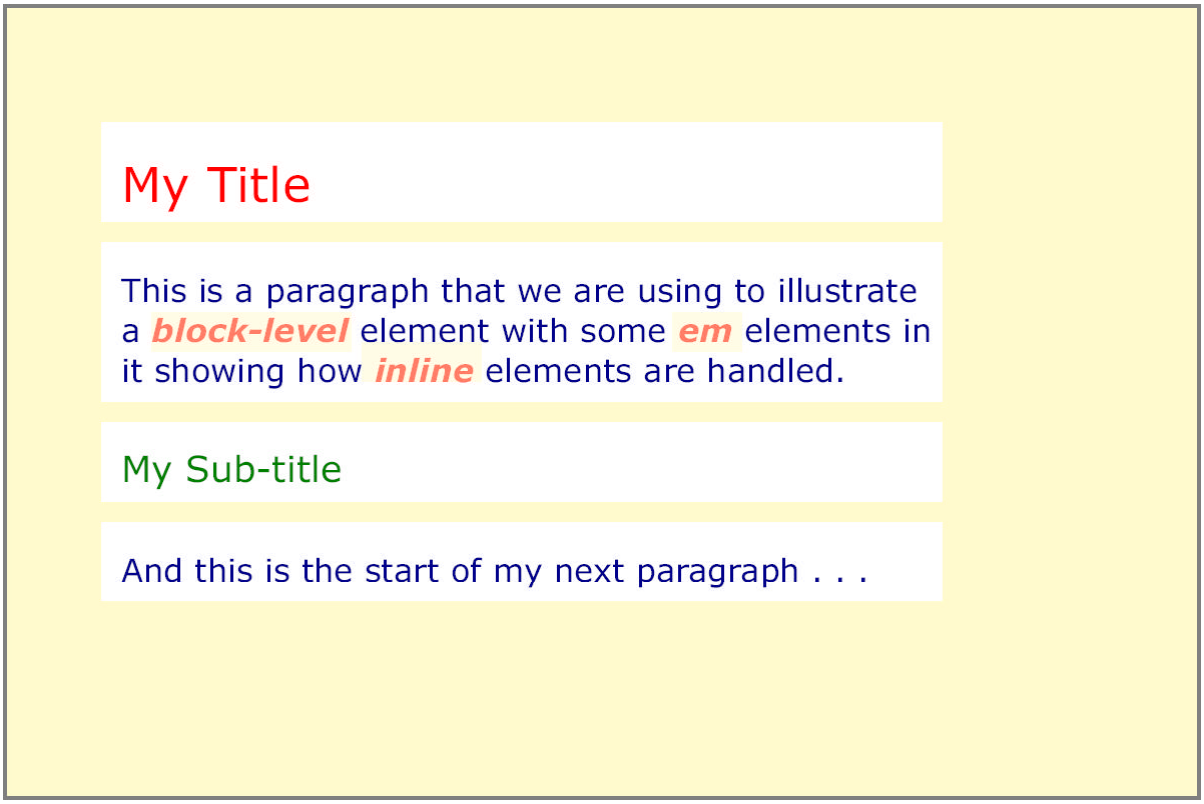

Figure 1.1: Block level and Inline Elements

Lists and tables are special types of elements that have their own unique styling.

When a block-level element is inserted into a document, it terminates the previous element and effectively starts a new line. Some examples are **<p>** and **<h1>**. Inline elements do not terminate the previous element and form part of the previous element; **em** is an example.

The **body** of an HTML page contains text marked-up using block-level elements (**h1, p, ul, div** etc). Each block-level element has an area which contains the content. For a **p** element, this is the area that contains the whole paragraph while for **h1** it is often an area consisting of a single line of text. CSS provides facilities for controlling where that area is placed relative to other areas.

The inline elements often inherit the style of the block-level element that they are part of but can be given special styling appropriate to the inline element. For example:

```
<h1>My Title</h1>
<p>This is a paragraph that I am using to illustrate
a <em>block-level</em> element
with some <em>em</em> elements in it showing
how <em>inline</em> elements are handled.</p>
<h2>My Sub-title</h2>
<p>And this is the start of my next paragraph . . .</p>
```

Figure 1.1 shows the areas covered by the block level elements and the sub-areas covered by the inline elements. The inline elements have been given their own background colour, colour, font-weight (bold) and font style (italic) while inheriting the text size and font from the enclosing paragraph.

1.3 CSS Attachment to Pages

There are four ways of linking a style sheet to an HTML page:

1. By defining a link from the HTML page to the style sheet (normally stored in a **xxx.css** file). This allows you to have a single style sheet that changes the appearance of many Web pages.
2. By specifying an HTML **style** element in the **head** of the page. This allows you to define the appearance of a single page.
3. By adding inline styles to specific elements in the HTML file by the **style** attribute. This allows you to change a single element or set of elements. This should only be used when absolutely necessary as it negates some of the advantages of having style sheets. It effectively over-rides the overall style for the page. Also, it is likely that it will be removed from CSS 3.
4. By importing a style sheet stored externally into the current style. The **style** element for the page can consist of a set of imports plus some rules specific to the page.

Initially, while you are experimenting, the simplest method is to use the HTML **style** element to add a style sheet to an HTML page. The **style** element, consisting of a set of rules, is placed in the document **head**.

When you have decided on your house style, it is likely that you will move to having the style sheet external to an individual page and either linked in or imported.

1.4 Main Facilities

The facilities provided by style sheets are much as you would expect:

1. Better control of fonts including colour
2. Better control of block-level layout (indents, margins, alignment etc)
3. Better control of inline layout, particularly with regard to diagrams and related text

1.5 Browser Support

To achieve these effects, the Web browser has to know what to do with style sheets. Really old browsers will not have this capability but the recent offerings from Microsoft, Opera and Netscape have good support now and are committed to full support in the future. HTML Editors are beginning to have Style Sheet additions and separate Style Sheet Editors are appearing. Visit the W3C CSS Web Page [1] for all the latest information.

Given the way CSS has been implemented, it is possible to design your web pages so that they still present the HTML information even when CSS support is not there. So there is little excuse for not adding style to your web pages now even if you expect your pages are to be viewed by really old browsers.

1.6 CSS Specification

This Primer will not tell you everything about CSS. If you get to the position where you really need to know precisely what happens in some subtle situation, you need to consult the formal specification that can be found on the World-Wide Web Consortium's web site [2] [3].

To make it as easy as possible to relate the specification to the Primer, the order of presentation here is similar to the order in the Specification.

2. CSS Rules

2.1 Introduction

Let us assume initially that we wish to define a style sheet by adding a **style** element to a page. The page will look something like:

```
<html>
<style type="text/css">
h1 {font: 12pt/14pt "Palatino"; font-weight: bold; color: red}
h2 {font: 10pt/12pt "Palatino"; font-weight: bold; color: blue}
p  {font: 10pt/12pt "Times"; color: black}
</style>
<body>
...
</body>
</html>
```

The set of rules are contained between **<style>** and **</style>**. Ignore the **type** attribute, for now. The reason for this will be come clearer later. For now let us concentrate on the rules. As stated in the introduction, each rule consists of two parts: a **selector** that defines what HTML elements are controlled by the rule and a **declaration** that says what the required effect is. The selectors above define three rules that apply to all **h1** elements, all **h2** elements and all **p** elements respectively. We shall show how we can be both more selective (control a subset of **h1** elements) or control a set of elements (a block of different HTML elements) later. Initially let us look at the declarations. Each declaration consists of the name of a property and the value to be assigned to it. So the simplest style rule is:

```
selector  { property: value }
```

To set several properties in a single rule requires the individual declarations to be separated by semicolons:

```
selector  { property1: value1; property2: value2; property3: value3 }
```

2.2 Syntax for CSS Properties

Sections 4 to 8 define the CSS properties that can be set. The general format for each sub-section is for the title to give a descriptive name followed immediately by the syntax definition (in BNF style) for the property declaration. For example:

font-weight: <value>

This says that the name of the property is **font-weight** and **<value>** is assigned to it. In the CSS specification, the possible values of **<value>** are defined. In this Primer, sometimes only a sample of the possible values are given to give an idea of what is possible rather than exhaustively list them all.

In BNF, the notation could be written:

```
<font-weight> ::= font-weight: <value>
```

This style is used if, for example, the definition of <value> needs to be given separately.

Some of the syntax definitions can be quite complicated. The notation used is as follows:

Notation	Meaning
<Abc>	A value of type Abc. Some of the more common types are discussed later.
abc	A keyword that must appear exactly as written.
X Y Z	X must occur then Y then Z in that order.
X | Y	Xor Y must occur.
X || Y || Z	X, Y or Z or some combination in any order.
[Abc]	The square brackets are used to group items together.
ST*	ST is repeated zero or more times. ST can be a bracketed set of items.
ST+	ST is repeated one or more times.
ST?	ST is optional. It can either appear once or not at all.
ST {A,B}	ST must occur at least A times and at most B times.

3. Length, Percentage, Color and URLs

3.1 Introduction

Four values (<length>, <percentage> , <color>, <url>) are used by many of the rules. They are explained in this section.

3.2 Length Values

<length> ::= [+ | -]? <number> <unit>

A length value is formed by an optional + or -, followed by a number, followed by a two-letter abbreviation that indicates the unit (this can be left out if the value is 0). There are no spaces in the middle of a length value.

Both relative and absolute length units are supported in CSS. Relative units give a length relative to another length property. The relative length units are:

Unit	Meaning
em	Historically width of letter **M**, but in CSS it equals the font height (if font is 12pt then 1em=12pt)
ex	x-height: the height of the letter **x** which varies between fonts (a 12pt Times Roman 'ex' will be bigger than a 12pt Baskerville 'ex')
px	pixels: in CSS the pixel is equal to one pixel on a normal computer display. So if the output is to go to a 1200dpi laser printer, the browser will take 1 pixel to mean about 12 laserprinter pixels.

CSS recommends that the pixel unit **px** should be regard as the smallest unit of resolution. So if you were reading information on the display at arms length distance away, it should be approximately 0.28mm or 1/90th of an inch. Consequently, a measurement of 12pt and 15px should be about the same.

The absolute length units are:

Notation	Meaning
in	inches
cm	centimetres
mm	millimetres
pt	Printing industry point; originally there was 72pt to a French inch which was larger than an English inch! Postscript and CSS define 1pt=1/72in
pc	picas; 1pc=12pt

Lengths such as **pt** and **in** are converted to pixels using the user agent's guess. So which units should you use in defining a Style Sheet? There is no real correct answer as it depends on what is the main output device and whether you would like to change the whole Style sheet by just applying a single change. In the Lie and Bos book [4], a strong recommendation is made to use **ems**. If the user has set the browser up to use 10pt type for displaying Web pages, then setting the **h1** font size to **2em** will ensure that the font-size is 20pt in size. If another user sets the browser normal font size to be 12pt, the **h1** element will have a 24pt font size.

The x-height of a font is critical to legibility. A 12pt font with a small x-height will be less readable than one with a larger x-height. Inches and millimetres depend on the browser knowing the size of the display being used. If I have a portable with a small display and attached to it is a much larger screen, what is an inch. Well the browser has to make a decision with some help from the user.

In summary, absolute units are not really absolute and relative units are not always relative. The only real recommendation that can be made is be careful if you plan to mix units and especially if you mix absolute and relative ones.

3.3 Percentage Values

<percentage> ::= [+ | -]? <number> %

A percentage value is formed by an optional + or -, followed by a number, followed by %. There are no spaces in a percentage value.

Percentage values are relative to other values, as defined for each property. Often the percentage value is relative to the element's font size but it might be the width of the page.

3.4 Color Values

<color> ::= <keyword> | # <hex> <hex> <hex> | # <hex> <hex> <hex> <hex> <hex> <hex> |
rgb (<int> <int> <int>) | rgb (<percentage. <percentage > <percentage>)

A color value is a keyword or a numerical RGB specification. The original set of keyword colours was aqua, black, blue, fuchsia, gray, green, lime, maroon, navy, olive, purple, red, silver, teal, white and yellow. These have been enhanced as a result of the requirements from other specifications. The set of keywords with the corresponding RGB values given in Appendix C are likely to be supported by a browser.

RGB colours can be defined in one of four ways:

1. #rrggbb(e.g., #00ff00)
2. #rgb (e.g., #0f0)
3. rgb(x,x,x) where x is an integer between 0 and 255
4. rgb(y%,y%,y%) where y is a number between 0.0 and 100.0

The following examples all specify the same colour:

```
pre { color: red }
h1 { color: #foo }
p  { color: #ff0000 }
h2 { color: rgb(255,0,0) }
h3 { color: rgb(100%, 0%, 0%) }
```

Values outside the range are clipped. So, for example, integer values have a maximum of 255 and everything above that will be changed to 255. Similarly 120% is interpreted as 100%.

3.5 URLs

<url> ::= url(<whitespace>? [' | "]? <characters in url> [' | "]? <whitespace>)

A URL value is of the form:

url(xyz)

where xyz is the URL. The URL may be quoted with either single (') or double (") quotes and may have whitespace before the initial quote or after the final quote.

Parentheses, commas, spaces, single quotes, or double quotes in the URL must be escaped with a backslash. Partial URLs are interpreted relative to the style sheet source, not to the HTML source. Some examples are:

```
body { background: url("circle.gif") }
body { background: url("http://www.clrc.images/monkey.gif") }
```

4. Font Properties

4.1 Font Family (set typeface)

font-family: [<family-name> | <generic-family>] [, [<family-name> | <generic-family>]]*
<family-name> ::= serif | sans-serif | cursive | fantasy | monospace

The font to use is set by the font-family property. Instead of defining a single font, a sequence of fonts should be listed. The browser will use the first if it is available but try the second and so on. The value can either be a specific font name or a generic font family.

Good practice is to define one or two specific fonts followed by the generic family name in case the first choices are not present. Thus a sample font-family declaration might look like this:

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

Any font name containing spaces (multiple words) must be quoted, with either single or double quotes. That is why the first font name is quoted but Times is not.

The first two requests are for specific fonts **New Century Schoolbook** and **Times**. As both are serif fonts, the generic font family listed as a backup is the **serif** font family. In this case, Times will be used if New Century Schoolbook is not available, and a serif font if Times is not available.

Some example fonts are:

serif
Times New Roman, Bodini, Garamond
sans-serif
Trebuchet, Arial, Verdana, Futura, Gill Sans, Helvetica
cursive
Poetica, Zapf-Chancery, Roundhand, Script
fantasy
Critter, Cottonwood
monospace
Courier, Courier New, Prestige, Everson Mono

See also the **font** property which lets the user define a set of font properties in a single command..

4.2 Font Style (italicise text)

font-style: normal | italic | oblique

The font-style property has one of three values: **normal**, **italic** or **oblique** (slanted). A sample style sheet might be:

```
h3 { font-style: italic }
p  { font-style: normal }
```

Italic and oblique are similar. ***Italic*** is normally a separate font while ***oblique*** often bends the normal font.

4.3 Font Variant (size of upper case)

font-variant: normal | small-caps

Normally, upper case characters are much larger than the lower-case characters. Small-caps are often used when all the letters of the word are in capitals. In this case the upper case characters are only slightly larger than the lowercase ones. An example is:

```
p { font-variant: small-caps }
```

For example, the text **Elephant** with font-variant set to **small-caps** would appear as ELEPHANT.

4.4 Font Weight (set thickness of type)

font-weight: normal | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900

The font-weight property sets the weight of the font to values like **normal** and **bold**. It also has values **bolder** and **lighter** which are relative to any inherited font -weight. Block-level elements are displayed with default values inherited from the **body** or **div** element that surrounds it. It is also possible to define absolute font-weights. Values range from 100 to 900 with **normal** being 400 and bold being **700** approximately. If that much control is not possible, some of the weights may be grouped together (for example, 100, 200 and 300 may all look the same) and if the specified weight is not available, an alternative value will be chosen. Some examples are:

```
h1 { font-weight: 800 }
h2 { font-weight: bold }
h3 { font-weight: 500 }
p  { font-weight: normal }
```

4.5 Font Size (set size of text)

font-size: <absolute-size> | <relative-size> | <length> | <percentage>

<absolute-size> ::= xx-small | x-small | small | medium | large | x-large | xx-large
<relative-size> ::= larger | smaller

The font-size property sets the size of the displayed characters. The absolute-size values are **small**, **medium** and **large** with additional possibilities like **x-small** (extra small) and **xx-small**. Relative sizes are **smaller** and **larger**. They are relative to any inherited value for the property. The length value defines the precise height in units like **12pt** or **1in**. Percentage values are relative to the inherited value. Some examples are:

```
h1 { font-size: large }
h2 { font-size: 12pt }
h3 { font-size: 5cm }
p  { font-size: 10pt }
li { font-size: 150% }
em { font-size: larger }
```

The guidance is that the medium font should be around 10pt so, in this case **li** and **em** might be similar in size.

4.6 Font (Shorthand)

font: [<font-style> || <font-variant> ||
<font-weight>]? || <font-size> [/<line-height>]? || <font-family>]

The font property may be used as a shorthand for the various font properties, as well as the line-height. For example:

```
p { font: italic bold 12pt/14pt Times, serif }
```

specifies paragraphs with a bold and italic Times or serif font with a size of 12 points and a line height of 14 points. Note: The order of attributes is significant: the font weight and style must be specified before the others.

5. Color and Background Properties

5.1 Color

color: <color>

The color property sets the foreground colour of a text element. See Section 3.4 for color values. For example:

```
h1 {color: blue }
h2 {color: rgb(255,0,0) }
h3 {color: #0F0 }
```

5.2 Background Color

background-color: <color> | transparent

The background-color property sets the background colour of an element. For example:

```
body { background-color: white }
h1   { background-color: #000080 }
```

The value **transparent** indicates that whatever is behind the element can be seen. For example if the background to the **body** element was specified as being red, a block-level element with **background-color** set to **transparent** would appear with a red background.

5.3 Background Image

background-image: <url> | none

The background-image property sets the background image of an element. For example:

```
body { background-image: url("/images/monkey.gif") }
p    { background-image: url("http://www.cclrc.com/pretty.png") }
h1   { background-image: none }
```

When a background image is defined, a compatible background colour should also be defined for those users who do not load the image or to cover the case when it is unavailable. If parts of the image are transparent, the background colour will fill these parts. For example:

```
<ul style="background-image:url('wall.png')">
<li>An item</li>
<li>Another</li>
<li>And third</li>
</ul>
```

Figure 5.1 shows what might be the result depending on the font-size and other properties specified for the list.

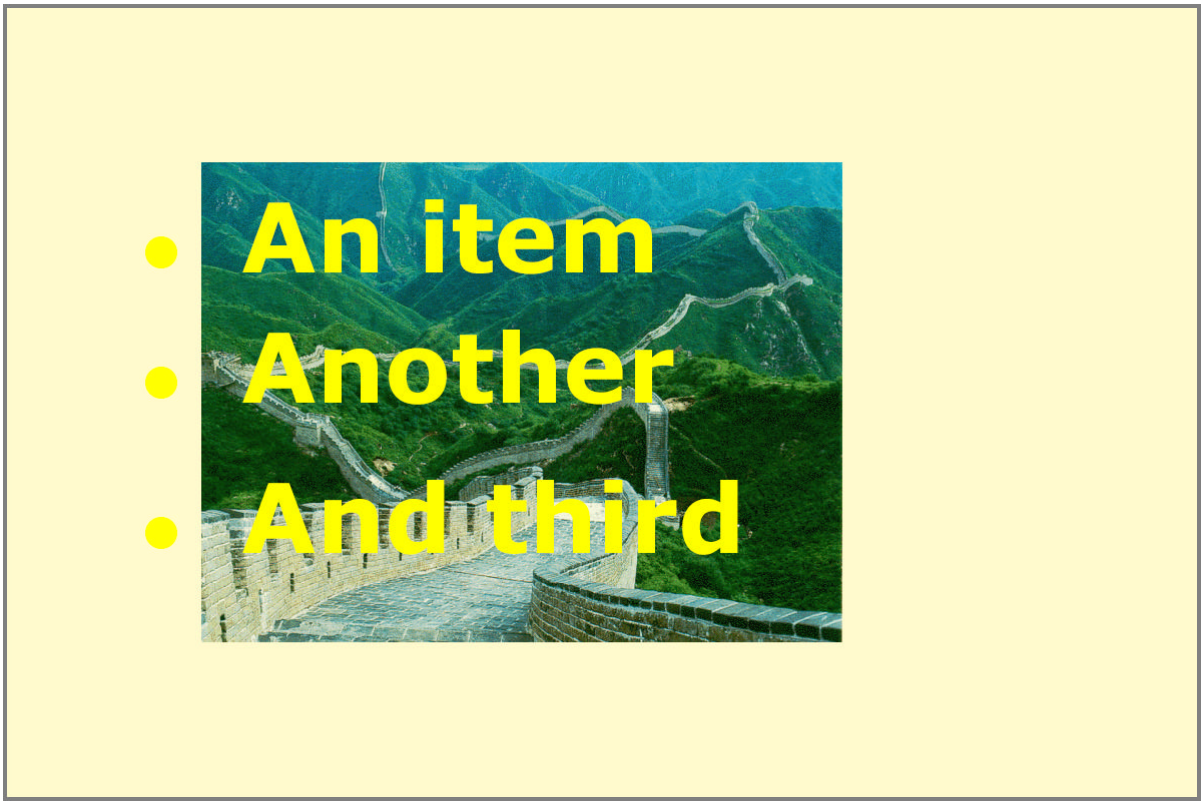

Figure 5.1: List with background-image

5.4 Background Repeat

background-repeat: repeat | repeat-x | repeat-y | no-repeat

If the background image does not fill the whole element area, this description specifies how it is repeated. The **repeat-x** value will repeat the image horizontally while the **repeat-y** value will repeat the image vertically. Setting **background-image** to **repeat** will repeat the image in both x and y directions. For example:

```
body { background-image: url(/images/monkey.gif) }
body { background-repeat: repeat-x }
```

In the above example, the monkey will only be tiled horizontallya as shown in Figure 5.2.

Figure 5.2: Monkey tiled horizontally

5.5 Background Attachment

background-attachment: scroll | fixed

The background-image may be pasted on to the screen of the display in which case, when the elements that it is associated with move, they will be on top of a different part of the image. The alternative is that the background image is attached to the element, in which case, when the element moves, the background stays the same. The background-attachment value of scroll indicates that the background moves with the content, while fixed indicates it is fixed. For example, the following specifies a fixed background image:

```
body { background-image: url("monkey.jpg") }
body { background-attachment: fixed }
```

This tends to be mainly useful if the background image is attached to the **body** element. If the page is very long and the user has to scroll down the page to see the bottom, the background will remain fixed if **background-attachment** is set to **fixed**. For most other uses, the default value of **scroll** makes more sense.

5.6 Background Position

background-position: [percentage> | length]{1,2} |
[[top | center | bottom] || [left | center | right]]

The effect of the background-image (especially when it is repeated) will depend significantly on the origin of the image relative to the display (in **fixed** attachment) and relative to the element (in **scroll** attachment).

The background-position property gives the initial position of the background image relative to the display or element depending on the attachment mode.

The easiest way to assign a background position is with the keywords:

- Horizontal: left, center, right
- Vertical : top, center, bottom

The background position defines a point on the image that coincides with a position on the area it is to be mapped onto. So **top left** says that the top left of the image coincides with the top left of the area and so on.

Percentages and lengths may also be used to assign the position of the area and background image.

When using percentages or lengths, the horizontal position is specified first, followed by the vertical position. Thus 20% 50% specifies that the point 20% across and 50% down the image is placed at the point 20% across and 50% down the area.

If only the horizontal value is given, the vertical position is assumed to be 50%.

The keywords are interpreted as follows:

Notation	Notation	Notation	Meaning
top left	left top		0% 0%
top	top center	center top	50% 0%
right top	top right		100% 0%
left	left center	center left	0% 50%
center	center center		50% 50%
right	right center	center right	100% 50%
bottom left	left bottom		0% 100%
bottom	bottom center	center bottom	50% 100%
bottom right	right bottom		100% 100%

5.7 Background (set background images or color)

background: [<background-color> || <background-image> || <background-repeat> || <background-attachment> || <background-position>]

The background property is a shorthand for the background-related properties. Some examples are:

```
body { background: white url(http://www.clrc/xyz.gif) }
p    { background: url(../backgrounds/lion.png) #F0F000 fixed }
h1   { background: #0F0 url(grass.jpg) no-repeat bottom left }
ul { background: white url('monkey.jpg') repeat scroll 0% 50%}
```

A value not specified will receive its default value (see Appendix B). For example, the first two rules above will have the background-position set to top left. The result of the fourth example is shown in Figure 5.3. As the image covers the complete area, the white background will only be seen if the user chooses not to download the image.

Figure 5.3: Monkey tiled and position center vertically

6. Text Properties

6.1 Word Spacing

word-spacing: normal | <length>

The word-spacing property defines additional spacing between words. Negative values are permitted which allows the letters to be closed up or even overlap. Word spacing may be influenced by alignment. See Section 3.2 for a definition of <length>. For example:

```
p  { word-spacing: 0.4em }
h1 { word-spacing: -0.2em }
h2 { word-spacing: normal }
```

6.2 Letter Spacing

letter-spacing: normal | <length>

The letter-spacing property defines additional spacing between characters. Negative values are permitted. A value of normal will still allow justification to take place. A setting of 0 is defined as preventing justification. For example:

```
h {letter-spacing: 0.2em }
p {letter-spacing: -0.1cm }
```

Figure 6.1 gives examples of the effect of additional letter spacing and word spacing.

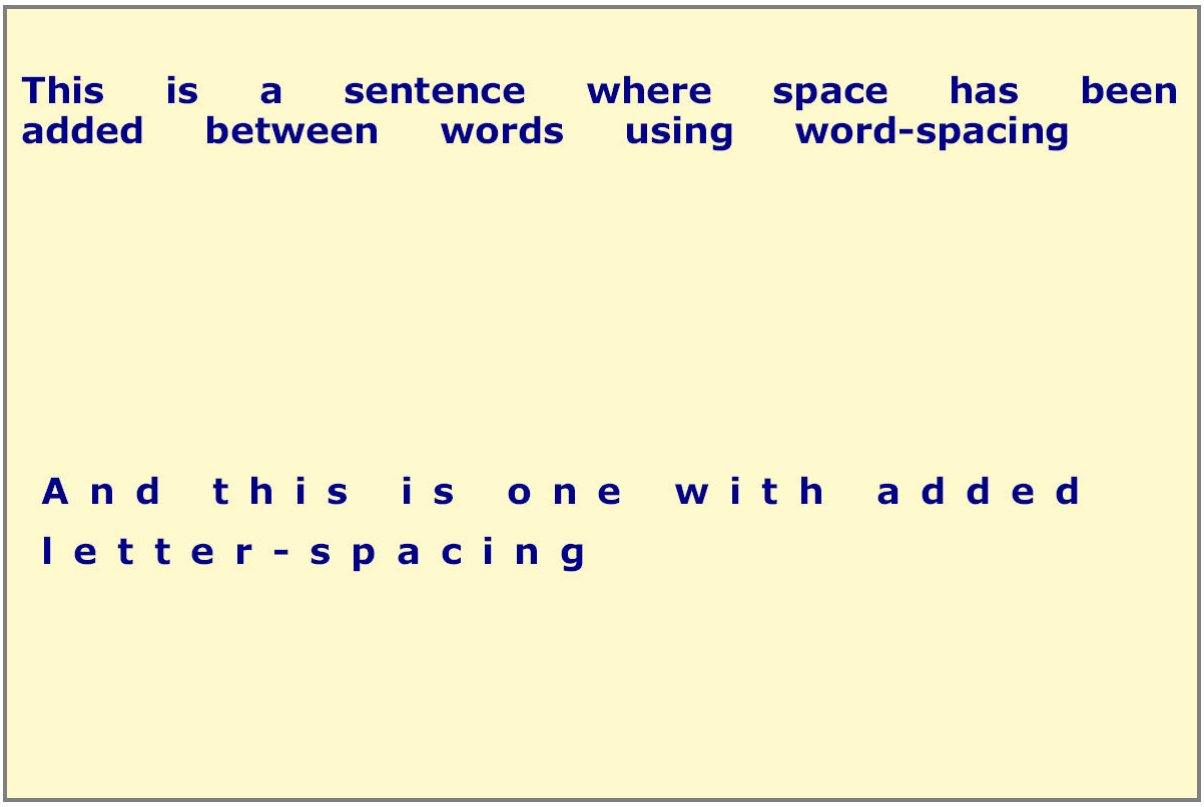

Figure 6.1: Word and Letter Spacing

6.3 Text Decoration (underlines, or otherwise highlights text)

text-decoration: none | [underline || overline || line-through || blink]

The text-decoration property defines how text is decorated. One or more settings can be made. For example:

```
h3 { text-decoration: underline blink }
```

6.4 Vertical Alignment

vertical-align: baseline | sub | super | top | text-top | middle | bottom | text-bottom | <percentage>

The vertical-align property defines the vertical positioning of an inline element (such as some text or even an image) relative to the current baseline. It could be used to produce x^2. The letter x in this case would be called the **parent** and the power 2 would be called the **child**.

The value may be a percentage of the element's line-height property. It specifies how much the element's baseline is raised above the parent's baseline. Negative values are permitted.

The other values are given in the following table.

Notation	Meaning
baseline	aligns baselines of element and with the parent's baseline
middle	aligns vertical midpoint of element with parent's baseline plus half the x-height (height of letter "x") of the parent
sub	subscripts the element
super	superscripts the element
text-top	aligns tops of element and top of parent element's font
text-bottom	aligns bottoms of element and bottom of parent element's font
top	aligns top of element with tallest element on the line
bottom	aligns bottom of element with lowest element on the line

Each font will have well-defined values for the aspects of the font defined in Figure 6.2 to aid it in achieving the various effects required for vertical alignment.

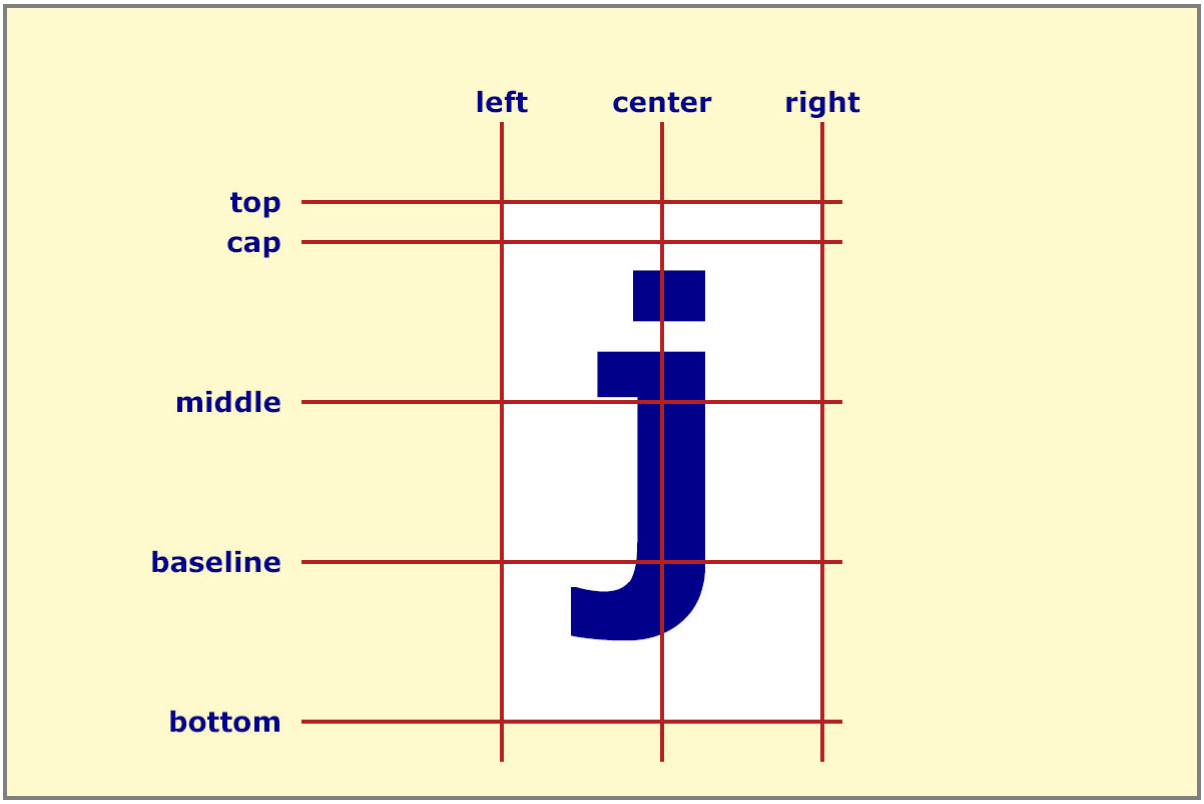

Figure 6.2: Font Aspects

The vertical-align property is also useful for aligning images. Some examples are:

```
em { vertical-align: 10% }
img { vertical-align: middle }
```

6.5 Text Transformation

text-transform: capitalize | uppercase | lowercase | none

The text-transform property allows text to be transformed by one of four properties:

Notation	Meaning
capitalize	capitalizes first character of each word
uppercase	capitalizes all characters of each word
lowercase	uses small letters for all characters of each word
none	leaves characters as defined

Examples:

```
h1 { text-transform: uppercase }
h2 { text-transform: capitalize }
```

For example, the text-transform property allows an organisation to define a house style for its documents irrespective of author preferences. The authors can have their own view of whether titles should be capitalized or not but the corporate house-style that is applied to all documents could use the text-transform property to insist on a particular style.

6.6 Text Alignment (set justification)

text-align: left | right | center | justify

The text-align property defines the horizontal alignment of text. For example:

```
h1 { text-align: center }
h2 {text-align: left }
h3 {text-align: right }
p  {text-align: justify }
```

6.7 Text Indentation (set distance from left margin)

text-indent: <length> | <percentage>

The text-indent property defines the amount of indentation that the first line of the element has. A percentage refers to the parent element's width. A common use is for indenting paragraphs. For example:

```
p { text-indent: 5em }
```

Negative values can be used to move the first line of the text out into the margin by the amount specified.

6.8 Line Height

line-height: normal | <number> | <length> | <percentage>

The line-height property defines the spacing between baselines of two adjacent lines of text It can be defined in points, inches, centimetres, or even pixels. When the value is a number, the line height is calculated by multiplying the element's font size by the number. Percentage values are relative to the element's font size. Negative values are not allowed.

The line-height property could be used to double space text:

```
p { line-height: 200% }
```

If the font-size is 10pt, the following rules all have the same effect:

```
p { line-height: 1.2 }
P { line-height: 1.2em }
P { line-height: 120% }
```

A value of normal is usually set to between 1.0 and 1.2

7. Box Properties

7.1 Introduction

Block-level elements such as **h1**, **p**, **ul** have an area that contains the content of the text in the element. CSS provides control of how that content is placed on the page. CSS allows the core content to be surrounded by padding, a border and a margin as shown in Figure 7.1

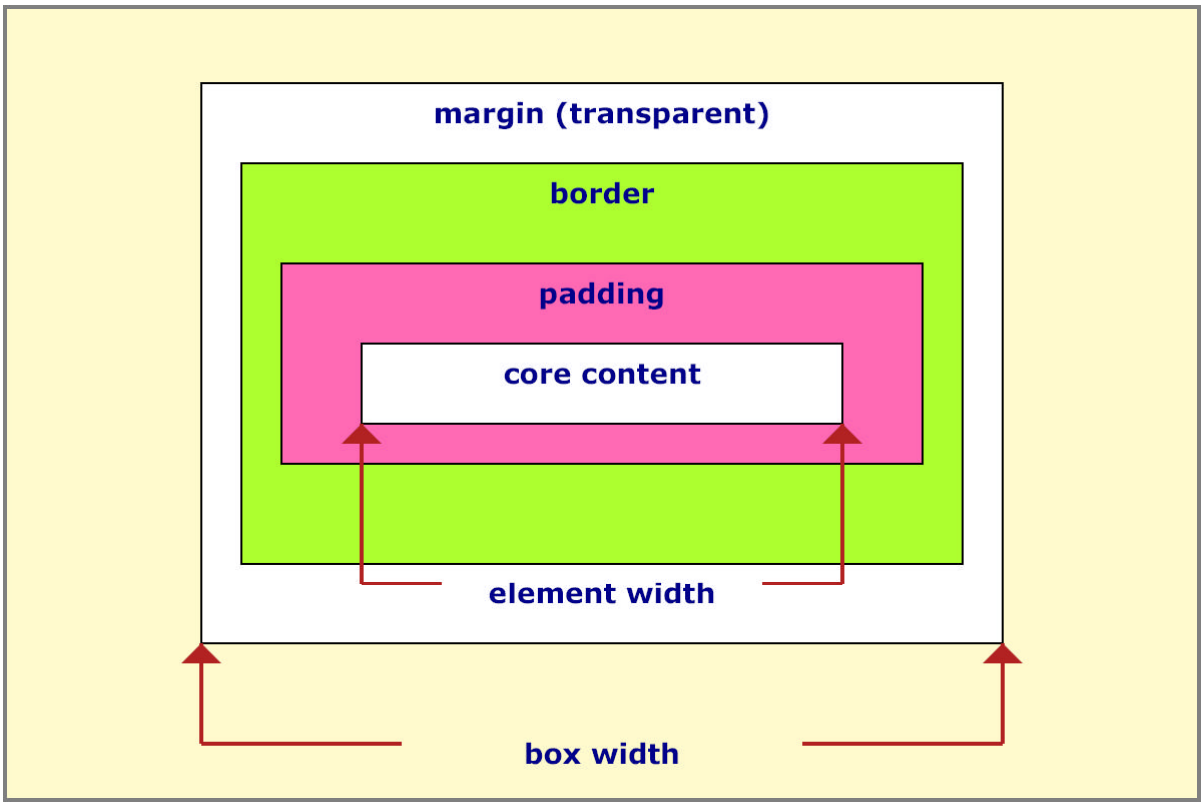

Figure 7.1: Box Model

The padding has the same background as the element, the border can be set to have a different background while the margin is always transparent so that the background of the parent element shows through. An example is shown in Figure 7.2.

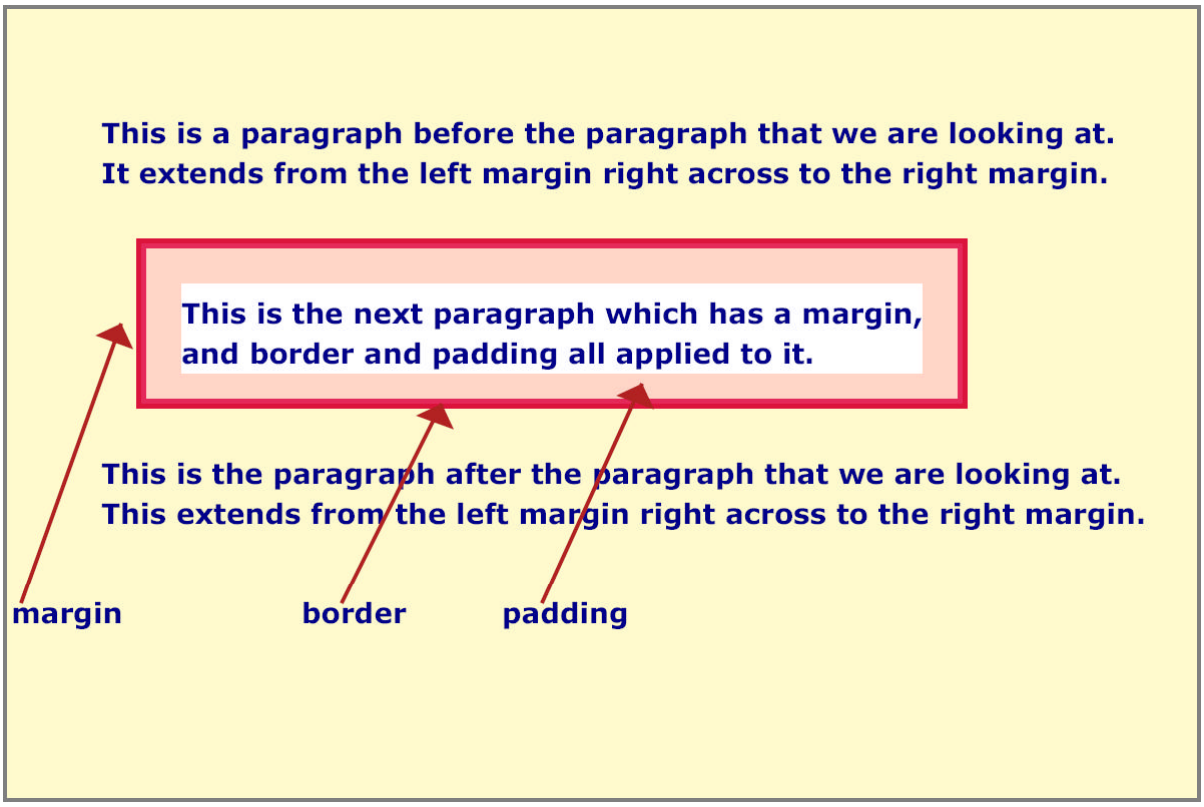

Figure 7.2: Example of box properties

If the next item below also has a margin, the top margin of the next element will be immediately below the border of the first element. Properties associated with margins, borders and padding can be set differently for the top, right, bottom and left sides (for example, their width). If all four are set by a single property (for example, margin), it is possible to specify between 1 and 4 values (notation **{1,4}**). The four values are defined in the order top, right, bottom and left. If less than 4 are specified, the rule is as follows. If only one value is given, it applies to all sides. If 2 or 3 are given, the missing values are the same as the opposite side.

7.2 Margin Setting

margin-top: <length> | <percentage> | auto
margin-right: <length> | <percentage> | auto
margin-bottom: <length> | <percentage> | auto
margin-left: <length> | <percentage> | auto
margin: [<length> | <percentage> | auto] {1,4}

The first four properties set one of the margins while the last property is a shorthand that can set all four in the order given above. The length can be given in points, inches, centimetres, or pixels. Percentage values are relative to the width of the parent element. Negative margins are permitted so that overlapping can be achieved.

The following rule would set no bottom margin for the document:

```
body { margin-bottom: 0 }
```

Other example might be:

```
li   { margin-left: 50% }
ul   { margin-bottom: 2em }
p    { margin-right: 1in }
body { margin: 1em 1in 2em 1in }
```

The last example sets the margin-top to 1em, the margin-bottom to 2em and the right and left margins to 1 inch.

7.3 Padding Setting

padding-top: <length> | <percentage>
padding-right: <length> | <percentage>
padding-bottom: <length> | <percentage>
padding-left: <length> | <percentage>
padding: [<length> | <percentage>] {1,4}

Padding is defined in much the same way as the margin. The main difference is that negative padding is not allowed. An example is:

```
h1 { background: red }
h1 { padding:   1em 1in }
```

The padding would also be coloured red.

7.4 Border Setting

border-top-width: thin | medium | thick | <length>
border-right-width: thin | medium | thick | <length>
border-bottom-width: thin | medium | thick | <length>
border-left-width: [thin | medium | thick | <length>
border-width: [thin | medium | thick | <length>]{1,4}

The general rules for defining borders are much as for margins and padding. However, negative borders are not allowed. The assumption is that the browser will make thin borders smaller than medium ones and thick ones larger than medium ones. No other guidance is given.

7.5 Border Color

border-color: <color> {1,4}

The border-color property sets the colour of the border. The four border parts (top, right, bottom, left) can be set to different colours. An example is:

```
P { border-color: black red black red}
```

This would set the border as a solid black line top and bottom and red at the sides.

7.6 Border Style

border-style: [none | dotted | dashed | solid | double | groove | ridge | inset |outset] {1,4}

The border-style property sets the style of the border and must be specified for the border to be visible (the default is none). For double, two lines are drawn and the width is between the outer edges of the two lines. Groove, ridge, inset and outset produce various 3D effects normally associated with buttons on a display.

7.7 Setting All Border Properties Together

border-top: [<border-top-width> || <border-style> || <border-color>]
border-right: [<border-right-width> || <border-style> || <border-color>]
border-bottom: [<border-bottom-width>|| <border-style> || <border-color>]
border-left: [<border-left-width> || <border-style> || <border-color>]
border: [<border-width> || <border-style> || <border-color>]

The first four properties are a shorthand for setting the width, style, and colour of one part of an element's border.

The border property is a shorthand for setting the same width, style, and colour to all four border parts. Examples of border declarations include:

```
h2 { border-top: dotted 3em }
p  { border-right: solid blue }
li { border-left: thin dotted #FF0000 }
h1 { border: medium blue double }
```

In the following example:

```
<html>
<style type="text/css">
<!--
ul { margin: A B C D }
ul { padding: E F G H }
ul { border: medium solid black }
ul { background-color: red }
li { margin: a b c d }
li { padding: e f g h }
li { border: medium solid black}
li { background-color: blue }
-->
</style>
<body>
<p>The possible values of the list are:</p>
<ul>
<li>first list element</li>
<li>second list element</li>
</ul>
<p>And this is where the next paragraph will be placed
</body>
</html>
```

The user has control of all the values shown in Figure 7.3.

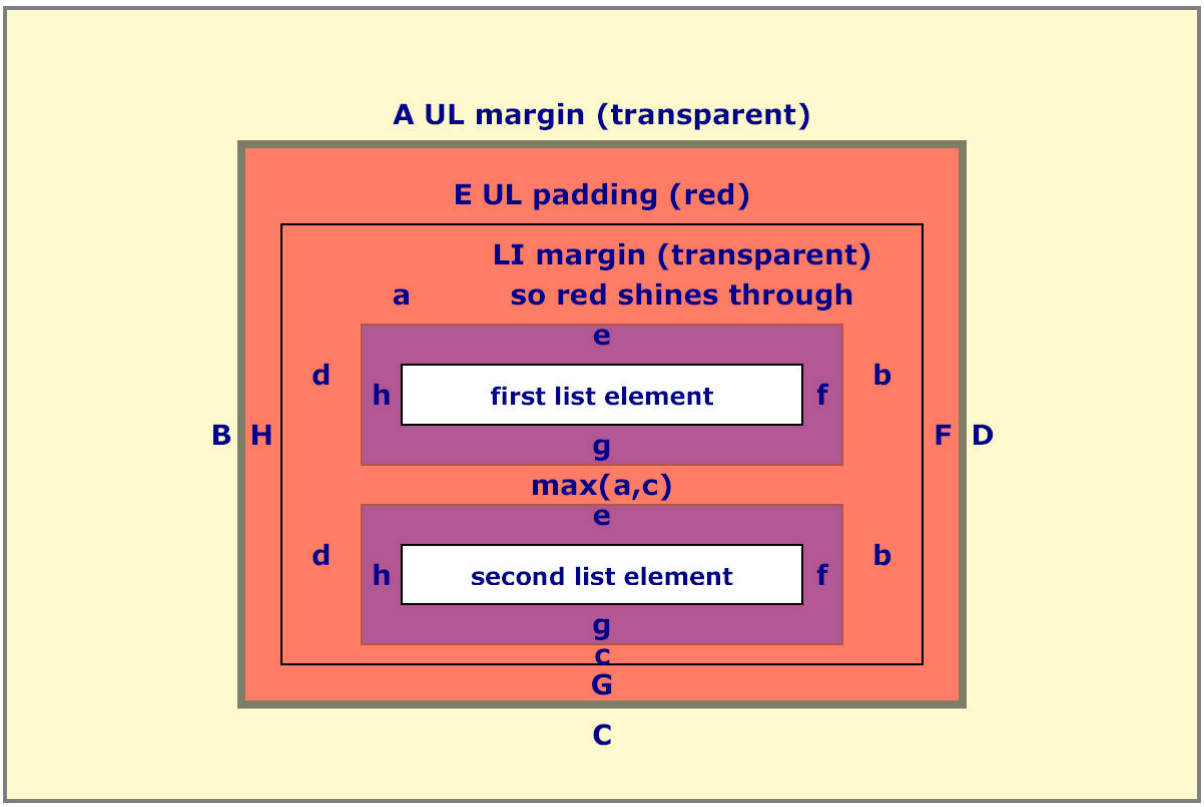

Figure 7.3: List Parameters

Note that the two margins in the middle are collapsed to a single margin.

The list in Figure 7.4 shows the kind of effect that can be achieved.

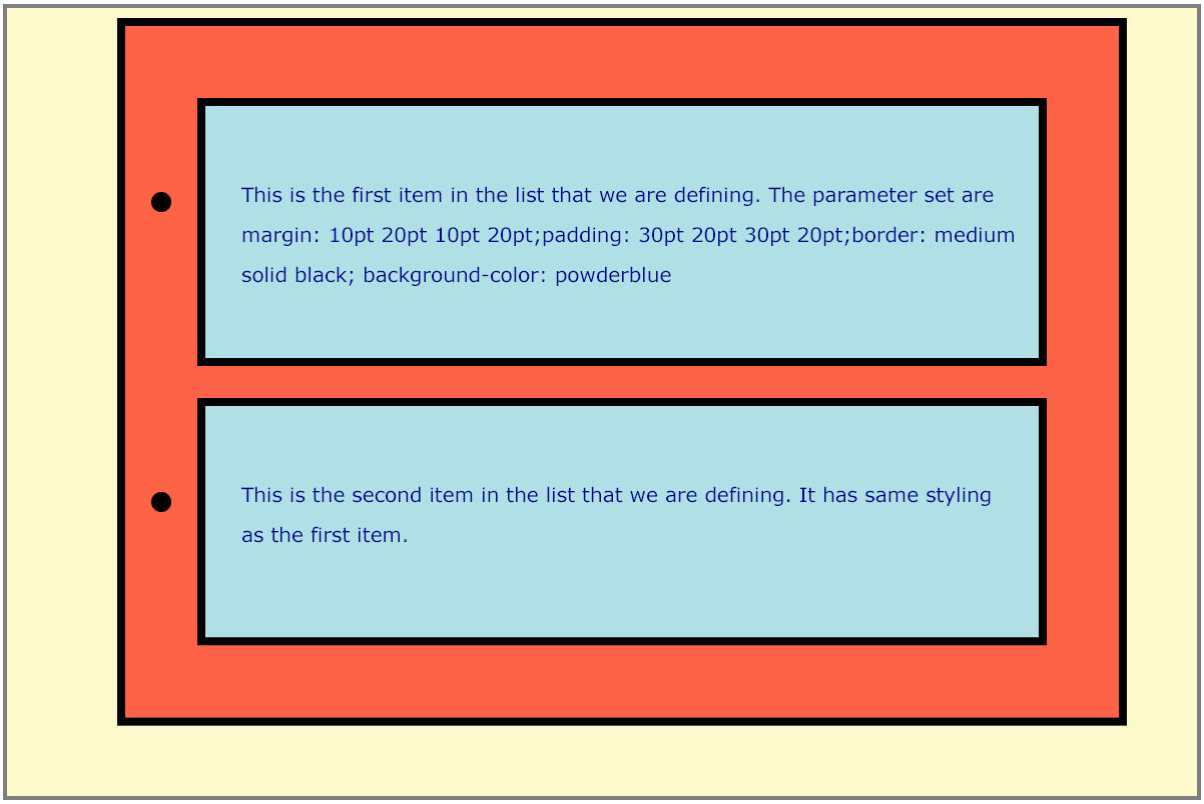

Figure 7.4: Example of a List

7.8 Width, Height of Images

width: <length> | <percentage> | auto
height: <length> | auto

These two properties apply to both block-level elements and replaced elements such as images. The width or height can be specified and the aspect ratio maintained by setting the other one to **auto**. If both are specified precisely, the scaling will not maintain the aspect ratio. If both are specified as auto, the intrinsic size of the image is used. For example:

```
img {width: 100px}
img {height: auto}
```

Percentage values for width refer to the parent element's width. Negative values are not allowed. Users should be aware that scaling by arbitrary amounts can severely degrade the image quality as can changing the aspect ratio so some care should be taken when using this property.

7.9 Float

float: left | right | none

The float property allows the user to wrap text around an element. If left is specified, the element floats to the left and the text wraps around to the right and vice versa. Figure 7.5 shows an image with the property **float:right** and shows how the paragraph that it is part of flows around the image.

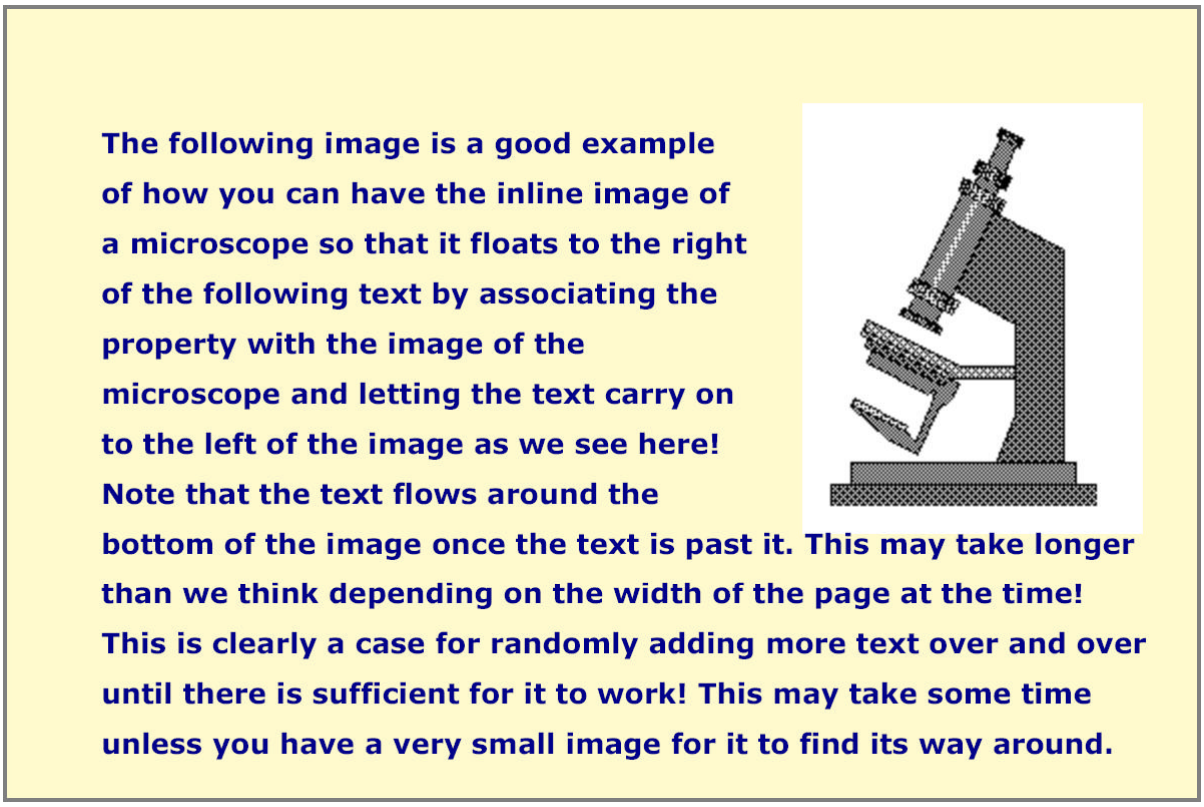

Figure 7.5: Image float:right

7.10 Clear

clear: none | left | right | both

The clear property can be used by box-level elements to ensure that a previous floating element is not adjacent to one of its sides. A value of **left** applied to an element such as **h1** would ensure that any images with float set to **left** defined above it in the page would not appear to its left. The element would effectively move below the image (by adding addtional space above the **h1** element). For example:

```
h1: {clear: left }
```

The value **right** ensures that it is does not overlap with any images floated to the right. The value **both** is equivalent to setting both **left** and **right**. The property can also be used by inline elements.

8. Classification Properties

8.1 Display

display: block | inline | list-item | run-in | inline-block | none

CSS when used with HTML has a good knowledge of which elements are inline elements and which are block-level. Even for elements defined by the user using **div** and **span** the type of element is known. The only thing not known is whether it is a list element which has different inheritance rules from normal paragraphs. If CSS is used with XML, the elements have no concept of inline and block-level and there needs to be a mechanism to define which elements are inline and which are block-level.

The **display** property describes how an element is displayed by giving an element one of four values:

Notation	Meaning
block	defines the element as being a new rectangular area positioned relative to other adjacent areas. H1 and P would normally displayed as a block.
inline	defines the element as a new area on the same line as the previous content.
list-item	same as block except a list-item marker is added.
run-in	can be inline or block depending on the context
inline-block	added in CSS 2.1. Acts as a block-level element but flows as though it was an inline box (like an image)
none	no display at all

Elements are given default display values by the browser, based on suggested rendering in the HTML specification.

Examples could be:

```
p  { display: block }
em  {display: inline }
li  {display: list-item }
img {display: none }
```

The **display** property allows the normal settings to be changed.

8.2 Whitespace

white-space: normal | pre | nowrap

The white-space property defines how spaces within an element are displayed. This property takes one of three values:

Notation	Meaning
normal	collapses multiple spaces into one
pre	does not collapse multiple spaces
nowrap	line wrapping is only done via an HTML **br** element

Examples could be:

```
pre { white-space: pre }
p   { white-space: normal }
```

8.3 List Style Type

list-style-type: disc | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none

The **list-style-type** property specifies the type of list-item marker to be used for ordered and unordered lists, and is used if:

- list-style-image (see 8.4) is none;
- image loading is turned off;
- the image pointed to by the URL cannot be displayed.

Some examples are:

```
li { list-style-type: square }
ul { list-style-type: decimal }
ol { list-style-type: lower-alpha }
```

The first displays small squares, the second 1, 2, 3, etc and the third a, b, c, etc.

8.4 List Style Image

list-style-image: <url> | none

The list-style-image property defines the image to be used as the list-item marker when image loading is turned on. It replaces the marker specified in the list-style-type property. For example:

```
ul { list-style-image: url("arrow.gif") }
li { list-style-image: url("triangle.png") }
```

8.5 List Style Position

list-style-position: inside | outside

This property defines where the marker is displayed relative to the list item. For **inside**, the lines will wrap under the marker instead of being indented while for **outside** it will not. An example is:

```
ul { list-style: outside }
ol {list-style: inside }
```

where the first would produce:

```
* List item 1 extending to
second line of list item
* List item 2 extending to
second line of list item
```

and the second would produce:

```
1. List item 1 extending to
   second line of list item
2. List item 2 extending to
   second line of list item
```

8.6 List Style

list-style: [<list-style-type> || <list-style-position> || <list-style-image>]

The list-style property is a shorthand for the list-style-type, list-style-position, and list-style-image properties. For example:

```
li { list-style: square inside }
ul { list-style: circle url(triangle.png) outside}
ol { list-style: upper-alpha none }
```

9. Structure and Control

9.1 Introduction

So far, individual rules have been applied to all elements of a specific type. While that may be appropriate for something like **h1**, it is less so for something like **p** as frequently a sub-set of the paragraphs in a document may require specific styling. Within HTML, a number of facilities are provided to allow more specialised control. These are described in this section.

9.2 Classes

The elements of a specific type can be grouped into classes by the HTML **class** attribute. For example:

```
<h1 class="firstsection"> Section One Heading </H1>
....
<h1 class="secondsection"> Section Two heading </H1>
...
<h1 class="thirdsection">Section Three Heading </H1>
...
```

The selector in a style rule need not be an element type but can be qualified by a class name. So for example:

```
<style>
h1.firstsection  {font: 15pt/17pt; color: red}
h1.secondsection {font: 15pt/17pt; color: green}
h1.thirdsection  {font: 15pt/17pt; color: blue}
</style>
```

Similarly, in a document written by two people you might have:

```
<style>
p.mytext  {font: 14pt/16pt; color: red}
p.histext {font: italic 12/14pt Times; color: green}
</style>
<body>
<p class="mytext"> This is a paragraph that I wrote.</p>
<p class="histext"> This is a paragraph that my partner wrote.</p>
</body>
```

In such an example, it is feasible that the other author also would have added headings and other features. Classes can be applied to different elements so the document could also contain:

```
<h1 class="histext"> His Title </h1>
```

In this case, the selector can just consist of the class name:

```
.histext  { font-size: small; color: green}
```

In this case, the **histext** class may be used with any element.

It is good practice to use classes to break up the content of a document into a semantically meaningful breakdown and consider how it should be styled later.

HTML allows several class names to be associated with a single element and each can be used for specifying styling. This allows the same element to be categorised in more than one way. For example:

```
<p class="mine code">x=y</p>
<p class="his ref"> [16]</p>
<p class="mine ref"> [18]</p>
<p class="his code">y=3</p>
```

Here the paragraphs are categorised on both their type and their author. Styling could be defined by:

```
.mine {color: red}
.his {color: green}
.code {font-family: Courier}
.ref {font-style:italic}
```

If there is a conflict between the two classes in the styling required, the normal precedence rules apply (see Section 12).

9.3 Identifiers

The **id** attributes in HTML are similar to classes in that they give another level of naming but in addition **id** attributes are unique and are associated with a single element. Thus it is possible to write:

```
<p id="abc123">Text quotation</p>
```

Each **id** attribute has a unique value over the document. The value consists of an initial letter followed by letters, numbers, digits, hyphens, or periods. The letters are restricted to A-Z and a-z.

The text above could be selected by:

```
#abc123 { text-transform: uppercase}
```

9.4 Contextual Selectors

Contextual selectors allow a finer specification based on the context of the element type. For example, emphasised sections within a **h1** element might be displayed differently from emphasised sections that appear within other headings. Contextual selectors consist of two or more simple selectors following each other. These selectors can be assigned properties and they will take precedence over simple selectors so for example:

```
h1 em { color: green }
p em { color: red }
div p em { background: blue }
```

The emphasised text within an **h1** element would be green while those in paragraphs would be red unless the paragraph resided within a **div** element in which case the emphasised text would have a blue background.

The selector only requires the second element specified to be a descendant of the first and not an immediate child. To insist that the second element is a child of the first element requires:

```
h1 > em { color: green }
p > em { color: red }
div > p > em { background: blue }
```

9.5 Grouping

If the same rule applies to a set of elements, they can be grouped together. For example, all of the headings could be given the same font and colour by:

```
h1, h2, h3, h4, h5, h6 {color: red; font-family:sans-serif }
```

9.6 Universal Selector

To match any element, it is possible to replace the element name in the selector by the symbol ***. In most of the examples used so far, there is no real use for the universal selector as omitting the name implies all possibilities. Where it is useful is within contectual selectors. For example:

```
div * li {color:red }
```

Within any **div** element, list items will be red independent of whether they are within **ul** or **ol** lists.

9.7 Attribute Selectors

It is possible to do a selection based on whether an element has an attribute and whether that attribute has a specified value. The possibilities are illustrated below.

```
h1[class] { color:blue }
h1[class="main"] { color:red }
h1[class~="main"] { color:green }
h1[class="main"][id="fred"] { color:yellow }
```

The first example will set to blue all **h1** elements that have a class attribute. All **h1** elements that have the class set exactly to **main** will be red. Those **h1** elements that have main as part of the class value will be green and the one that has **id** set to **fred** as well will be set to yellow.

9.8 Comments

Comments are denoted within style sheets with the same conventions that are used in C programming. A sample CSS comment would be:

```
p {color: green }  /* A comment for this rule */
```

9.9 The Elements div and span

The two HTML elements **div** (for division) and **span** (for span over) can be used in conjunction with the **calss** attribute to define new HTML elements. For example:

```
<div> class = "newh1">
A Different Kind of Title
</div>
```

effectively defines a new HTML element **newh1**. A selector of the form:

```
div.newh1 { color: green }
```

would specify how this new element would appear. The **span** element acts in much the same way as **div** but is an inline rather than a block-level element.

10. Pseudo-classes and Pseudo-elements

10.1 Introduction

Pseudo-classes and pseudo-elements are similar to classes but are special in that they are automatically added to certain elements or sub-parts of elements by a CSS-supporting browser. Rules for pseudo-classes or pseudo-elements take the form

```
selector:pseudo-class { property: value }
selector:pseudo-element { property: value }
```

10.2 Anchor Pseudo-class

Pseudo-classes are assigned to an anchor element to allow links to be displayed differently depending on whether they are visited or active links. A visited link could be displayed in a different colour and font size, for example. An active link is a link that is being followed at the moment by the browser. Normally, this would only be visible if the browser was displaying a hierarchy of pages. Examples might be:

```
a:link { color: red }
a:active { color: green; font-size: 140% }
a:visited{ color: blue; font-size: 60%; text-decoration: none}
```

The third rule would remove the underline from links that had already been visited.

Some browsers provide user support for how links should be displayed. In consequence, the styling of links may not be as useful as might be thought. The effect proposed by the stylesheet may not appear.

10.3 First Line Pseudo-element

Many typographic styles start by capitalising or making bold the first line of text in an article. CSS provides support for this via a first-line pseudo-element. For example:

```
P:first-line { text-transform: capitalize; font-weight: bold }
```

might display the paragraph:

```
<p>
When computers first were used in office automation, they were
ineffective.
</p>
```

with the first few words that appear on the first line set in capitals.

Figure 10.1: Property first-line:capitalize

10.4 First Letter Pseudo-element

The first-letter pseudo-element is used to create effects on the initial letter of an element. For example:

```
P:first-letter { font-size: 200%; float: left }
P:first-line { text-transform: capitalize; font-weight: bold }
```

would create a drop cap twice the normal font size for the initial letter and the first line capitalised as shown in Figure 10.2.

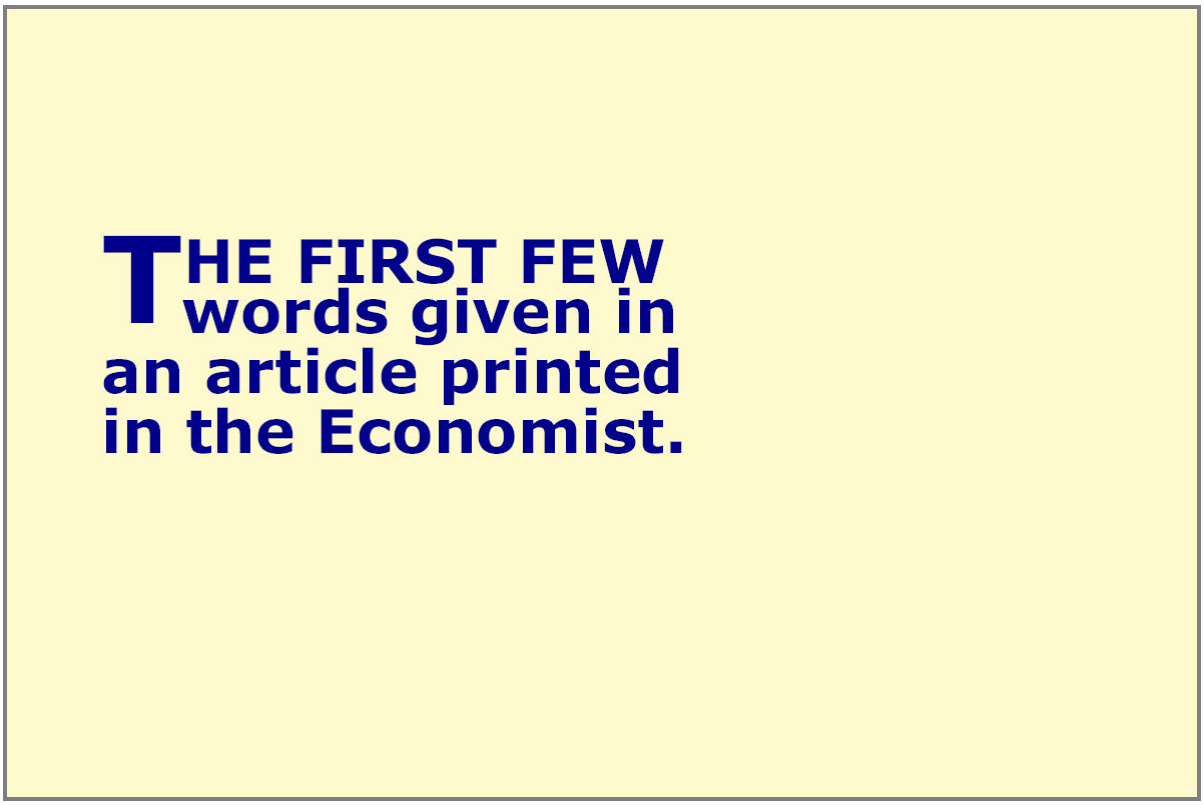

Figure 10.2: Example of a list

10.5 Context Pseudo-elements

content: [<string> | <url> | <counter> | attr(X) | open-quote | close-quote]+

Two pseudo-elements exist to allow information to be added before or after a specific element. Such information is called **generated content**. For example:

```
p.note:before {content:"Note: "}
p.note:after {content:" (end of Note)"
. . .
<p class=note">A paragraph</p>
```

The two pseudo-elements **:before** and **:after** are used to add **content** to the document to be presented that does not appear in the original HTML. In this example paragraphs of **class note** have the text **Note:** added before the paragraph and the text **(end of Note)** added after the paragraph. This would generate a paragraph looking like:

```
Note: A paragraph (end of Note)
```

An example of generated content that has always been there in HTML is the decoration for lists (bullets or numbers) and this facility allows users both to tailor that decoration and produce decoration for their own uses. It is possible to point to a resource thus allowing a large block of content or even an image to be inserted. While there is good support for the facility in Netscape and Opera's latest offerings, the facility is not supported in IE5.5.

Two other examples to illustrate how the facility could be used are:

```
p {counter-increment: par-num}
h1 {counter-reset: par-num}
p:before {content: counter(par-num, upper-roman) ". "}
. . .
img:before {content: attr(alt) }
```

In the first, the property **counter-reset** resets the value of the variable **par-num** to zero. Each **p** causes the variable to be incremented and output before the paragraph. Consequently, all the paragraphs are numbered in each major section starting each time from 1.

In the second example, the **img** element has the value of its **alt** attribute output before the image. For a text-only browser, it would be possible to remove the image and replace it with the **alt** text. For example:

```
img { width: 0 !important; height: 0 !important }
img:before { content: "[" attr(alt) "]"; color: yellow !important;
background: red !important; margin: 0 0.33em !important }
```

The image is reduced to zero size and the alternative text is presented in yellow on a red background.

11. Linking Style Sheets to HTML

11.1 Linking to an External Style Sheet

```
<link rel=<stylesheet> href= <url> type="text/css" [title=<string>]? [ media="<media>"]?
>
<media> ::= [ <medium> [,<medium>]* ]
<medium> ::= [ print | screen | projection | braille | embossed | handheld | aural | all ]
<stylesheet> ::= [stylesheet | alternate stylesheet]
```

To link to an external style sheet, the author creates a file containing the appropriate style rules and uses the **link** command as above. Convention is to name the file with a **.css** file extension. It allows the same style sheet to be used for any number of pages. More than one stylesheet can be provided for each media. One is defined with the **stylesheet** attribute set to **stylesheet** and the others set to **alternate stylesheet**. Both Netscape 6 and Opera 7 allow you to select which style you want from the **View** menu.

If the style sheets are called mystyles.css and myaltstyles.css and are located at:

http://cclrc.ac.uk/

the style sheets would be added by:

```
<head>
<title>Title of page</title>
<link rel="stylesheet" href="http://cclrc.ac.uk/mystyles.css" >
<link rel="alternate stylesheet"
href="http://cclrc.ac.uk/myaltstyles.css" >
</head>
```

Other examples of **link** are:

```
<link rel="stylesheet" href="style.css" type="text/css" media="screen">
<link rel="stylesheet" href="another.css" type="text/css"
media="screen, print">
<link rel="stylesheet" href="aural.css" type="text/css" media="aural">
```

The **<link>** tag is placed in the document **head**. The optional **type** attribute is used to specify a media type. The type **text/css** specifies a Cascading Style Sheet. This allows browsers to ignore style sheet types that they do not support. Configuring the server to send **text/css** as the Content-type for style sheets files is also sensible.

Internet Explorer registers the Internet Media (MIME) type for style sheets, so it is not necessary for the user to register the "text/css" type on the server.

External style sheets should not contain any HTML tags. The style sheet should consist merely of style rules. For example:

```
p { color: red }
```

in a file could be used as an external style sheet.

The **<link>** tag has an optional **media** attribute, which specifies those media to which the style sheet should be applied. Possible values are:

Value	Meaning
print	for output to a printer
screen	for presentation on computer screens
projection	for projected presentations
braille	for presentation on Braille tactile feedback devices
embossed	for paged braille printers
handheld	for small monochrome screens (phone, PDA)
projection	for overhead data projectors
tv	for low resolution devices with limited scrolling
tty	for limited devices with fixed width characters
aural	for aural presentation
all	for all output devices (the default)

Multiple media are specified through a comma-separated list or the value **all**.

The **rel** attribute is used to define the relationship between the linked file and the HTML document. By setting **rel="stylesheet"** a style sheet is specified.

A single style sheet may also be provided by multiple style sheet **link**s:

```
<link rel="stylesheet" href="main.css" title="House">
<link rel="stylesheet" href="tables.css" title="House">
<link rel="stylesheet" href="forms.css" title="House">
```

In this example, three style sheets are combined into one "House" style that is applied as a default style sheet. To combine multiple style sheets into a single style, one must use the same title with each style sheet.

External style sheet are best used when the style is to be applied to many pages. Thus an external style sheet could be used by an author to change the look of an entire site by changing the house style sheet. Most browsers should cache an external style sheet thus avoiding a delay in page presentation once the style sheet has been cached.

11.2 Embedding a Style Sheet

<style type="text/css"> [title=<string>]?[media = <media>]? <rules> </style>

To embed a style sheet in an HTML document,a **<style> </style>** block needs to be added at the top of the document, between the **<html>** and **<body>** tags. The **<style>** tag has an attribute, **type** which specifies the Internet Media type as **text/css** (allowing browsers that do not support this type to ignore style sheets). The **<style>** tag is followed by any number of style definitions or rules and terminated with the **</style>** tag.

The following example specifies styles for the **<body>**, **<h1>**, **<h2>**, and **<p>** tags:

```
<html>
<style type="text/css">
body {font: 10pt "Times"}
h1  {font: 15pt/17pt "Palatino"; font-weight:bold; color: maroon}
h2  {font: 13pt/15pt "Palatino"; font-weight:bold; color: blue}
p   {font: 10pt/12pt "Palatino";  color: black}
</style>
<body>
...
</body>
</html>
```

Another example is:

```
<style type="text/css" media="screen"
body  { background:url(grass.gif) red; color: black }
p em { background: yellow; color: black}
.note { margin-right: 1in; margin-left: 1in }
</style>
```

The **style** element is placed in the document head. The type attribute is used to specify a media type in the same way as the **link** element. Similarly, the title and media attributes may be specified with styling.

Embedded style sheet are normally used when a document has a unique style.

11.3 Importing a Style Sheet

A style sheet may be imported using the import statement. This statement may appear in a .css file or inside the **style** element. For example:

```
<style type="text/css" media="screen">
@import url(http://www.clrc.ac.uk/ralstyle.css);
@import url(/stylesheets/local.css);
p {background: yellow; color: black }
</style>
```

All @import statements **must** occur at the start of the style sheet. Any rules specified in the style sheet override rules in the imported style sheets so in our example, if an imported style sheets contains a P definition, paragraphs would still have a yellow background.

The order in which style sheets are imported is important as it determines how cascading takes place. If the ralstyle.css imported specified that **em** elements be displayed in blue and the local.css style sheet specified that **em** elements be shown in yellow, the second rule would define that the **em** elements should be in yellow.

Imported style sheets can be modularised in a number of ways. For example, a site might define separate style sheets for each element. It might have a **simple.css** style sheet for rules such as **body**, **p**, **h1**, and **h2** and an **extra.css** style sheet for less common elements such as **code**, **blockquote**, and **dfn**. There might be a **tables.css** style sheet if table elements are to be used. Which style sheets are included in HTML documents will depend on the need.

11.4 Inlining Style

Style may be inlined using the **style** attribute associated with individual elements. For example:

```
<p style="color: red; font-family: 'New Century Schoolbook', serif">
</p>
```

Only this paragraph is styled by this **style** attribute, specifically in red with the New Century Schoolbook font, if available. Note that New Century Schoolbook is contained within single quotes in the **style** attribute as double quotes are used to contain the style declarations. Inlining styles mix content with presentation. It is recommended that this is used only as a last resort. They also apply to all media, as there is no method for specifying the medium for an inlined style.

11.5 Mixing Methods

Initially, it is probably a good idea to use just one method of specifying style sheets. However, if you want to have a corporate style that can be changed when justified it might well be sensible to link to the corporate style and embed the style local to the page. Similarly, the reader may well have his preferred style linked in rather than specify it for each document.

12. Cascading Order and Inheritance

Inheritance has been implicit in some of the examples used already. Selectors which define a broad class of elements will be inherited by a narrower selector. For example, a colour defined for the **body** of a document will also be applied to text in a paragraph as the paragraph is part of the body. Properties defined for all paragraphs will also apply to paragraphs of a sub-class if not overwritten.

Style sheets can be defined for each page. The style sheet for a page can include a style sheet imported from another page. Readers can specify their own style sheet. When multiple style sheets are used, the style sheets may all want to control the same selection. Therefore, rules are required to specify which style sheet rule will take precedence. The precise rules are quite complicated but are approximately as follows:

In the example above, it is assumed that the author has started with some general style sheet that is imported. This has been followed by the local house style sheet for the company. The author has defined some generic style that is probably linked in to each page of a set and finally there may be some specific styling for the individual page. Within the page some special effect may be required on just one element or part of an element to make a specific point. An author's style sheet takes precedence over a reader's which takes precedence over a browser's. As the author becomes more specific, that style takes precedence over less specific ones.

For some situations, it is important that the reader's preferences take control (for example, a person with limited vision might want to increase the font sizes; somebody using his mobile phone might want to decrease them!). Thus, browsers need to have an ability to turn off the author's style or to change these precedence rules between author and reader.

The relative ordering can be changed by insisting that a particular style is more important than its place in the hierarchy suggests. For example, the company may insist that some aspects of the house style must be adhered to under all circumstances. The priority of a particular style rule can be enhanced by the **!important** attribute. For example:

```
h1 { font-size 16pt !important; background: maroon !important }
```

A browser must define all the relevant properties for all elements. This is achieved as follows:

1. See if the cascade through the style sheets defines a value for the specific property for that element. If that is found, use it.
2. If the cascade does not define a value, see if the property is inherited.
3. If neither the cascade or the inheritance defines a value, use the property's default initial value if there is one. Normally if one is not specified, the browser will define one.

The style sheets come from three different origins:

1. **Author:** all that we have talked about so far have been the author's styling for the document.
2. **User:** users are allowed to define their own style sheet and browsers must provide support for them. For example, in IE6, under the Tools/Internet Options/Accessibility menu, the user can define their own style sheet. Mozilla and Netscape require the user stylesheet to be placed in a specific directory with the name userContent.css. Opera is similar to IE6.
3. **Browser:** browsers will provide a default style sheet so that if no styling is provided, titles do appear as titles and emphasised text is emphasised.

The precedence rules for the three origins is quite simple:

1. Author and User style sheets take precedence over the browser style sheet.
2. Author style sheet takes precedence over User style sheet unless the rule is defined as **! important** in which case it is the other way around.

So the order is:

1. User styles marked !important
2. Author styles marked !important
3. Author styles
4. User styles
5. Browser styles

The author may have defined the styling for the document by linking to a style, including a style element at the head of the document (and this may import style sheets). Finally the author may have defined a style attribute for a specific element. The rules as to which takes precedence here is:

1. A **style** attribute attached to an element takes precedence over the style sheet defined by the **style** element in the head of the document.
2. If the **style** element imports style sheets these **must** appear before any locally defined styling and the latter takes precedence over the imported styling.
3. The imported style sheet may also have imported style sheets within it and these are of lower precedence than itself and so on.
4. The linked style sheet has lower precedence to the one defined by the **style** element.

Note that precedence here is between rules of equal specificity. More specific rules will always take precedence over less specific ones whatever the source.

This allows us to remove rules of the same specificity if variants come from several sources. At the end of the prioritising, only one is retained.

To define which is the most specific rule, we need to check through the rules of different specificity. Here the rules are:

1. More specific selectors take precedence over less specific ones. **id** attributes take precedence over **class** attributes and they take precedence over element names. So a single **id** takes precedence over several **class** names and a single **class** name takes precedence over several element names.
2. If after all that several rules still remain with the same specificity, the one that appears last in the text is taken.

To ensure that an element inherits a value of a property from its parent, the only way to be sure is to have the most specific rule that wins the above cascade defining the property as having the value **inherit**. A common mistake is to have a document as follows:

```
<html>
<style type="text/css">
<!--
ul { color:red }
li { font-size:12pt }
-->
</style>
<body>
<p>The possible values of the list are:</p>
<ul style="font-size:14pt">
<li>first list element</li>
<li>second list element</li>
</ul>
</body>
</html>
```

The font-size of the list elements will be 12pt and not 14pt. The only way for the 14pt to be inherited by the list elements is if there are no styling rules for list elements that apply to this list.

13. Layout and Media

13.1 Positioning

position: static | relative | absolute | fixed | inherit

The normal flow for the position of a block-level element is to position it immediately after the previous element. We have seen that for images it is possible to move them out of the normal flow by allowing them to **float** to the left or right. For each block-level element, it is possible to define the **position** of that element using one of the values given above:

static
Block-level element is laid out according to normal flow
relative
Properties left, right, top and bottom define offsets with respect to the position it would normally be at
absolute:
relative to the containing block-level element so not really absolute
fixed
fixed **relative** to the viewport

Of these, **fixed** is poorly supported in browsers but **absolute** is although not completely correctly in some.

```
p.left { style="float:left;clear:left;background-color:silver;
width:100pt;border:solid;margin:5pt} . . . <p class="left">Button<p>
```

This would position a button on the left of the display. Several uses of this and similar elements could be used to create a layout as shown in Figure 13.1.

Figure 13.1: Layout using float and clear

Positions of block-level elements can be made absolutely as follows:

```
<p style="background-color:lemonchiffon; width:80pt;
margin:2pt;position:absolute; top:75pt;left:125pt">Button 1</p>
```

By varying the values of **top** and **left** an arrangement of buttons under the control of the author can be achieved as shown in Figure 13.2.

13.2 Media

Within a style sheet it is possible to define rules that only apply to a specific media by using the **@media** rule. Some examples are:

```
@media print, handheld { body { font-size: 10pt } }
@media screen { body { font-size: 12pt } }
@media projection
{ body { font-size: 16pt }
p { font-size:10pt}
. . .
}
```

The first rule only applies if the media is print or a handheld device. The second applies if the output is the display. The third shows how a set of styling can all be defined for a specfic media by enclosing them in a single **@media** rule. The possible media types are listed below.

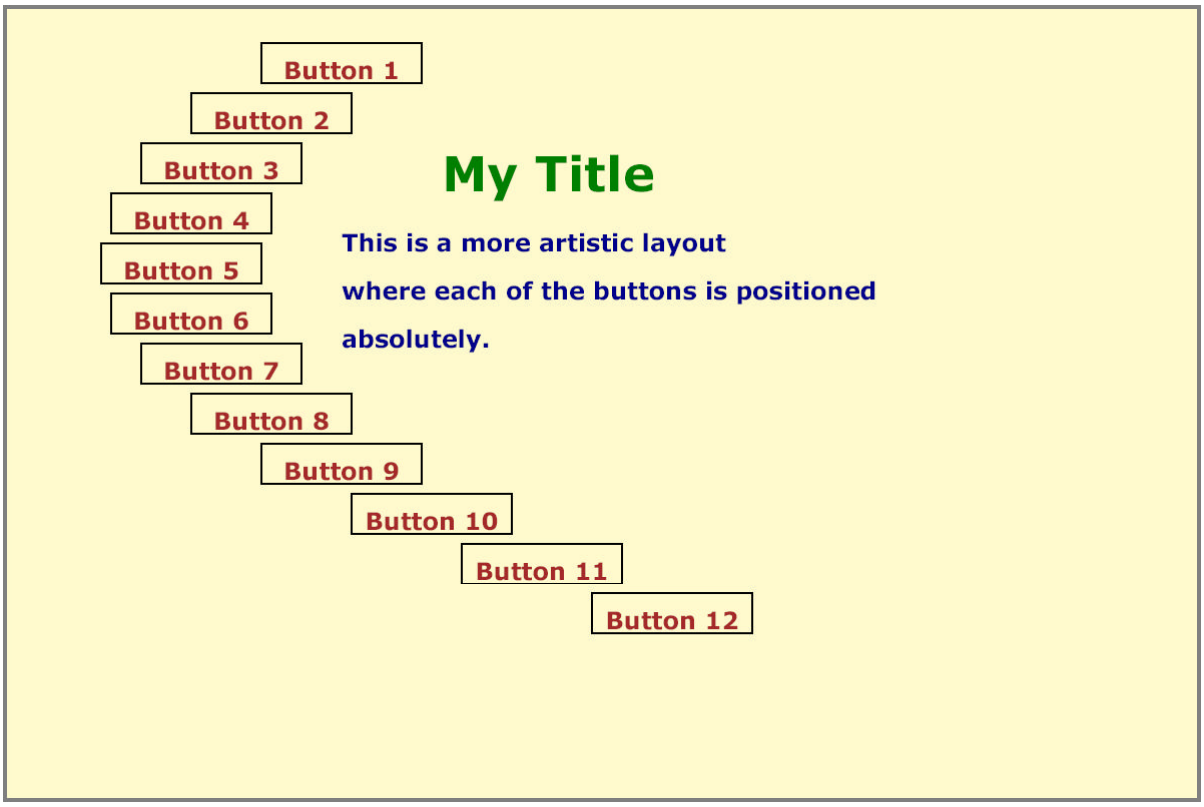

Figure 13.2: Absolute Positioning

all
All devices
aural
for speech synthesisers
braille
braille tactile feedback devices
embossed
paged braille printers
handheld
small monochrome screens (phone)
print
paged devices
projection
overhead projectors
screen
normal display
tty
limited print devices
tv
low resolution

Media such as print and aural have styling specific to that media.

13.3 Printing

Printing devices have the concept of a page. For outputting to such devices there is a need to control the size of a page and when a page break occurs.

size: <length>{1,2} | auto | portrait | landscape

The size property together with margin allows the author to define the characteristics of the page:

```
@page {size: 8.5in 11in; margin: 2cm}

@page:left {margin-left: 4cm; margin-right: 3cm}

@page:first {margin-top: 10cm}
```

The property **@media** has the pseudo-elements **left**, **right** and **first** associated with it to allow specific styling for the first page and left and right pages.

page-break-before: auto | always | avoid | left | right
page-break-after: auto | always | avoid | left | right
page-break-inside: avoid | auto
orphans: <integer>
widows: <integer>

Some examples are:

```
h1 {page-break-before: always}

p {page-break-inside: always; orphans: 2}
```

The property **orphans** specifies the minimum number of lines of a paragraph at the bottom of a page. The property **widows** specifies the minimum number of lines of a paragraph at the top of a page. Between them they help to provide some control over when a page break should appear. For ensuring that chapters start on new pages and similar requirements, the propert **page-break** has the following possibilities:

auto
Don't force or forbid page break (before, after, inside)
always
Always force a page break (before, after)
avoid
Avoid a page break (before, after, inside)
left
Force page breaks (before, after) to make the next page a left one
right
Force page breaks (before, after) to make the next page a right one

13.4 Aural Output

The aural rendering of a document is particularly useful for people with accessibility problems (blind or in a car) that make it difficult to see the document. Browsers have been developed that have this as their major form of output. They can also be used in a teaching environment such as learning to read where a mixture of text and aural output adds to the learning experience. For aural devices, the space in which the document is rendered is 3-dimensional and has a temporal dimension. Some of the properties special to this media are:

volume
The median volume of the aural output
speak
Defines whether text will be converted to speech and whether the individual letters will be spelled ou.
pause
Defines any pausing required
cue
Defines any lead in or lead out required to some aural rendering
play-during
Specifies background aural output, similar to **background-color** for display devices.
azimuth, elevation
Defines the position of the aural output in the 3-dimensional space
Voice Characteristics
A set of properties allowing differentiation between several aural renderings
speak-punctuation, speak-numeral
Defines how punctuation and numbers should be presented

The relevant definitions are:

volume : <number> | <percentage> | silent | x-soft | soft | medium | loud | x-loud
speak: normal | none | spell-out
pause-before: <time> | <percentage>
pause-before: <time> | <percentage>
pause: [<time> | <percentage>] {1,2}
cue-before: <url> | none
cue-after: <url> | none
cue: [<url> | none] [<url> | none]
play-during: <url> mix? repeat? | auto | none
azimuth: <angle> | <auralposition> | leftwards | rightwards
<auralposition> ::= [left-side | far-left | left | center-left | center | center-right | right | far-right | right-side] || behind
elevation: <angle> | below | level | above | higher | lower
speech-rate: <number | x-slow | slow | medium | fast | x-fast | faster | slower
voice-family: [specific-voice}, .. [generic-voice]
pitch: <frequency> | x-low | low | medium | high | x-high
pitch-range: <number> stress: <number> richness: <number>
speak-punctuation: code | none speak-numeral: digits | continuous

Many of these properties have reasonably obvious meanings. An example style sheet might be:

```
h1 {voice-family: paul; cue-before: url("ping.au") }
p.heidi {azimuth: center-left}
p.peter {azimuth: far-right}
p.goat {volume: x-soft}
em {speak: spell-out}
ul {pause-before: 20ms}
blockquote.sad {play-during: url("violins.aiff") }
p.god {elevation: above; speech-rate: slow; volume:loud}
code {speak-punctuation: code; speak-numeral: digits}
```

14. Tables

14.1 Two Models

Styling of tables in CSS is complicated by the fact that there are two different models available:

- collapsing borders
- separate borders

The first only allows one border to appear while the second allows multiple borders. The second is the default. Suppose we have the table:

```
<table title="Membership of W3C" summary="Test Table">
<caption>W3C Membership</caption>
<thead>
<tr>
<th>Type</th><th>Americas</th><th>Europe</th><th>Pacific</th><th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<th>Full</th><td>62</td><td>29</td><td>17</td><td>108</td>
</tr>
<tr>
<th>Affiliate</th><td>240</td><td>123</td><td>47</td><td>410</td>
</tr>
<tr>
<th>Total</th><td>302</td><td>152</td><td>64</td><th>518</th>
</tr>
</tbody>
</table>
```

If the style sheet is:

```
table { border-collapse:separate }
td {border:solid blue 2pt}
th {border:solid red 1pt}
```

The result would be:

W3C Membership

Type	Americas	Europe	Pacific	Total
Full	62	29	17	108
Affiliate	240	123	47	410
Total	302	152	64	**518**

By changing the styling to:

```
td {border:solid blue 2pt}
th {border:solid red 1pt}
table {border:dotted green 3pt; border-collapse:separate;}
```

the result becomes:

W3C Membership

Type	Americas	Europe	Pacific	Total
Full	62	29	17	108
Affiliate	240	123	47	410
Total	302	152	64	518

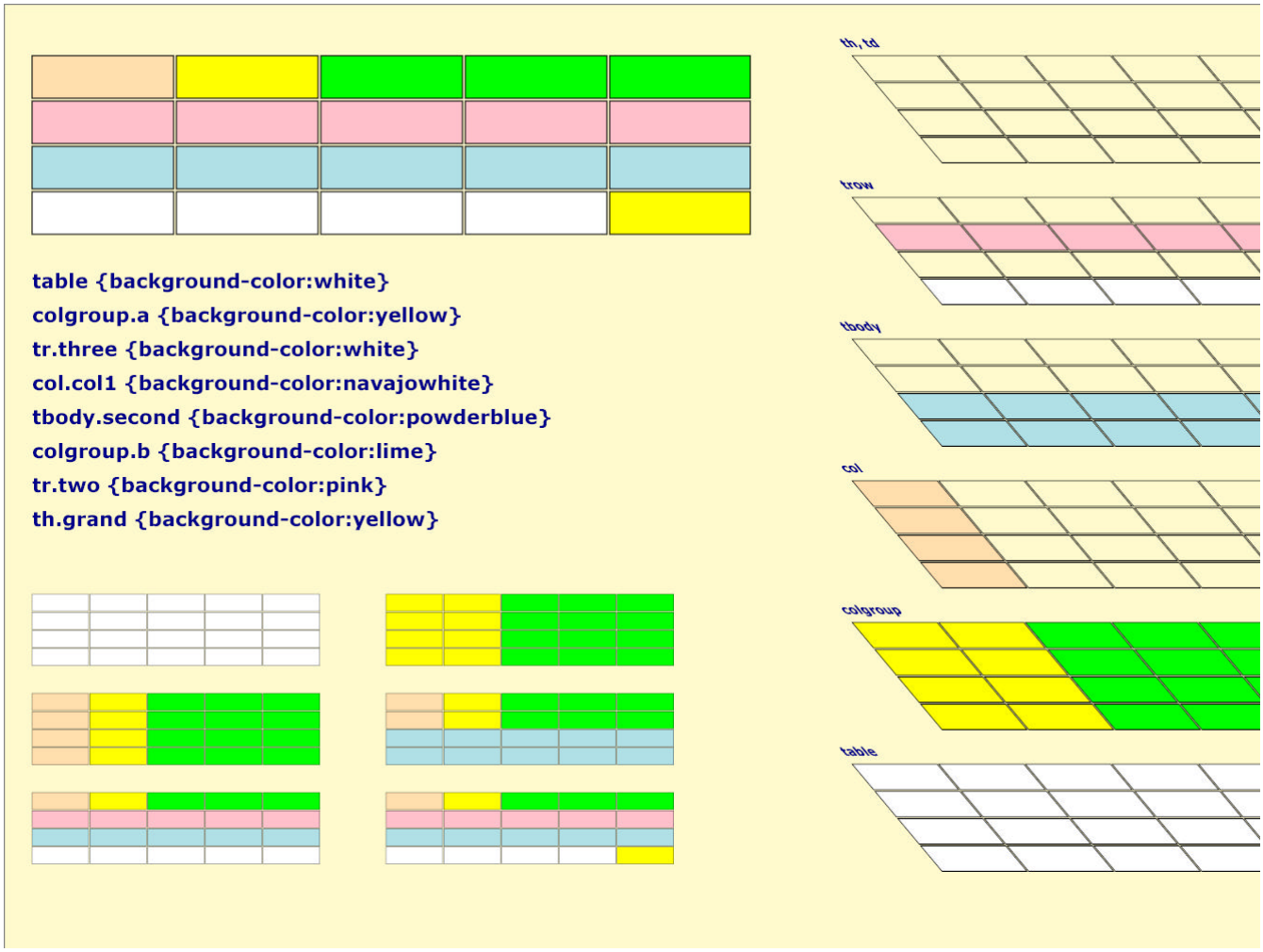

Figure 14.1: Table Properties

By setting the border to collapse results in:

```
td {border:solid blue 2pt}
th {border:solid red 1pt}
table {border:dotted green 3pt; border-collapse:collapse;}
```

The result becomes:

W3C Membership

Type	Americas	Europe	Pacific	Total
Full	62	29	17	108
Affiliate	240	123	47	410
Total	302	152	64	518

Rows and columns can be styled individually and there is a well-defined algorithm that states which takes precedence (see Figure 15.1). Cells take precedence over rows and so on.

An example styling columns and rows is:

```
<style>
td {border:solid blue 2pt}
th {border:solid red 4pt}
table {border:dotted green 3pt;border-collapse:collapse;background-color:white}
colgroup.a {background-color:yellow}
tr.three {background-color:white}
col.col1 {background-color:navajowhite}
tbody.second {background-color:powderblue}
colgroup.b {background-color:lime}
tr.two {background-color:pink}
th.grand {background-color:yellow}
</style>


<table>
<colgroup class="a" ><col class="col1"/><col class="col2" /></colgroup>
<colgroup class="b" span="3" />
<thead>. . .</thead>
<tbody class="first">
  <tr class="two">
    <th>Full</th><td>62</td><td>29</td><td>17</td><td>108</td>
  </tr>
</tbody>
<tbody class="second">
  <tr>
    <th>Affiliate</th><td>240</td><td>123</td><td>47</td><td>410</td>
  </tr>
  <tr class="three">
    <th>Total</th><td>302</td><td>152</td><td>64</td><th class="grand">518</th>
  </tr>
</tbody>
</table>
```

The result is:

W3C Membership

Type	Americas	Europe	Pacific	Total
Full	62	29	17	108
Affiliate	240	123	47	410
Total	302	152	64	**518**

Appendix A

References

There are some useful Web sites relevant to CSS:

1. http://www.w3.org/Style/CSS
   The W3C CSS Web Site which contains up-to-date links to anything relevant to CSS.
2. http://www.w3.org/TR/REC-CSS1
   Cascading Style Sheets, level 1 Recommendation, December 1996 (updated January 1999)
3. http://www.w3.org/TR/REC-CSS2/
   Cascading Style Sheets, level 2 CSS 2 Specification, May 1998.
4. Cascading Style Sheets: Designing for the Web (2nd Edition), Hakon Wium Lie and Bert Bos
   Addison Wesley, 1999.
5. http://www.w3.org/TR/CSS21/
   Cascading Style Sheets, level 2 revision 1 CSS 2.1 Specification, Working Draft, January 2003
6. http://www.w3.org/Style/CSS/current-work
   Cascading Style Sheets, level 3 CSS 3 Module Descriptions
7. http://www.blooberry.com/indexdot/css/
   Indexdot CSS Index
8. http://www.westciv.com/style_master/academy/css_tutorial/index.html
   A tutorial from Western Civilisation
9. http://www.westciv.com/style_master/house/CSS_gallery.html
   plus a Gallery of sites
10. http://www.htmlhelp.com/reference/css/
    Web Design Group Introduction
11. http://www.ericmeyeroncss.com/links/resources.html>
    Eric Meyer's list of Resources
12. http://www.glasshaus.com/samplechapters/1221/default.asp
    CSS Chapter from glasshaus's Web Professional's Handbook
13. http://www.glasshaus.com/bookInfo.asp?bookId=59
    Cascading style Sheets by Owen Briggs et al (2nd Edition) Autumn 2003
14. http://www.csszengarden.com/
    CSS Zen Garden Site

Appendix B: Quick Reference Guide

B.1 Syntax

a b c
a is followed by b is followed by c, in that order.
a | b
either a or b must occur
a || b
either a or b or both must occur, in any order
[a b]
brackets, used for grouping
a?
a is optional
a*
a is repeated 0 or more times
a+
a is repeated 1 or more times
a{1,4}
a is repeated 1 to 4 times.

B.2 Definitions

Block-level elements
an element which has a line break before and after (e.g. **<h1>**, **<p>**)
Replaced element
an element which is replaced by content pointed to from the element. E.g., **<img>**.

B.3 Properties

In each definition:

- The default value is shown in bold, or given separately
- Values apply to all elements unless otherwise stated
- Properties are inherited unless the property name is **emphasised**.
- Most properties can have a value **inherit**. These have been omitted to simplify the descriptions.

Name	Meaning
absolute-size	xx-small | x-small | small | **medium** | large | x-large | xx-large
absolute-unit	mm | cm | in | pt | pc
background	<background-color> || <background-image> || <background-repeat> || <background-attachment> || <background-position>
background-attachment	**scroll** | fixed
background-color	<color> | **transparent**
background-image	<url> | **none**
background-position	[<percentage> | <length>]{1,2} | [top | center | bottom] || [left | center | right]
background-repeat	**repeat** | repeat-x | repeat-y | no-repeat
border	<border-width> || <border-style> || <color>
border-bottom	<border-bottom-width> || <border-style> || <color>
border-bottom-width	thin | **medium** | thick | <length>
border-color | transparent	<color>{1,4}
border-left	<border-left-width> || <border-style> || <color>
border-left-width	thin | **medium** | thick | <length>
border-right	<border-right-width> || <border-style> || <color>
border-right-width	thin | **medium** | thick | <length>
border-style	**none** | dotted | dashed | solid | double | groove | ridge | inset | outset
border-top	<border-top-width> || <border-style> || <color>
border-top-width	thin | **medium** | thick | <length>
border-width	[thin | medium | thick | <length>]{1,4}
clear	**none** | left | right | both
color	<color>
color	<color-name> | <rgb>
color-name	aqua | black | blue | fuchsia | gray | green | lime | maroon | navy | olive | purple | red | silver | teal | white | yellow | orange
cursor	[<url> | auto | crosshair | default | pointer | move | e-resize | ne-resize | nw-resize | n-resize | progress | se-resize | sw-resize | s-resize | w-resize| text | wait | help
display	**inline** | block | list-item | inline-block | none
float	left | right | **none**
font	[<font-style> || <font-variant> || <font-weight>]? <font-size> [/ <line-height>]? <font-family>
font-family	[<family-name> | <generic-family>] [, [<family-name> | <generic-family>]]*
font-size	**<absolute-size>** | <relative-size> | <length> | <percentage>
font-style	**normal** | italic | oblique
font-variant	**normal** | small-caps
font-weight	**normal** | bold | bolder | lighter | 100 | 200 | 300 | 400 | 500 | 600 | 700 | 800 | 900

Name	Meaning
generic-family	serif | sans-serif | cursive | fantasy | monospace
height	<length> | <percentage> | **auto**
length	[+|-]?<number><unit>
letter-spacing	**normal** | <length>
line-height	**normal** | <number> | <length> | <percentage>
list-style	[<list-style-type> || <list-style-position> || <list-style-image>]
list-style-image	<url> | **none**
list-style-position	inside | **outside**
list-style-type	**disc** | circle | square | decimal | lower-roman | upper-roman | lower-alpha | upper-alpha | none
margin	[<length> | <percentage> | auto] {1,4}
margin-bottom	<length> | <percentage> | auto
margin-left	<length> | <percentage> | auto
margin-right	<length>| <percentage> | auto
margin-top	<length> | <percentage> | auto
number	<digit>+[.<digit>*]?
padding	[<length> | <percentage>] {1,4}
padding-bottom	<length> | <percentage>
padding-left	<length> | <percentage>
padding-right	<length> | <percentage>
padding-top	<length> | <percentage>
percentage	<number>%
relative-size	larger | smaller
relative-unit	em | ex | px
rgb	#<hex><hex><hex> | #<hex><hex><hex><hex><hex><hex> | rgb(<number>, <number>, <number>) | rgb(<percentage><percentage><percentage>)
text-align	left | right | center | justify
text-decoration	**none** | [underline || overline || line-through || blink]
text-indent	<length> | <percentage>
text-transform	capitalize | uppercase | lowercase | **none**
unit	<absolute-unit> | <relative-unit>
url	url(<quote>? <text> <quote>?)
vertical-align	**baseline** | sub | super | top | text-top | middle | bottom | text-bottom | <percentage>
white-space	normal | pre | nowrap| pre-wrap | pre-line
width	<length> | <percentage> | auto
word-spacing	**normal** | <length>

Appendix C: CSS Colour Keywords

The following table gives the set of keywords that are likely to be support by browsers. The minimal set are emphasised.

Colour Name	RGB Value	Colour Name	RGB Value
aliceblue	(240, 248, 255)	darkslategrey	(47, 79, 79)
antiquewhite	(250, 235, 215)	darkturquoise	(0, 206, 209)
aqua	(0, 255, 255)	darkviolet	(148, 0, 211)
aquamarine	(127, 255, 212)	deeppink	(255, 20, 147)
azure	(240, 255, 255)	deepskyblue	(0, 191, 255)
beige	(245, 245, 220)	dimgray	(105, 105, 105)
bisque	(255, 228, 196)	dimgrey	(105, 105, 105)
black	(0, 0, 0)	dodgerblue	(30, 144, 255)
blanchedalmond	(255, 235, 205)	firebrick	(178, 34, 34)
blue	(0, 0, 255)	floralwhite	(255, 250, 240)
blueviolet	(138, 43, 226)	forestgreen	(34, 139, 34)
brown	(165, 42, 42)	**fuchsia**	(255, 0, 255)
burlywood	(222, 184, 135)	gainsboro	(220, 220, 220)
cadetblue	(95, 158, 160)	ghostwhite	(248, 248, 255)
chartreuse	(127, 255, 0)	gold	(255, 215, 0)
chocolate	(210, 105, 30)	goldenrod	(218, 165, 32)
coral	(255, 127, 80)	**gray**	(128, 128, 128)
cornflowerblue	(100, 149, 237)	grey	(128, 128, 128)
cornsilk	(255, 248, 220)	**green**	(0, 128, 0)
crimson	(220, 20, 60)	greenyellow	(173, 255, 47)
cyan	(0, 255, 255)	honeydew	(240, 255, 240)
darkblue	(0, 0, 139)	hotpink	(255, 105, 180)
darkcyan	(0, 139, 139)	indianred	(205, 92, 92)
darkgoldenrod	(184, 134, 11)	indigo	(75, 0, 130)
darkgray	(169, 169, 169)	ivory	(255, 255, 240)
darkgreen	(0, 100, 0)	khaki	(240, 230, 140)
darkgrey	(169, 169, 169)	lavender	(230, 230, 250)
darkkhaki	(189, 183, 107)	lavenderblush	(255, 240, 245)
darkmagenta	(139, 0, 139)	lawngreen	(124, 252, 0)
darkolivegreen	(85, 107, 47)	lemonchiffon	(255, 250, 205)
darkorange	(255, 140, 0)	lightblue	(173, 216, 230)
darkorchid	(153, 50, 204)	lightcoral	(240, 128, 128)
darkred	(139, 0, 0)	lightcyan	(224, 255, 255)
darksalmon	(233, 150, 122)	lightgoldenrodyellow	(250, 250, 210)
darkseagreen	(143, 188, 143)	lightgray	(211, 211, 211)
darkslateblue	(72, 61, 139)	lightgreen	(144, 238, 144)
darkslategray	(47, 79, 79)	lightgrey	(211, 211, 211)

Colour Name	RGB Value	Colour Name	RGB Value
lightpink	(255, 182, 193)	paleturquoise	(175, 238, 238)
lightsalmon	(255, 160, 122)	palevioletred	(219, 112, 147)
lightseagreen	(32, 178, 170)	papayawhip	(255, 239, 213)
lightskyblue	(135, 206, 250)	peachpuff	(255, 218, 185)
lightslategray	(119, 136, 153)	peru	(205, 133, 63)
lightslategrey	(119, 136, 153)	pink	(255, 192, 203)
lightsteelblue	(176, 196, 222)	plum	(221, 160, 221)
lightyellow	(255, 255, 224)	powderblue	(176, 224, 230)
lime	(0, 255, 0)	**purple**	(128, 0, 128)
limegreen	(50, 205, 50)	**red**	(255, 0, 0)
linen	(250, 240, 230)	rosybrown	(188, 143, 143)
magenta	(255, 0, 255)	royalblue	(65, 105, 225)
maroon	(128, 0, 0)	saddlebrown	(139, 69, 19)
mediumaquamarine	(102, 205, 170)	salmon	(250, 128, 114)
mediumblue	(0, 0, 205)	sandybrown	(244, 164, 96)
mediumorchid	(186, 85, 211)	seagreen	(46, 139, 87)
mediumpurple	(147, 112, 219)	seashell	(255, 245, 238)
mediumseagreen	(60, 179, 113)	sienna	(160, 82, 45)
mediumslateblue	(123, 104, 238)	**silver**	(192, 192, 192)
mediumspringgreen	(0, 250, 154)	skyblue	(135, 206, 235)
mediumturquoise	(72, 209, 204)	slateblue	(106, 90, 205)
mediumvioletred	(199, 21, 133)	slategray	(112, 128, 144)
midnightblue	(25, 25, 112)	slategrey	(112, 128, 144)
mintcream	(245, 255, 250)	snow	(255, 250, 250)
mistyrose	(255, 228, 225)	springgreen	(0, 255, 127)
moccasin	(255, 228, 181)	steelblue	(70, 130, 180)
navajowhite	(255, 222, 173)	tan	(210, 180, 140)
navy	(0, 0, 128)	**teal**	(0, 128, 128)
oldlace	(253, 245, 230)	thistle	(216, 191, 216)
olive	(128, 128, 0)	tomato	(255, 99, 71)
olivedrab	(107, 142, 35)	turquoise	(64, 224, 208)
orange	(255, 165, 0)	violet	(238, 130, 238)
orangered	(255, 69, 0)	wheat	(245, 222, 179)
orchid	(218, 112, 214)	**white**	(255, 255, 255)
palegoldenrod	(238, 232, 170)	whitesmoke	(245, 245, 245)
palegreen	(152, 251, 152)	**yellow**	(255, 255, 0)
		yellowgreen	(154, 205, 50)