Color and Background Properties

- Color
- Background Color
- Background Image
- Background Repeat
- Background Attachment
- Background Position
- Background (Sets background images or color)

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 }
```

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 **backgroundcolor** set to **transparent** would appear with a red background.

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>
```

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 }
```

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.

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:

xNotation 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%

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.

Font Properties

- Font Family (Sets typeface)
- Font Style (Italicises text)
- Font Variant (Mostly upper case)
- Font Weight (Sets thickness of type)
- Font Size (Sets size of text)
- Font (Shorthand)

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

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.

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.

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 }
```

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.

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.

Box Properties

- Introduction
- Margin Setting
- Padding Setting
- Border Setting
- Border Color
- Border Style
- Setting All Border Properties Together
- Width and Height of Images
- Float
- Clear

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

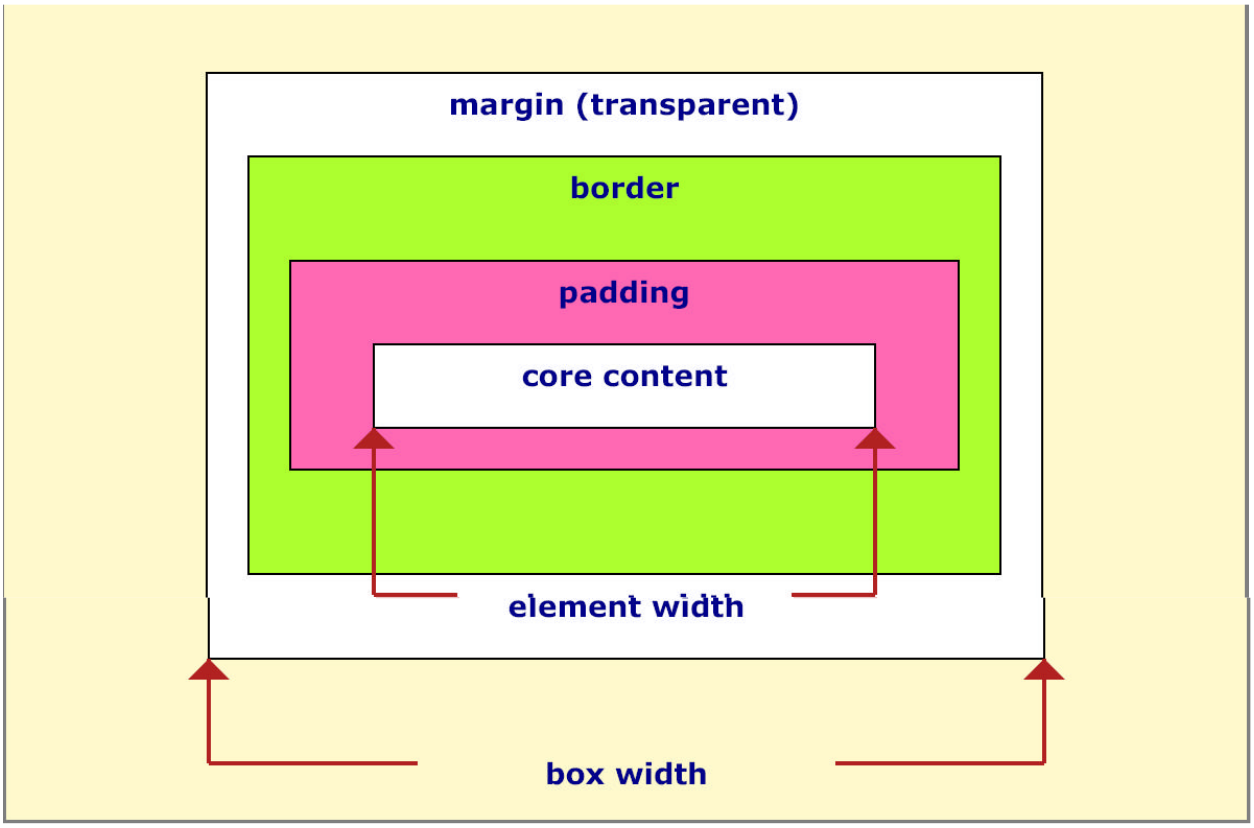

Figure 7.1: Box Model

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.

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.

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.

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>]{1,4}
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.

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.

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.

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>
```

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.

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.

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.

Structure and Control

- Introduction
- Classes
- Identifiers
- Contextual Selectors
- Grouping
- Universal Selector
- Attribute Selectors
- Comments
- The Elements div and span

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.

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

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}
```

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 }
```

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 }
```

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.

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.

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 */
```

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.

Pseudo-classes and Pseudo-elements

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 }
```

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.

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.

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

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