**Values and Units**

**Keywords, Strings, and Other Text Values**

Everything in a stylesheet is text, but there are certain value types that directly represent strings of text as opposed to, say, numbers or colors. Included in this category are URLs and, interestingly enough, images.

**Keywords:**

For those times when a value needs to be described with a word of some kind, there are keywords. A very common example is the keyword none, which is distinct from 0 (zero). Thus, to remove the underline from links in an HTML document, you would write:

a:link, a:visited {text-decoration:*none*;}

If a property accepts keywords, then its keywords will be defined only for the scope of that property. If two properties use the same word as a keyword, the behavior of the keyword for one property will not necessarily be shared with the other. As an example, normal, as defined for letter-spacing, means something very different than the normal defined for font-style.

**Global keywords**

CSS3 defines three “global” keywords that are accepted by every property in the specification: inherit, initial, and unset.

**Inherit:**

The keyword inherit makes the value of a property on an element the same as the value of that property on its parent element. In other words, it forces inheritance to occur even in situations where it would not normally operate. In many cases, you don’t need to specify inheritance, since many properties inherit naturally. Nevertheless, inherit can still be very useful.

For example, consider the following styles and markup:

#toolbar {background: blue; color: white;}

<div id="toolbar">

    <a href="one.html">One</a> |

    <a href="two.html">Two</a> |

    <a href="three.html">Three</a>

</div>

The div itself will have a blue background and a white foreground, but the links will be styled according to the browser’s preference settings. They’ll most likely end up as blue text on a blue background, with white vertical bars between them.

You could write a rule that explicitly sets the links in the “toolbar” to be white, but you can make things a little more robust by using inherit. You just add the following rule to the stylesheet:

#toolbar a {color: *inherit*;}

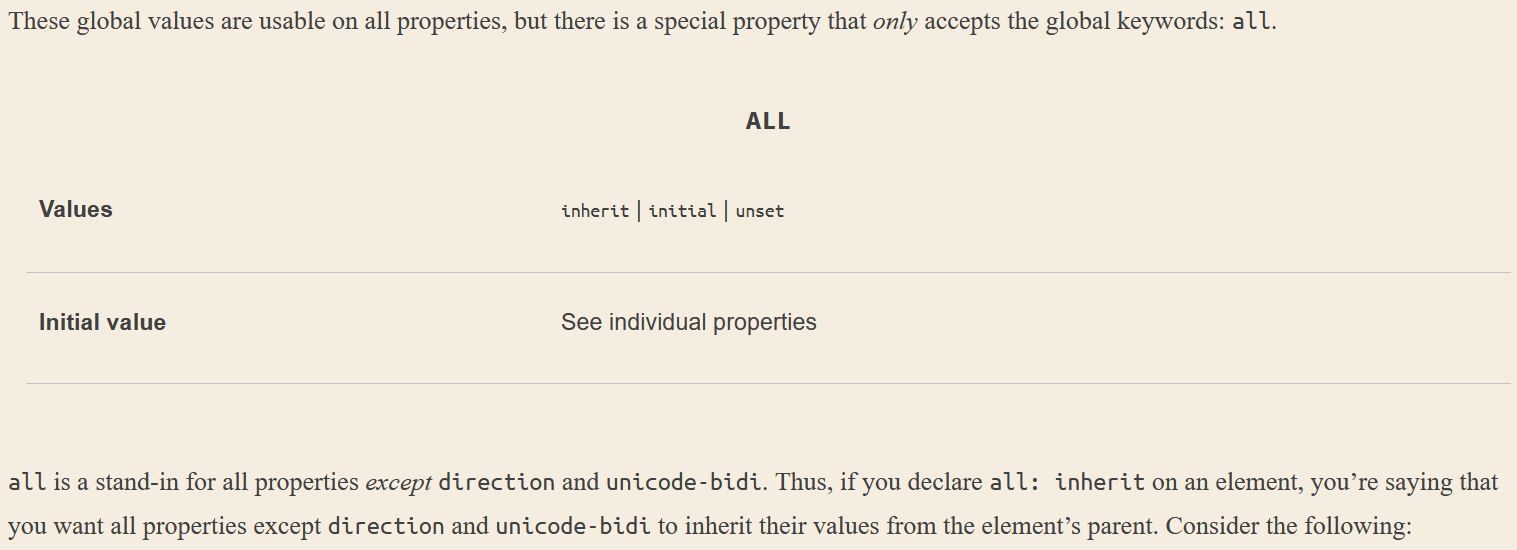
**Initial**

The keyword initial sets the value of a property to the defined initial value, which in a way means it “resets” the value. For example, the default value of font-weight is normal. Thus, declaring font-weight: initial is the same as declaring font-weight: normal.

This might seem a little bit silly until you consider that not all values have explicitly defined initial values. For example, the initial value for color is “depends on user agent.” That’s not a funky keyword you should type! What it means is that the default value of color depends on things like the preferences settings in a browser. While almost nobody changes the default text color setting from black, someone might set it to a dark gray or even a bright red. By declaring color: initial;, you’re telling the browser to set the color of the element to whatever the user’s default color is set to be.

**Unset**

The keyword unset acts as a universal stand-in for both inherit and initial. If the property is inherited, then unset has the same effect as if inherit was used. If the property is not inherited, then unset has the same effect as if initial was used.



For Example:

section {color: white; background: black; font-weight: *bold*;}

#example {all: *inherit*;}

    <section>

        <div id="example">This is a div.</div>

    </section>

You might think this causes the div element to inherit the values of color, background, and font-weight from the section element. And it does do that, yes—but it will *also* force inheritance of the values of *every single other property in CSS* (minus the two exceptions) from the section element.

Maybe that’s what you want, in which case, great. But if you just want to inherit the property values you wrote out for the section element, then the CSS would need to look more like this:

**Strings**

A string value is an arbitrary sequence of characters wrapped in either single or double quotes, and is represented in value definitions with <string>. For example:

'Hello World'

If you have some reason to include a newline in your string value, you can do that by escaping the newline itself. CSS will then remove it, making things as if it had never been there. Thus, the following two string values are identical from a CSS point of view:

"This is the right place \

for a newline."

"This is the right place for a newline."

If, on the other hand, you actually want a string value that includes a newline character, then use the Unicode reference \A where you want the newline to occur:

"This is a better place \Afor a newline." */\* String with a newline character\*/*

**URLs**

If you’ve written web pages, you’re almost certainly familiar with URLs (or, as in CSS2.1, URIs). Whenever you need to refer to one—as in the @import statement, which is used when importing an external stylesheet—the general format is:

url(protocol://server/pathname)

**Absolute vs Relative URL:**

Absolute URL: Defines the absolute URL (location) of the file being referred in web-space. It can be on any server.

Relative URL: Defines the location of the file relative to the document that uses it. The file has to be on the same server as the page that contains the URL.

Note: In CSS, relative URLs are relative to the stylesheet itself, not to the HTML document that uses the stylesheet.

Note that there cannot be a space between the url and the opening parenthesis:

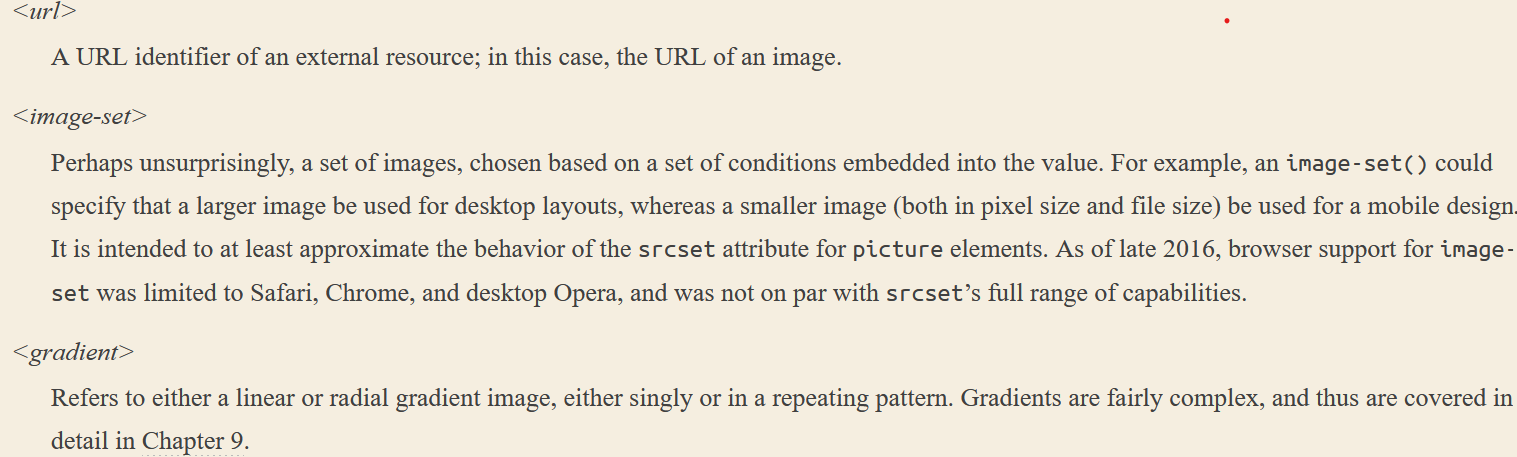
body {background: url(http://www.pix.web/picture1.jpg);}   */\* correct \*/*

body {background: url (images/picture2.jpg);}*/\* INCORRECT - invalid declaration and thus ignored \*/*

**Images**

An *image value* is a reference to an image, as you might have guessed. Its syntax representation is *<image>*.

At the most basic level of support, which is to say the one every CSS engine on the planet would understand, an *<image>* value is a *<url>* value. In more advanced user agents, *<image>* stands for one of the following:

****

**Identifiers**

There are a few properties that accept an identifier value, which is a user-defined identifier of some kind; the most common example is generated list counters. They are represented in the value syntax as <identifier>. Identifiers themselves are words, and are case-sensitive; thus, myID and MyID are, as far as CSS is concerned, completely distinct and unrelated to each other. In cases where a property accepts both an identifier and one or more keywords, the author should take care to never define an identifier identical to a valid keyword.

**Numbers and Percentages:**

**Integers:** An integer value is about as simple as it gets: one or more numbers, optionally prefixed by a + or − sign to indicate a positive or negative value. That’s it. Integer values are represented in value syntax as <integer>

**Numbers:** A number value is either an <integer> or a real number, which is to say an integer followed by a dot and then some number of following integers. Additionally, it can be prefixed by either + or − to indicate positive or negative values. Number values are represented in value syntax as <number>.

**Percentages:** A percentage value is a <number> followed by a percentage sign (%), and is represented in value syntax as <percentage>. Percentage values are always relative to another value, which can be anything—the value of another property of the same element, a value inherited from the parent element, or a value of an ancestor element.

**Fractions**: A fraction value (or flex value) is a <number> followed by the label fr. Thus, one fractional unit is 1fr. This is a concept introduced by Grid Layout, and is used to divide up fractions of the unconstrained space in a layout.

**Distances**

These length units are divided into two types: *absolute length units* and *relative length units*.

**Absolute Length Units**

* Inches: found on a ruler.
* Centimeters: 2.54 cm make an inch
* Millimeters: 10 millimeters to a centimeter.
* Quarter-millimeters: 40Q units in a centimeter
* Points: 72 points to an inch.
* Picas: A pica is equivalent to 12 points, which means there are 6 picas to an inch.
* Pixels: A pixel is a small box on screen, but CSS defines pixels more abstractly. In CSS terms, a pixel is defined to be the size required to yield 96 pixels per inch. Many user agents ignore this definition in favor of simply addressing the pixels on the screen. Scaling factors are brought into play when page zooming or printing, where an element 100px wide can be rendered more than 100 device dots wide. In general, if you declare something like font-size: 18px, a web browser will almost certainly use actual pixels on your screen—after all, they’re already there—but with other display devices, like printers, the user agent will have to rescale pixel lengths to something more sensible. In other words, the printing code has to figure out how many dots there are in a pixel.

**Pixel Theory**

In its discussion of pixels, the CSS specification recommends that, in cases where a display’s resolution density is significantly different than 96 pixels per inch (ppi), user agents should scale pixel measurements to a “reference pixel.” CSS2 recommended 90 ppi as the reference pixel, but CSS2.1 and CSS3 recommend 96 ppi. The most common example is a printer, which has dots instead of pixels, and which has a lot more dots per inch than 96! In printing web content, then, it may assume 96 pixels per inch and scale its output accordingly.

If a display’s resolution is set to 1,024 pixels wide by 768 pixels tall, its screen size is exactly 10 2/3 inches wide by 8 inches tall, and the screen is filled entirely by the display pixels, then each pixel will be 1/96 of an inch wide and tall. As you might guess, this scenario is a fairly rare occurrence. So, on most displays, the actual number of pixels per inch (ppi) is higher than 96—sometimes much higher. The Retina display on an iPhone 4S, for example, is 326 ppi; the display on the iPad 264 ppi.

**Resolution Units (only used in media queries)**

**Dots per inch (dpi):** The number of display dots per linear inch.

**Dots per centimeter (dpcm):** The number of display dots per linear centimeter.

**Dots per pixel unit (dppx):** The number of display dots per CSS px unit. As of CSS3, 1dppx is equivalent to 96dpi because CSS defines pixel units at that ratio.

**Relative Length Units**

Relative units are so called because they are measured in relation to other things. The actual (or absolute) distance they measure can change due to factors beyond their control, such as screen resolution, the width of the viewing area, the user’s preference settings, and a whole host of other things.

**Em and Ex Units**

In CSS, one “em” is defined to be the value of font-size for a given font. If the font-size of an element is 14 pixels, then for that element, 1em is equal to 14 pixels. When setting the size of the font, on the other hand, the value of em is relative to the font-size of the parent element. In theory, one em is equal to the width of a lowercase m in the font-used (old typographical term- though that cannot be assured in CSS).

Ex on the other hand, refers to the height of a lowercase x in the font being used. Therefore, if you have two paragraphs in which the text is 24 points in size, but each paragraph uses a different font, then the value of ex could be different for each paragraph.

**REM UNIT**

rem is always calculated using the root element. In HTML, that’s the html element. Thus, declaring any element to have font-size: 1rem; is setting it to have the same font-size value as the root element of the document. This will usually be the user’s default font-size, unless changed by the user.

**CH UNIT**

Equal to the advance measure of the “0” (ZERO, U+0030) glyph found in the font used to render it.

**VIEWPORT-RELATIVE UNITS**

Following three viewport-relative size units are there. These are calculated with respect to the size of the viewport—browser window, printable area, mobile device display, etc.

Viewport width unit (vw): This unit is calculated with respect to the viewport’s width, which is divided by 100. Therefore, if the viewport is 937 pixels wide, 1vw is equal to 9.37px. If the viewport’s width changes, say by dragging the browser window wider or more narrow, the value of vw changes along with it.

Viewport height unit (vh): Same as above but wit viewport’s height.

Viewport minimum unit (vmin) : This unit is 1/100 of the viewport’s width or height whichever is lesser.

Viewport maximum unit (vmax): This unit is 1/100 of the viewport’s width or height whichever is greater.