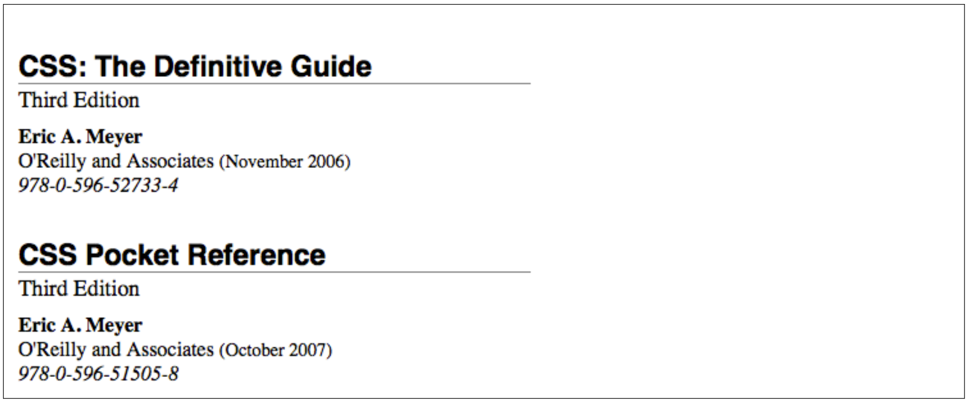

CSS: The Definitive Guide

Third Edition

Eric A. Meyer
O'Reilly and Associates (November 2006)
978-0-596-52733-4

CSS Pocket Reference

Third Edition

Eric A. Meyer
O'Reilly and Associates (October 2007)
978-0-596-51505-8

Figure 1-3. Styled display of an XML document

Before learning how to write CSS in detail, we need to look at how one can associate CSS with a document. After all, without tying the two together, there's no way for the CSS to affect the document. We'll explore this in an HTML setting since it's the most familiar.

Bringing CSS and HTML Together

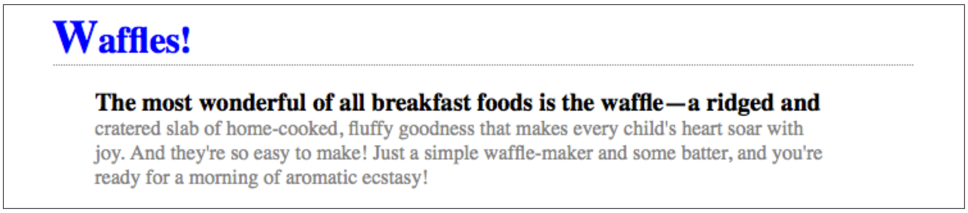
I've mentioned that HTML documents have an inherent structure, and that's a point worth repeating. In fact, that's part of the problem with web pages of old: too many of us forgot that documents are supposed to have an internal structure, which is altogether different than a visual structure. In our rush to create the coolest-looking pages on the web, we bent, warped, and generally ignored the idea that pages should contain information with some structural meaning.

That structure is an inherent part of the relationship between HTML and CSS; without it, there couldn't be a relationship at all. To understand it better, let's look at an example HTML document and break it down by pieces:

```
<html>
<head>
<title>Eric's World of Waffles</title>
<meta http-equiv="content-type" content="text/html; charset=utf-8">
<link rel="stylesheet" type="text/css" href="sheet1.css" media="all">
<style type="text/css">
/* These are my styles! Yay! */
@import url(sheet2.css);
</style>
</head>
<body>
<h1>Waffles!</h1>
<p style="color: gray;">The most wonderful of all breakfast foods is
the waffle—a ridged and cratered slab of home-cooked, fluffy goodness
```

```
that makes every child's heart soar with joy. And they're so easy to make!
Just a simple waffle-maker and some batter, and you're ready for a morning
of aromatic ecstasy!
</p>
</body>
</html>
```

The result of this markup and the applied styles is shown in Figure 1-4.


Waffles!

The most wonderful of all breakfast foods is the waffle—a ridged and cratered slab of home-cooked, fluffy goodness that makes every child's heart soar with joy. And they're so easy to make! Just a simple waffle-maker and some batter, and you're ready for a morning of aromatic ecstasy!

Figure 1-4. A simple document

Now, let's examine the various ways this document connects to CSS.

The link Tag

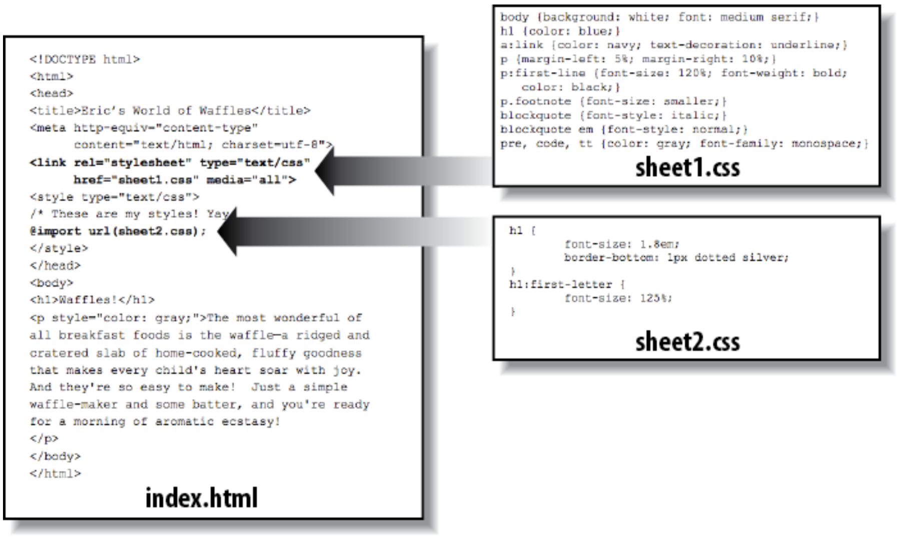
First, consider the use of the `link` tag:

```
<link rel="stylesheet" type="text/css" href="sheet1.css" media="all">
```

The `link` tag is a little-regarded but nonetheless perfectly valid tag that has been hanging around the HTML specification for years, just waiting to be put to good use. Its basic purpose is to allow HTML authors to associate other documents with the document containing the `link` tag. CSS uses it to link stylesheets to the document; in Figure 1-5, a stylesheet called *sheet1.css* is linked to the document.

These stylesheets, which are not part of the HTML document but are still used by it, are referred to as *external stylesheets*. This is because they're stylesheets that are external to the HTML document. (Go figure.)

To successfully load an external stylesheet, `link` must be placed inside the `head` element but may not be placed inside any other element. This will cause the web browser to locate and load the stylesheet and use whatever styles it contains to render the HTML document in the manner shown in Figure 1-5. Also shown in Figure 1-5 is the loading of the external *sheet2.css* via the `@import` declaration. Imports must be placed at the beginning of the stylesheet that contains them, but they are otherwise unconstrained.



```
<!DOCTYPE html>
<html>
<head>
<title>Eric's World of Waffles</title>
<meta http-equiv="content-type"
      content="text/html; charset=utf-8">
<link rel="stylesheet" type="text/css"
      href="sheet1.css" media="all">
<style type="text/css">
/* These are my styles! Yay
@import url(sheet2.css);
</style>
</head>
<body>
<h1>Waffles!</h1>
<p style="color: gray;">The most wonderful of
all breakfast foods is the waffle—a ridged and
cratered slab of home-cooked, fluffy goodness
that makes every child's heart soar with joy.
And they're so easy to make!  Just a simple
waffle-maker and some batter, and you're ready
for a morning of aromatic ecstasy!
</p>
</body>
</html>
```

```
body {background: white; font: medium serif;}
h1 {color: blue;}
a:link {color: navy; text-decoration: underline;}
p {margin-left: 5%; margin-right: 10%;}
p:first-line {font-size: 120%; font-weight: bold;
   color: black;}
p.footnote {font-size: smaller;}
blockquote {font-style: italic;}
blockquote em {font-style: normal;}
pre, code, tt {color: gray; font-family: monospace;}
```

```
h1 {
        font-size: 1.8em;
        border-bottom: 1px dotted silver;
}
h1:first-letter {
        font-size: 125%;
}
```

Figure 1-5. A representation of how external stylesheets are applied to documents

And what is the format of an external stylesheet? It's a list of rules, just like those we saw in the previous section and in the example HTML document; but in this case, the rules are saved into their own file. Just remember that no HTML or any other markup language can be included in the stylesheet—only style rules. Here are the contents of an external stylesheet:

```
h1 {color: red;}
h2 {color: maroon; background: white;}
h3 {color: white; background: black;
 font: medium Helvetica;}
```

That's all there is to it—no HTML markup or comments at all, just plain-and-simple style declarations. These are saved into a plain-text file and are usually given an extension of *.css*, as in *sheet1.css*.

An external stylesheet cannot contain any document markup at all, only CSS rules and CSS comments, both of which are explained later in the chapter. The presence of markup in an external stylesheet can cause some or all of it to be ignored.

The filename extension is not required, but some older browsers won't recognize the file as containing a stylesheet unless it actually ends with *.css*, even if you *do* include the correct `type` of `text/css` in the `link` element. In fact, some web servers won't hand over a file as `text/css` unless its filename ends with *.css*, though that can usually be fixed by changing the server's configuration files.

Attributes

For the rest of the `link` tag, the attributes and values are fairly straightforward. `rel` stands for "relation," and in this case, the relation is `stylesheet`. The attribute `type` is always set to `text/css`. This value describes the type of data that will be loaded using the `link` tag. That way, the web browser knows that the stylesheet is a CSS stylesheet, a fact that will determine how the browser deals with the data it imports. After all, there may be other style languages used in the future, so it's important to declare which language you're using.

Next, we find the `href` attribute. The value of this attribute is the URL of your stylesheet. This URL can be either absolute or relative, depending on what works for you. In our example, the URL is relative. It just as easily could have been something like *http://meyerweb.com/sheet1.css*.

Finally, we have a `media` attribute. The value of this attribute is one or more *media descriptors*, which are rules regarding media types and the features of those media, with each rule separated by a comma. Thus, for example, you can use a linked stylesheet in both screen and projection media:

```
<link rel="stylesheet" type="text/css" href="visual-sheet.css"
   media="screen, projection">
```

Media descriptors can get quite complicated, and are explained in detail later in the chapter. For now, we'll stick with the basic media types shown.

Note that there can be more than one linked stylesheet associated with a document. In these cases, only those `link` tags with a `rel` of `stylesheet` will be used in the initial display of the document. Thus, if you wanted to link two stylesheets named *basic.css* and *splash.css*, it would look like this:

```
<link rel="stylesheet" type="text/css" href="basic.css">
<link rel="stylesheet" type="text/css" href="splash.css">
```

This will cause the browser to load both stylesheets, combine the rules from each, and apply them all to the document. For example:

```
<link rel="stylesheet" type="text/css" href="basic.css">
<link rel="stylesheet" type="text/css" href="splash.css">

<p class="a1">This paragraph will be gray only if styles from the
stylesheet 'basic.css' are applied.</p>
<p class="b1">This paragraph will be gray only if styles from the
stylesheet 'splash.css' are applied.</p>
```

The one attribute that is not in this example markup, but could be, is the `title` attribute. This attribute is not often used, but it could become important in the future and, if used improperly, can have unexpected effects. Why? We will explore that in the next section.

Alternate stylesheets

It's also possible to define *alternate stylesheets*. These are defined by making the value of the `rel` attribute `alternate stylesheet`, and they are used in document presentation only if selected by the user.

Should a browser be able to use alternate stylesheets, it will use the values of the `link` element's `title` attributes to generate a list of style alternatives. So you could write the following:

```
<link rel="stylesheet" type="text/css"
   href="sheet1.css" title="Default">
<link rel="alternate stylesheet" type="text/css"
   href="bigtext.css" title="Big Text">
<link rel="alternate stylesheet" type="text/css"
   href="zany.css" title="Crazy colors!">
```

Users could then pick the style they want to use, and the browser would switch from the first one, labeled "Default" in this case, to whichever the user picked. Figure 1-6 shows one way in which this selection mechanism might be accomplished (and in fact was, early in the resurgence of CSS).

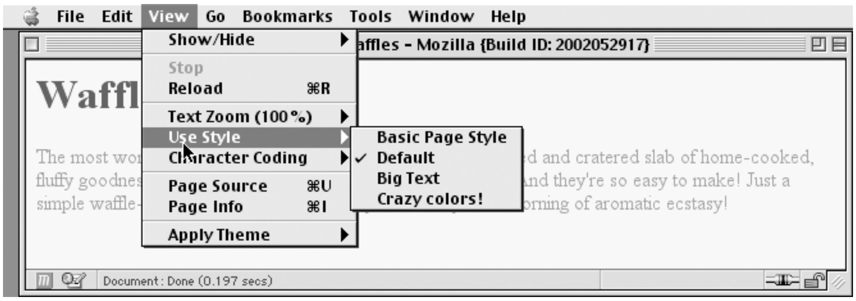

File Edit View Go Bookmarks Tools Window Help
affles - Mozilla {Build ID: 2002052917}
Waffl
Show/Hide
Stop
Reload ⌘R
Text Zoom (100%)
Use Style
Character Coding
Page Source ⌘U
Page Info ⌘I
Apply Theme
Basic Page Style
✓ Default
Big Text
Crazy colors!
The most wo
fluffy goodnes
simple waffle-
d and cratered slab of home-cooked,
nd they're so easy to make! Just a
orning of aromatic ecstasy!
Document: Done (0.197 secs)

Figure 1-6. A browser offering alternate stylesheet selection

As of late 2016, alternate stylesheets were supported in most Gecko-based browsers like Firefox, and in Opera. They could be supported in the Internet Explorer family through the use of JavaScript but are not natively supported by those browsers. The WebKit family did not support selecting alternate stylesheets. Compare this to the age of the browser shown in Figure 1-6--it's almost shocking.

It is also possible to group alternate stylesheets together by giving them the same `title` value. Thus, you make it possible for the user to pick a different presentation for your site in both screen and print media:

```
<link rel="stylesheet" type="text/css"
   href="sheet1.css" title="Default" media="screen">
<link rel="stylesheet" type="text/css"
   href="print-sheet1.css" title="Default" media="print">
<link rel="alternate stylesheet" type="text/css"
   href="bigtext.css" title="Big Text" media="screen">
<link rel="alternate stylesheet" type="text/css"
   href="print-bigtext.css" title="Big Text" media="print">
```

If a user selects "Big Text" from the alternate stylesheet selection mechanism in a conforming user agent, then *bigtext.css* will be used to style the document in the screen medium, and *print-bigtext.css* will be used in the print medium. Neither *sheet1.css* nor *print-sheet1.css* will be used in any medium.

Why is that? Because if you give a `link` with a `rel` of `stylesheet` a title, then you are designating that stylesheet as a *preferred stylesheet*. This means that its use is preferred to alternate stylesheets, and it will be used when the document is first displayed. Once you select an alternate stylesheet, however, the preferred stylesheet will *not* be used.

Furthermore, if you designate a number of stylesheets as preferred, then all but one of them will be ignored. Consider the following code example:

```
<link rel="stylesheet" type="text/css"
   href="sheet1.css" title="Default Layout">
<link rel="stylesheet" type="text/css"
   href="sheet2.css" title="Default Text Sizes">
<link rel="stylesheet" type="text/css"
   href="sheet3.css" title="Default Colors">
```

All three `link` elements now refer to preferred stylesheets, thanks to the presence of a `title` attribute on all three, but only one of them will actually be used in that manner. The other two will be ignored completely. Which two? There's no way to be certain, as HTML doesn't provide a method of determining which preferred stylesheets should be ignored and which should be used.

If you don't give a stylesheet a title, then it becomes a *persistent stylesheet* and is always used in the display of the document. Often, this is exactly what an author wants.

The style Element

The `style` element is one way to include a stylesheet, and it appears in the document itself:

```
<style type="text/css">...</style>
```

`style` should always use the attribute `type`; in the case of a CSS document, the correct value is `"text/css"`, just as it was with the `link` element.

The `style` element should always start with `<style type="text/css">`, as shown in the preceding example. This is followed by one or more styles and is finished with a closing `</style>` tag. It is also possible to give the `style` element a `media` attribute, which functions in the same manner as previously discussed for linked stylesheets.

The styles between the opening and closing `style` tags are referred to as the *document stylesheet* or the *embedded stylesheet* (because this kind of stylesheet is embedded within the document). It will contain many of the styles that will apply to the document, but it can also contain multiple links to external stylesheets using the `@import` directive.

The @import Directive

Now we'll discuss the stuff that is found inside the `style` tag. First, we have something very similar to `link`: the `@import` directive:

```
@import url(sheet2.css);
```

Just like `link`, `@import` can be used to direct the web browser to load an external stylesheet and use its styles in the rendering of the HTML document. The only major difference is in the syntax and placement of the command. As you can see, `@import` is found inside the `style` container. It must be placed before the other CSS rules or it won't work at all. Consider this example:

```
<style type="text/css">
@import url(styles.css); /* @import comes first */
h1 {color: gray;}
</style>
```

Like `link`, there can be more than one `@import` statement in a document. Unlike `link`, however, the stylesheets of every `@import` directive will be loaded and used; there is no way to designate alternate stylesheets with `@import`. So, given the following markup:

```
@import url(sheet2.css);
@import url(blueworld.css);
@import url(zany.css);
```

all three external stylesheets will be loaded, and all of their style rules will be used in the display of the document.

As with `link`, you can restrict imported stylesheets to one or more media by providing media descriptors after the stylesheet's URL:

```
@import url(sheet2.css) all;
@import url(blueworld.css) screen;
@import url(zany.css) projection, print;
```

As noted in "The link Tag" on page 8, media descriptors can get quite complicated, and are explained in detail in Chapter 20, *Media-Dependent Styles*.

`@import` can be highly useful if you have an external stylesheet that needs to use the styles found in other external stylesheets. Since external stylesheets cannot contain any document markup, the `link` element can't be used—but `@import` can. Therefore, you might have an external stylesheet that contains the following:

```
@import url(http://example.org/library/layout.css);
@import url(basic-text.css);
@import url(printer.css) print;
body {color: red;}
h1 {color: blue;}
```

Well, maybe not those exact styles, but hopefully you get the idea. Note the use of both absolute and relative URLs in the previous example. Either URL form can be used, just as with `link`.

Note also that the `@import` directives appear at the beginning of the stylesheet, as they did in the example document. CSS requires the `@import` directive to come before any other rules in a stylesheet. An `@import` that comes after other rules (e.g., `body {color: red;}`) will be ignored by conforming user agents.

Older versions of Internet Explorer for Windows do not ignore any `@import` directive, even those that come after other rules. Since other browsers do ignore improperly placed `@import` directives, it is easy to mistakenly place the `@import` directive incorrectly and thus alter the display in other browsers.

HTTP Linking

There is another, far more obscure way to associate CSS with a document: you can link the two via HTTP headers.

Under Apache, this can be accomplished by adding a reference to the CSS file in a *.htaccess* file. For example:

```
Header add Link "</ui/testing.css>;rel=stylesheet;type=text/css"
```

This will cause supporting browsers to associate the referenced stylesheet with any documents served from under that *.htaccess* file. The browser will then treat it as if it were a linked stylesheet. Alternatively, and probably more efficiently, you can add an equivalent rule to the server's *httpd.conf* file:

```
<Directory /path/to/ /public/html/directory>
Header add Link "</ui/testing.css>;rel=stylesheet;type=text/css"
</Directory>
```

The effect is exactly the same in supporting browsers. The only difference is in where you declare the linking.

You probably noticed the use of the term "supporting browsers." As of late 2017, the widely used browsers that support HTTP linking of stylesheets are the Firefox family and Opera. That restricts this technique mostly to development environments based on one of those browsers. In that situation, you can use HTTP linking on the test server to mark when you're on the development site as opposed to the public site. It's also an interesting way to hide styles from the WebKit and Internet Explorer families, assuming you have a reason to do so.

There are equivalents to this technique in common scripting languages such as PHP and IIS, both of which allow the author to emit HTTP headers. It's also possible to use such languages to explicitly write a `link` element into the document based on the server offering up the document. This is a more robust approach in terms of browser support: every browser supports the `link` element.

Inline Styles

For cases where you want to just assign a few styles to one individual element, without the need for embedded or external stylesheets, employ the HTML attribute `style` to set an inline style:

```
<p style="color: gray;">The most wonderful of all breakfast foods is
the waffle—a ridged and cratered slab of home-cooked, fluffy goodness...
</p>
```

The `style` attribute can be associated with any HTML tag whatsoever, except for those tags that are found outside of `body` (`head` or `title`, for instance).

The syntax of a `style` attribute is fairly ordinary. In fact, it looks very much like the declarations found in the `style` container, except here the curly braces are replaced by double quotation marks. So `<p style="color: maroon; background: yellow;">` will set the text color to be maroon and the background to be yellow *for that paragraph only*. No other part of the document will be affected by this declaration.

Note that you can only place a declaration block, not an entire stylesheet, inside an inline `style` attribute. Therefore, you can't put an `@import` into a `style` attribute, nor can you include any complete rules. The only thing you can put into the value of a `style` attribute is what might go between the curly braces of a rule.

Use of the `style` attribute is not generally recommended. Indeed, it is very unlikely to appear in XML languages other than HTML. Many of the primary advantages of CSS —the ability to organize centralized styles that control an entire document's appearance or the appearance of all documents on a web server—are negated when you