DZone Refcardz

CONTENTS INCLUDE:

Core CSS: Part I

By Molly E. Holzschlag

ABOUT CSS

As Cascading Style Sheets mature as a language of design and a tool of Web site and application management, a deep understanding of how the language really works is essential. However, most people have learned CSS the same way they've learned HTML—by viewing source, copying template codes, reading books and articles. While this "bootstrap" method of learning is often the best way to find great techniques, it may not be the best for knowing how to manage, debug, customize and even advance those techniques.

What our training hasn't necessarily provided are the core concepts within CSS. This is why the Core CSS series may contain simple examples of things you already know. You'll just get to know them better here! In this foundational reference card, you'll find not only a bit of history and rationale for use, rule structure and syntax, but also a thorough resource as to the Cascade, inheritance and specificity—core principles of CSS that will expand and strengthen your professional ability to work with CSS.

CSS RATIONALE AND USE

The idea behind CSS is not a new one. We've seen the separation of presentation before in desktop publishing, where master style sheets can be created to control the layout, typefaces and colors used in a given design. Cascading Style Sheets were conceived to do exactly that: Remove the style from the document and place it separately from the code to be styled.

The benefits, when used carefully, can be outstanding. Some benefits of using style sheets include:

- **Design flexibility**
  - More image options
  - Better typographic control
  - Far more flexible layout options
  - Print design support
  - Handheld device design support
- **Easier site maintenance**
  - One style sheet, infinite pages
  - Design changes are very easy
  - Changes can be made quickly
  - Reduces time to launch
- **Measurable returns**
  - Faster loading documents
  - Far smaller documents
  - User experience improves
  - Accessibility improves
  - SEO (search engine optimization) improves

The first proposal for CSS was made by Håkom Wium Lie, now CTO of Opera Software. He worked with Bert Bos to co-author the first CSS specification, which believe it or not, became a recommendation in 1996! By 1998 CSS 2.0 brought us richer options, as we find later in advancing versions CSS 2.1 and CSS 3.0.

Version	Date	Implementations
CSS 1.0	First proposed 1994, First specification in 1996	Still flawed CSS 1 portions in all CSS browsers
CSS 2.0	1998	No full implementation
CSS 2.1	Not yet published as a complete specification	Some close to complete implementations
CSS 3.0 (Modular)	Certain modules are ahead of others in development	Some CSS 3.0 features are implemented in versions of WebKit, Mozilla and Opera browsers

Table 1. CSS Versions, Publication Dates and Implementation

As CSS evolves, we find it becoming more and more important for not only visual designers in terms of managing the esthetics of the site, but technologists working on large web sites or looking to create rock-solid applications.

A Separate Piece

The term for sites designed using table-based layouts and HTML presentation rather than CSS are referred to as being authored in *presentational HTML*. This means that the presentation (the design, style, and layout) isn't separated from the markup (content with basic formatting).

Consider this header, which contains elements and attributes that define presentation:

```
<h1><font size="5" color="red" face="Arial,
Helvetica, sans-serif">Welcome!</font></h1>
```

Using presentational HTML, every time you need a new font size, color or face it has to be explicitly defined in that document. And then redefined. In CSS, we can set up presentation and have →

Core CSS: Part I

www.dzone.com

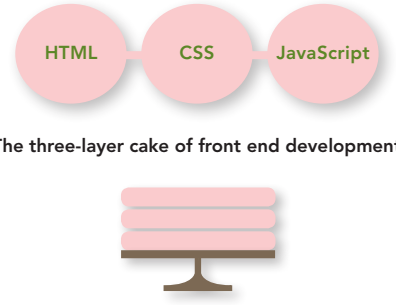
A Separate Piece, continued

it used not only multiple times within a given document, but across literally millions of documents. So, if I instead had a separate style sheet with this rule:

```
h1 {font-size: 80%; color: red; font-family:
Arial, Helvetica, sans-serif;}
```

Every single document I want to apply this style to can be attached to this sheet. Then, if I need to change a million documents with an `h1` of red to an `h1` of green, I simply go to the one style sheet, change the color in one location, one time, save the style and instantly all the documents connected to the sheet will now have green `h1` headers (this is the point where you tell the boss it's going to take all day to make the change, grab your stuff, and head to the beach)!

So from the get-go the principles of CSS suggest that we gain many benefits from separating the technology layers that make up front end Web development: Document, presentation, behavior (Figure 1).



The three-layer cake of front end development

Figure 1. Document (markup+content); Presentation (CSS); Behavior (JavaScript)

Of course, just because we "bake" our three-layered cake separately, it all has to come together and just be "cake" at some point! This is where standards-based design and best practices come significantly into play, and of course those skills rely in turn on quality learning in terms of both the languages and Web browsers we use.

Learning and Implementation Curves

However, there's been difficulty along the road to adopting, learning and managing CSS. This difficulty is due to a number of influences, but two of the major concerns are a steep learning curve and lack of consistent browser implementation of specifications.

Especially of challenge for visual designers is that CSS nomenclature and concepts are programmatic rather than graphic-design oriented (Table 2) and there are no tools that can fully replace designer understanding. A fun, if not frightening metaphor would be to ask graphic designers if they code in postscript, the underlying language for vector drawing in Adobe products.

CSS Term	Graphic Design	Meaning
line-height	Leading	Space between lines
font-family	Typeface	Used to describe specific type faces such as Helvetica
color	Type color	The color of the text characters
layer	Multiple meanings within software tools	For Dreamweaver users, a layer is actually an absolutely positioned element.
#FFF	White	White

Table 2. Brief Comparison of terminology in visual design and CSS

Learning and Implementation Curves, continued

At first glance it's easy to think that these are simple issues and quickly remedied. But it is undeniable that the lack of clarity in terminology has led to a steeper learning curve than using a table (grid) and then presentational elements to work with that grid—much more intuitive to designers who were taught traditional grid, color and typographic design.

Hot Tip If you use Dreamweaver, avoid using Dreamweaver's Layers (a term that's been dropped from CS4, thankfully), as they use style in a less-than best case scenario. While this feature can be helpful for wireframing, it can be downright disastrous in live sites. Review topics such as positioning and CSS floats for alternatives to this feature.

That there's a lack of consistent and well-paced implementation in Web browsers and other user software is sadly well known, making actual use of CSS a combination of good code riddled with workarounds, hacks, filters and JavaScript patchwork to repair problems across browser and browser versions. Error handling is a particularly frustrating part of this, particularly as Web browsers implement different versions of different specifications at different times (Figure 2).

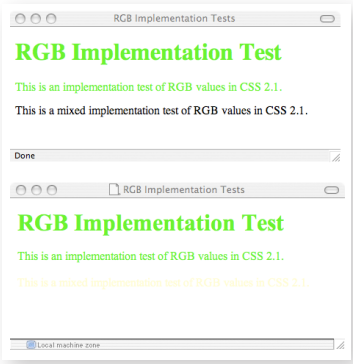


Figure 2. Error handling in browsers. In the first instance, if there's a mistake in the rule, the browser simply drops the erroneous rule and reverts to the browser style. In the second instance, the browser instead makes an attempt to find a nearest value and apply the color to nearly disastrous results.

This is why learning as much as you can about how CSS works is so empowering. As you begin to understand that most frustration with CSS is not your fault and learn some techniques to work with some of CSS's complexities, you'll be able to reduce the frustration caused by certain browser differences, CSS implementation, and be able to focus on the use of CSS for design and document/application management.

UNDERSTANDING STYLE RULE SYNTAX

We'll take a look at rule syntax here, which will set you up to quickly understand the basic structure of CSS as we discuss rules in the context of other language issues.

A CSS rule contains at least one selector and at least one declaration within a declaration block. A declaration is made up of a property name and a corresponding value. Declaration blocks are defined by curly brackets "{}" and declarations are separated with a semicolon:

```
h1 {color: red;}
```

→

Understanding Style Rule Syntax, continued

This rule in turn has the browser find a match to the `h1` selector and give it a color of red (Figure 3).

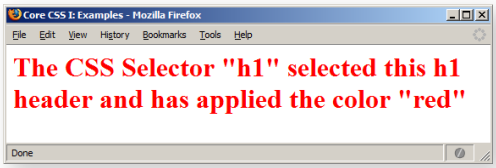


Figure 3. Using the h1 selector to apply the color "red" to a corresponding h1 element

Additional declarations are simply added to the block:

```
h1 {color: red; font-size: 80%; font-family: Verdana;}
```

this rule asks the browser to match any h1, color it red, size it relatively to 80% and apply the Verdana typeface (Figure 4):

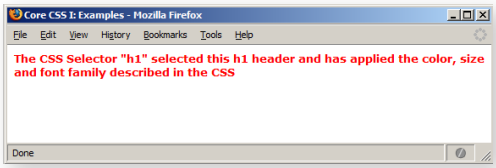


Figure 4. Adding additional declarations to the rule.

Grammar	Purpose	Examples

Selector	A *selector* chooses ("selects") an element within markup documents to be styled. There are many selector types in CSS 2.1, and even more to come in future years.	`h1`
		`#content`
		`.module`
		`:hover`

Declaration	A declaration is made up of a CSS *property* and a related *property value*. CSS properties are numerous and define various styles as they relate to colors, text, positioning, margins, padding, and positioning. A declaration can have as many property and value pairs as you like, contained as a group in a declaration block and separated with a semi-colon ";"	`color: red;`
		`font-variant: small caps;`
		`margin: 0 0 0;`
		`background–image:(my.jpg);`

Declaration Block	Multiple declarations related to a given selector are referred to as a *Declaration Block*.	`{color: red; font-variant: small caps; margin: 0 0 0; background –image: (my.jpg); }`
CSS Rule	A selector plus a declaration or declaration block makes a CSS rule.	`.module {color: red; font-variant: small caps; margin: 0 0 0; background –image: (my.jpg); }`
Style Sheet	Any set of style rules	(See "Style Sheet Types" later in this reference)

Table 3. Rule Syntax in CSS

Once again, bear in mind that there are many selector, property, and property value types. You'll work with many of them within the series, and be sure to look for the online references provided so you'll have plenty of resources.

INHERITANCE

It's important to know that many properties and associated values are inherited. It's a fairly simplistic concept, simply relate it to what you know about inheritance in living beings. I have my mother's curly hair, the shape of my father's eyes. And, just as we could map out a family tree and see where some of those features came from, so can we use the document tree to do the same (Figure 5).

Inheritance, continued

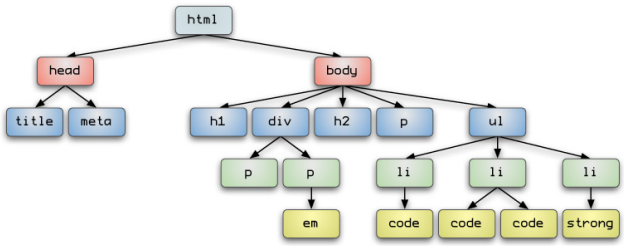


Figure 5. Inheritance. Imagine pouring a bucket of blue paint onto the body element. Because color is an inheritable property, all text descending from body will be blue until another style overrides the inherited color.

Some properties are not inheritable, mostly those related to the box model (margins, padding, box widths and so on), however most are. Authors can tap into the power of inheritance by allowing inheritable properties to be inherited by their children or descendents, or prohibit inheritance if so desired.

STYLE SHEET TYPES

There are three primary types of style sheets as follows:

- **Browser.** The browser style sheet is the default style of a given browser. It is either an actual .css file such as we find in Mozilla browsers, or hard coded into the software. Browser styles are different between browsers and versions, so being aware of them is extremely important. Wherever you do not supply a style, the browser style will be used instead.
- **User.** User style sheets are a great concept that has unfortunately not been brought to bear on a large scale. User styles are meant for accessibility purposes. My aging eyes require larger text and higher contrast, I can write a quick style sheet to address my issues and apply it via the browser.
- **Author.** The author is you! That is, author styles are those styles that the developer or designer is in charge of creating to create the design and management scenarios for a given Web site or application.

Hot Tip

At some point you might have come across the !important (referred to as "bang important") keywords. The proper use for !important is to create a balance between author and user style sheets, a necessity for accessibility purposes. If used in an author style sheet but not a user style sheet, the author's rule is considered to have more weight and therefore will apply. Because of this, !important is useful only in two places: As a diagnostic tool which you remove from the declaration after debugging a problem; and in a user style sheet. Otherwise, please avoid usage at all costs.

Author Style Sheets by Type

There are three types of author style sheets: Inline, embedded and linked (external). Each is authored differently and has different applications, benefits, and concerns.

Inline Style

Inline style is style that is used directly in the markup document to style one discrete element. No matter what other style sheets →

Author Style Sheets by Type, continued

might be influencing the document, an inline style is considered more specific and therefore will apply to that element no matter other conflicting styles.

Consider the following paragraph element:

```
<p>This paragraph is styled only by default
browser styles</p>
```

The style is placed within the style attribute as a value:

```
<p style="color: blue;">This paragraph will now
have a blue color.</p>
```

Figure 6 shows the comparison.

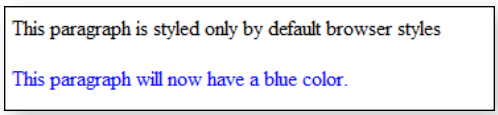


Figure 6. Applying style to a discrete element using inline style.

If you're thinking "but that code really looks just like presentational markup!" give yourself a big pat on the back. In recent years, many people, including those at the W3C responsible for advancing markup and CSS, have advised that using this technique isn't really separating presentation from the document at all!

So what benefits does inline style really offer? Table 4 provides some best practice insights as to when to avoid and when to use inline style.

Scenario	Issues	Best Practice
Inline style in small versus large Web sites	If you have a very small site (10 documents or less) the risk of losing track of inline styles is less than if you are working on very large sites, where it's easy to lose track of inline styles unless they are meticulously documented. And who does that?	Avoid use of inline style in almost all professional web sites, and in particular those sites which are large or expected to grow significantly.
Inline style for debugging purposes	As the section on "specificity" will demonstrate, an inline style has the highest specificity of any other rule that might be trying to style the element in question. If you are having trouble getting to the heart of the matter, dropping an inline style into the element you are having trouble with can help determine that in fact, there's a conflict.	Use inline style when necessary to debug. Typically, if you are able to apply a style inline that you'd been struggling with before means you have rules conflicting somewhere that need to be found. Find the conflicts, repair the rules, and remove the diagnostic inline style prior to publishing!
Inline style as "quick" fixes (aka laziness)	There are very few benefits from using inline style, but one I find that's great is for quick fixes and blog posts, which you can do on the fly.	Despite the fact that I do this myself, it's not something I'd recommend, particularly for professional sites.

Table 4. Best Practices: Inline Style

Embedded Style

Embedded style is used to control the style of a single document. In this case, the style element is used to define the embedded area for the document's style as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www/w3/org/TR/xhtml1">
<head>
<title>Core CSS I: Examples</title>
<style type="text/css">
p {color: blue;}
</style>
</head>
<body>
<p>In this case, all paragraphs on the page will
turn blue.</p>
</body>
</html>
```

So as with inline style, we're left looking at a type of style sheet that, while handy in some cases, doesn't provide the benefits we're looking for. With the style element in the head portion of the document, we do achieve slightly better separation of presentation from our document's content and structure, but only in that same document. Table 5 provides some insights into the best ways to use embedded style.

Scenario	Issues	Best Practice
Embedded style in small versus large Web sites	If you have a weblog with one template document that controls your entire site, it is feasible to use embedded style in this instance. However, in *any* professional site or app development, avoid using embedded style, for it, like inline style, can contribute to confusion when debugging.	Avoid in professional practice.
Embedded style for debugging and workflow purposes	As with inline styles, if you're trying to isolate why a given style isn't applying, you can use embedded style to work through some conflicts. Another use that I find helpful is that during development, I like to work in one document, embedding my styles and building out the content and markup all in the same place.	Though not ideal, embedded style can be used to debug and find conflicts in the case of multiple style sheets. Workflow advantages as described can be useful, the one caveat in all instances of professional sites: Remove your embedded styles out to appropriate external styles after you're done working, test, and you're good to go.
Embedded style as "quick" fixes (aka laziness)	Where is that style? What if you want to use it again more efficiently?	Avoid publishing embedded style sheets.

Table 5. Best Practices: Embedded Style

Linked (External Style)

Linked style is the true "holy grail" of style sheet types. It provides us with the broadest application of style; allows us to manage the presentational aspects of a site from a handful of style sheets; performance is faster due to the browser placing the styles into memory (cache); and the worst of conflicts are avoided.

Linked style sheets are separate text documents containing your style rules, saved with a .css file extension and linked to from the HTML documents you want to style using the link element in the head portion of your markup document:

```
<head>
<link rel="stylesheet" type="text/css" src="style/
global.css">
</head>
```

In almost all cases, the linked style sheet is the one you will be working with most.

Hot Tip Be careful with case matching between your CSS and HTML documents. If you create a selector H1 in upper case, then it will only select h1s in upper case within the markup documents. Best practices suggest keeping all HTML elements and attribute names in lower case (this is required in XHTML) and keeping CSS lower case as well, helping to avoid potential case-related conflicts. Also, while many programmers find camel case (class="ModuleTwo") intuitive, this also can cause case-matching problems, particularly in larger-scale sites, particularly those being managed by multi-person teams.

APPLICATION HIERARCHY AND SORT ORDER

Of course, most working Web developers and designers are well aware that working with CSS just can't be that straight forward! There are many reasons why CSS is as broad in scope as it is, but flexibility and power are two of the most credible reasons for why you can approach a given problem with numerous solutions. With freedom comes responsibility, and the same is true for professional Web development.

In order to visualize why CSS can quickly fall from powerful friend to chaotic foe, consider Figure 7.

Figure 7. Imagine a global style for the University itself. Then, each individual department wants its own identity. This is a very common large-organization issue, and one which inevitably leads to multiple styles all over the site, poorly documented and managed when in fact some intelligent coordination could be used to manage the site's presentation much more efficiently.

The Cascade

Revisiting the browser, user, author relationship, we can take a look at how rules "cascade" from one style sheet type to another. Here's the general rule of thumb:

- All explicit styles override browser style
- A user style sheet, when properly authored, will override author style
- An inline style overrides a conflicting embedded style
- An embedded style overrides a conflicting linked style

Rule Order

The order in which rules are sorted becomes critical in resolving conflicting rules. Many readers are likely to have heard "the rule closest to the content wins"—which is somewhat accurate but also a bit misleading.

Sort order, the term that is used to describe the sorting of multiple CSS rules, is the process by which a Web browser sorts the rules it is given. If we have a scenario where there are two linked style sheets, an embedded sheet in the document in question, and an inline style, the browser has to sort through those and resolve sort conflicts. Consider this XHTML :

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html xmlns="http://www/w3/org/TR/xhtml1">
<head>
<link rel="stylesheet" type="text/css" src="style/global.css" />
<link rel="stylesheet" type="text/css" src="style/local.css" />
<style type="text/css">
p {font-family: Times;}
</style>
</head>
<body>
<p style="font-family: Arial;">Which font will this be?</p>
<p>What about this paragraph?</p>
</body>
</html>
```

Rule Order, continued

If you imagine all the rules within global.css expanding out, then the local.css expanding out, then the embedded style added onto that, you get one long style sheet. If somewhere in the first two I had conflicting rules that styled paragraphs using the Geneva font, the sort order process will see the last style in that long sheet as Times. Therefore, Times will be used in all instances of p as defined within our scope *with the exception of the element containing the inline style*. As mentioned earlier, inline styles are more specific, and therefore will always "trump" another style in a scenario like this.

Specificity

There is one final deal-breaker for the rules of Cascade and sort, and that is the *specificity* of a given selector. I've kept the examples here simple for a reason as selectors are complicated and actually take up about a third of the Core CSS series.

Specificity is an algorithm with a broad base that allows an author to create very specific rules. These rules often involve a number of selector types, and are calculated based on the selectors in use in the rule. If a rule is found to be more specific than one that comes later in the sort order *the more specific rule is applied* no matter where the rule resides in the sort.

Consider the following rule:

```
#content p {font-family: Garamond;}
```

This is a combination of an ID selector (#content) and an element selector (p). The space between the two selectors indicates a descendent. So, let's say I have this rule in global.css. Because it is more specific, any paragraph that descends from an element with an ID of #content will now be in Garamond, not in Times.

Specificity is one of the terribly misunderstood and under-taught portions of conflict resolution within a CSS application hierarchy. Understanding how to calculate specificity is easier if you have a table available to work through a given conflict, then count up the types of selectors that exist in your rules in the exact order shown in Table 6.

Example	Count # of ID Selectors	Count # of Class Selectors	Count # of Element Selectors
`ul`	0,	0,	1
`#content ul li`	1,	0,	2
`#content ul li ul li`	1,	0,	4

Table 6. Calculating Specificity

We can now see that the most specific rule is the last one. Therefore, any list item style that is not as specific will not apply to a nested list item within the document area with an ID of "content"—regardless of where that more specific rule resides in the sort.

There's one specific specificity exception here. Remember that I mentioned inline style has the highest specificity? Table 7 shows how inline style comes into play:

Example	Presence of Inline style in element	Count # of ID Selectors	Count # of Class Selectors	Count # of Element Selectors
`ul`	1,	0,	0,	1
`#content ul li`	1,	1,	0,	2
`#content ul li ul li`	1,	1,	0,	4

Table 7. Specificity and presence of inline style

If t here are inline styles within the element, a count of 1 goes into the first (optional) column, skyrocketing the specificity of the given element. This is why inline style is really so powerful.

FIND MORE ONLINE

The following online references will be helpful additions to the learning in this refcard.

Reference	URL
CSS 2.1 Specification	http://www.w3.org/TR/CSS21/
CSS Discussion List	http://www.css-discuss.org/
CSS-D Wiki (Lots of helpful information and links)	http://css-discuss.incutio.com/
CSS Zen Garden (Beautiful showcase site)	http://www.csszengarden.com/

Table 8. Online References

NEXT STEPS

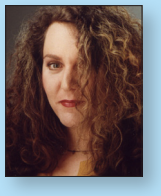
It should be clear that CSS has nuances that only time and experience can reveal. Well, that and good references! Look for the remaining "Core CSS" series and go into depth with selectors, the box model, floats, positioning and the z-index. Sound exciting? I think so too!

More **Core CSS** Refcardz:

Core CSS: Part II—October 2008

Core CSS: Part III—November 2008

ABOUT THE AUTHOR



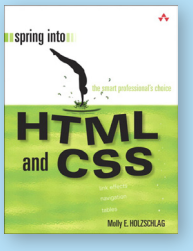
Molly E. Holzschlag

Molly E. Holzschlag is a well-known Web standards advocate, instructor, and author. She is an Invited Expert to the W3C, and has served as Group Lead for the Web Standards Project (WaSP). She has written more than 30 books covering client-side development and design for the Web. Currently, Molly works to educate designers and developers on using Web technologies in practical ways to create highly sustainable, maintainable, accessible, interactive and beautiful Web sites for the global community. She consults with major companies and organizations such as AOL, BBC, Microsoft, Yahoo! and many others in an effort to improve standards support, workflow, solve interoperability concerns and address the long-term management of highly interactive, large-scale sites. A popular and colorful individual, Molly has a particular passion for people, blogs, and the use of technology for social progress.

Web Site

http://www.molly.com

RECOMMENDED BOOK



With *Spring Into HTML and CSS* you'll master today's best practices: the real nuts and bolts, not theory or hooey. You'll learn through dozens of focused HTML, XHTML, and CSS examples: crafted for simplicity and easy to adapt for your own projects.

BUY NOW

books.dzone.com/books/spring-html-css

Get More FREE Refcardz. Visit refcardz.com now!

Upcoming Refcardz:

Core Seam
Core CSS: Part III
Hibernate Search
Equinox
EMF
XML
JSP Expression Language
ALM Best Practices
HTML and XHTML

Available:

Essential Ruby
Essential MySQL
JUnit and EasyMock
Getting Started with MyEclipse
Spring Annotations
Core Java
Core CSS: Part II
PHP
Getting Started with JPA
JavaServer Faces
Core CSS: Part I
Struts2
Core .NET
Very First Steps in Flex
C#
Groovy
NetBeans IDE 6.1 Java Editor
RSS and Atom
GlassFish Application Server
Silverlight 2

Visit **refcardz.com** for a complete listing of available Refcardz.

Design Patterns
Published June 2008

DZone

DZone communities deliver over 4 million pages each month to more than 1.7 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-18-9
ISBN-10: 1-934238-18-X
50795
9 781934 238189
$7.95

Version 1.0

DZone Refcardz

CONTENTS INCLUDE:

Core CSS: Part II

By Molly E. Holzschlag

ALL ABOUT SELECTORS

Just in case you've not read *Core CSS: Part I*, I'll briefly review the purpose of a CSS selector. A selector in a style sheet signals the browser to find matches within those markup (HTML, XHTML, XML) documents to which the style sheet is related.

There are more than a few selectors available for use (Table 1), but even intermediate and advanced CSS authors don't always have an opportunity to use some of them, largely due to cross-browser support issues for a given selector. *Core CSS: Part II* will cover CSS 2.0/2.1 selectors. Where a selector is unavailable in contemporary Web browsers, a caution will be provided to alert you to any support concerns.

Version	Date
Element	Selects by matching element
Class	Selects by matching class name
ID	Selects by matching id name
Pseudo Class	Selects by matching predefined pseudo class
Descendent (also known as "contextual")	Selects by descendant elements
Child	Selects by first-level (child) elements
Adjacent Sibling	Selects by matching sibling element
Attribute	Selects by matching attribute names and values
Pseudo Elements	Selects by matching predefined pseudo element

Table 1. CSS Selectors Covered in this Refcard

To assist in visualizing how these selectors actually match, for each example you'll see an element, the corresponding CSS, and a document tree that visualizes what is selected within a sample document. Also provided are some use examples.

Hot Tip There's also a universal selector, *, which when used will select every single element within a document. It's used in several hacks, including the infamous "star html" hack, which is problematic and invalid. While the universal selector is important to know about, it's probably not going to be something you use too often in real-world scenarios.

Element Selectors

Element selectors, also referred to as "type" selectors, select by matching elements. They are very broad in scope. For example, if I have a million documents with many more millions of **h2** elements within them, by using an element selector I can single handedly apply styles to all of those **h2** elements using one rule. Element selectors are supported in all CSS browsers and are very widely used for these reasons.

Here's a CSS rule using an element selector:

```
h2 {color: #f00; font-family: Garamond; font-size:
22px; font-variant: small-caps;}
```

In the corresponding markup document(s), all h2s are selected and the style is applied (Figure 1).

Element Selectors, continued

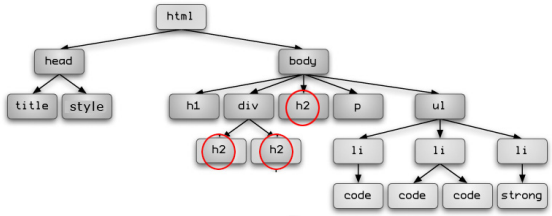


Figure 1. An HTML document tree showing that each of the h2s in the document has been selected.

Class Selectors

Class selectors are extremely useful selectors that allow authors to add a class attribute to a given element in the markup, with a custom value. Then using that value preceded by a dot, write a corresponding rule using the class name.

An example of an element with an added class attribute in the markup document would be:

```
<p class="warning">Any paragraph in any document on
any page containing this class will have the class
rules apply.</p>
```

In the CSS:

```
.warning {color: red; font-weight: bold;}
```

Figure 2 shows how this class, as written, will apply to both elements with the class attribute named "warning".

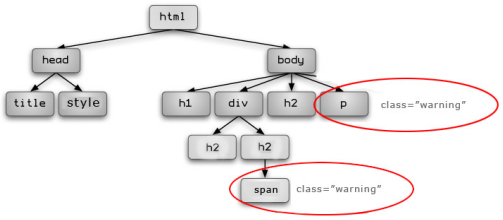


Figure 2. Selecting all elements with a class of warning.

→

Class Selectors, continued

Hot Tip You can limit a class to a specific element by placing the element selector before the class: p.warning. If you were to do this, only the paragraph will take on the class styles. Any other, non-conflicting styles that exist for the element p will also be sorted and included.

Multi-Classing

An interesting and occasionally useful technique is multi-classing. This means using more than one class to get the style you're after. A good use scenario would be a portal site in which you have multiple modules that have common colors and features, but require different background images. Consider the following style rules:

```
.module1 {width: 200px; margin: 5px; border:
1px solid blue;}

.weather {background-image: url(images/"sunshine.jpg");}
```

To multiclass, I'd simply add both classes to the module element, with each name separated by a space:

```
<div class="module1 weather"> . . . </div>
```

The element will now pick up the styles of both classes. Typically, use of 2-3 class names can be helpful within context, but it's not a practice I'd recommend using without a strong sense of your site hierarchy and management.

It's also important to point out that the source order of the class names in the markup document *is of no consequence*. However, if there are conflicts between the classes, sort order and specificity rules in CSS will calculate which rule takes precedence.

Hot Tip Avoid underscores and other special characters in class and ID names. The best practice currently is to use hyphenation: nav-main (not nav_main). Also, while camelCasing is extremely useful to coders, it can add a layer of extra testing because CSS requires case-matching, so case within the markup documents and any associated CSS must match for rules to apply.

ID Selectors

ID selectors are meant to identify a discrete portion of a document. This means an ID name can be used exactly one time in a given document. This is why ID's are particularly useful in CSS layout when identifying significant portions of the document, such as "content" "nav" or "site-info"— because they are unique, discrete pieces of the document structure. Assuming only one document, here's a right/wrong comparison:

Right:

```
<div id="content"> . . . </div>
<div id="sub-content"> . . . </div>
```

Wrong:

```
<div id="content"> . . . </div>
<div id="content"> . . . </div>
```

In the style sheet ID selectors are written using the hash "#" (also known as an 'octothorpe' for the word geeks among you) preceding a custom name.

```
#content {width: 500px; border: 1px solid #fff;}
```

Figure 3 shows how this will now select the div with the unique id attribute named "content".

ID Selectors, continued

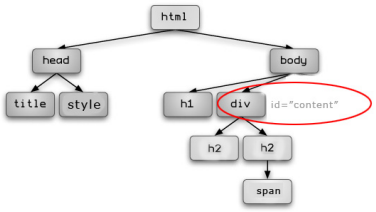


Figure 3. Using ID selectors to identify the content area. The id name can only be used once per document, but many times within a site.

Hot Tip While ID and Class names are in fact customized to suit your needs, it's considered best practice to avoid presentational names such as `.redfont` or `#sidebar`. What happens when the boss says "update all the red fonts on the site to be blue?" Easy enough to change the style in moments and update all those fonts, but now the markup documents are littered with `class="redfont"` when the actual visual result is blue! To avoid confusion of this nature, use naming that is descriptive (referred to as *semantic naming*) and where possible, consider creating conventions to be used site-wide.

Pseudo Class Selectors

A pseudo class selector is a set of predefined class-like selectors. Pseudo class selectors are written with a colon followed by the predefined pseudo class name. Pseudo classes can then be attached to a variety of elements in order to achieve a given result. It's likely you've used pseudo classes as much as element, ID, and class selectors, for a number of them are integral to styling links (Table 2).

Selector	Purpose	Example
`:link`	Selects links that have not been visited	`a:link {color: blue;}`
`:visited`	Selects links that have been visited	`a:visited (color: violet;}`
`:hover`	Selects an element as the mouse passes over.	`a:hover {color: #ccc;}`
`:focus`	Selects the element that has focus	`a:focus {background-color: orange;}`
`:active`	Selects a link that is being activated	`a:active {color: red;}`
`:first-child`	Selects an element's first child	`div:first-child {font-style: italic;}`
`:lang`	Selects by matching human language	`html:lang (de) {font-size: 80%;}`

Table 2. Pseudo Class Selectors

Note that `:hover`, `:focus` and `:active` are all referred to as "dynamic pseudo classes" because along with presentation they also allow for dynamic behavior, such as creating interesting looks for navigation, assisting with usability, and styling form controls. An example would be (Figure 4).

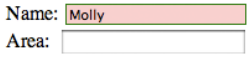


Figure 4. The form control that has focus (in this case the text input box associated with "Name") takes on the style you see here using the `:focus` selector on the input element.

Child Selectors

Child selectors are created by combining a parent element with the > combinator and a child element. This allows you to style only the child element or elements of the parent, without having those styles inherit down the tree. It's also a great way →

Child Selectors, continued

to reduce the use of class attributes, which help make managing sites all the more easy. Consider Figure 5.

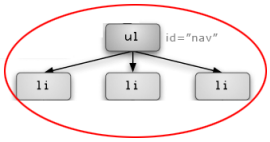


Figure 5. An unordered list element (parent) with three list item child elements.

Here, we have a parent element, ul, and we want to style each of the three list items below. The CSS rule would simply be:

```
ul>li {border: 0; margin 0; padding: 0;}
```

Now all the children of any ul will have 0 border, margin and padding. Because in this example, the ul has an ID, we can use that to limit this rule only to that discrete document element:

```
ul#nav > li {border: 0; margin 0; padding: 0;}
```

Not only has this limited the rule to the ul with an id of "nav", but it has also made the rule more specific both literally and technically. Also, you'll note that there's no white space surrounding the combinator in the first example, whereas in the second, there is. Either way is acceptable according to the spec.

You can use as many children within the selector as is required. In a scenario such as Figure 6, you could write a very specific selector to select only the children of the nested ordered list item and style it with a leading zero decimal.

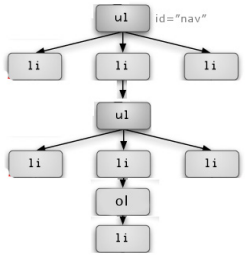


Figure 6. Tree depicting a nested ordered list within a nested unordered list with a parent unordered list. Using Child selectors, we can select children by following their ancestral path.

The resulting CSS would be:

```
ul#nav > li > ul > li > ol > li {list-style-type:
decimal-leading-zero;}
```

CAUTION: CHILD SELECTORS ARE NOT IMPLEMENTED IN INTERNET EXPLORER 6.0 OR BELOW.

Descendent Selectors

Descendent selectors, as with Child selectors, begin with an element that has descendents. The combinator for descendents is a space. Since children are descendents, we can re-examine the same parent-child relationship we first did when examining child selectors (Figure 7).

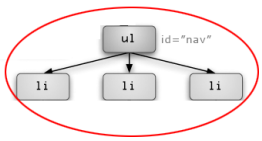


Figure 7. Children are also descendents.

If I wanted to set a style for any descendent list items within an unordered list using a descendent selector, I can do so as follows:

```
ul#nav li {list-style-type: none;}
```

Descendent Selectors, continued

The differences is that not only the `li` children of the `ul` will be styled, but all `li` descendants of that `ul` and the `ol` will get the same style as well since all list items descend from the *original unordered list* (Figure 8).

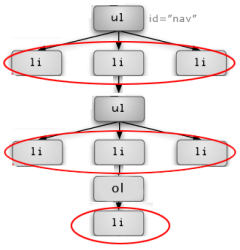


Figure 8. Descendent selectors select all descendents of the defined parent element, in this case, nested unordered lists. Note that the list item that is child to the ordered list element will also receive the style, for it too is a descendent of the original parent list.

As with child selectors, we can create strings to reach a particular element within the tree:

```
ul#nav li ul li ol li {list-style-type: decimal-
leading-zero;}
```

The selector will now select the very last list item in Figure 8, which is the child of the ordered list item element in the tree hierarchy. None of the other list items will take on this rule.

Fortunately, Descendent selectors are widely supported in current CSS browsers including IE 6.0 and later.

Adjacent Sibling Selectors

An Adjacent Sibling selector allows you to select an element based on its nearest sibling element. Consider the following markup:

```
<div>
<h1>Main Content Header</h1>
<p>First paragraph</p>
<p>Second paragraph</p>
<p>Third paragraph</p>
</div>
```

It's a common design theme to style a first paragraph somewhat differently using a larger font, or emphasized font, bringing the reader's eye to the critical introductory material. Using an Adjacent sibling selector, we can do this quite easily without using a class attribute on the first paragraph. The combinator for the Adjacent sibling selector is the plus sign, +.

```
h1+p {font-weight: bold;}
```

This selects the first adjacent paragraph element (Figure 9), with no change to any of the other siblings.

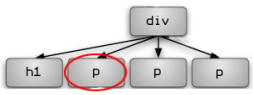


Figure 9. Selecting a sibling using Adjacent sibling selectors.

You can use multiple sibling elements to reach a given sibling. Let's say you wanted to select not the first but the third paragraph in the example and have it display as italic. The syntax would be:

```
h1+p+p {font-style: italic;}
```

Figure 10 shows the selection.

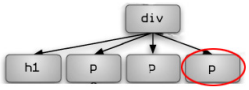


Figure 10. Selecting a further removed sibling using a string of adjacent siblings.

→

Adjacent Sibling Selectors, continued

CAUTION: ADJACENT SIBLING SELECTORS ARE NOT IMPLEMENTED IN INTERNET EXPLORER 6.0 OR BELOW.

Attribute Selectors

Attribute selectors are a curious piece of selectors because they really are more akin to programmatic pattern matching than presentational design needs. There are four Attribute selectors that are available in CSS 2.1 (Table 3).

Attribute Selector	Pattern Matching	Example
`[name]`	Selects by presence of attribute name for a given element	`a[title] {font-style: italic;}`
`[name+value]`	Selects by presence of the attribute name plus its value	`img[src="photo.jpg"]`
`[name~="value"]`	Selects by the attribute name plus the presence of a specific space separated word within the attribute value	`img[alt~="Portland"]`
`[name|="value"]`	Selects by the attribute name plus the presence of a hyphenated word within the attribute value	`a[title|="top-down"]`

Table 3. Attribute Selectors

To select by attribute name...

Compare the following two links:

```
<a href=http://molly.com/" title="go to Molly's Web site">Molly.Com, Inc.</a>
<a href="http://molly.com/'>Molly.Com, Inc.</a>
```

In the first link, there's a title attribute. Using the following CSS:

```
a[title] {font-style: italic;}
```

We can style any anchor elements with a title attribute present, but the style will not apply where no title attribute is present (Figure 11).

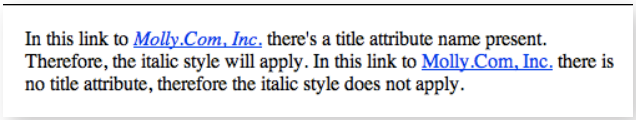


Figure 11. Applying style using an attribute name selector.

To select by attribute name and value...

Consider the following two HTML image elements:

```
<img src="images/photo.jpg">
<img src="images/screenshot.jpg">
```

To add a specific style to the first instance, you can use the following syntax:

```
img[src="photo.jpg"] {border: 2px solid #000;}
```

The selector will match only an image element with an attribute of `src="images/photo.jpg"` and no other image elements will be selected (Figure 12).

Figure 12. Applying a border to only the photo using the complete (name+value) attribute selector.

Attribute Selectors, continued

To select by presence of multiple space separators and hyphens...

Consider the following XHTML image elements:

```
<img src="images/vacation01.jpg" alt="Portland vacation photo" />
<img src="images/vacation04.jpg" alt="vacation photo" />
<img src="images/vaction03.jpg" alt="vacation photo of Portland" />
```

To add style to only those images that have an alt attribute (and all your images should!), and a series of space separated words that include "Portland" (note that the case must match as well) you'd use the following syntax:

```
img[alt~="Portland"] {border: 5px solid green;}
```

Figure 13 shows the results.

Figure 13. Applying a border to only the photos with multiple space separated words where "Portland" appears within the image's alternative text string.

Similarly, you can select by the presence of an attribute name plus a hyphenated, specified word within the value. Consider the following HTML:

```
<p title="nursery-rhyme">Mary, Mary, quite contrary, how does your garden grow?</p>
<p title="song-lyric">And she's buying a Stairway to Heaven</p>
<p title="traditional-rhyme">Roses are red, violets are blue</p>
```

Add this CSS rule:

```
p[title|="rhyme"] {color: blue;}
```

Both the first and third elements will take the style, whereas the middle one will not. (Figure 14).

Figure 14. Applying style using pattern matching. Note however that in the case of hyphen matching, order matters. The hyphenated word must be first in the string. Had we switched the third paragraph's attribute to `title="rhyme-traditional"` the style should not apply.

CAUTION: ATTRIBUTE SELECTORS ARE NOT IMPLEMENTED IN INTERNET EXPLORER 6.0 OR BELOW.

Pseudo Elements

As with pseudo classes, pseudo elements are predefined elements within CSS. There are four of which to be aware, as described in Table 4.

Pseudo Element	Purpose	Example
`:first-line`	Selects only the first line of text in a given element.	`blockquote:first-line {font-weight: bold;}`
`:first-letter`	Selects only the first letter of text in a given element	`p:first-letter {font-size: 250%}`
`:before`	Used to generate content before a given element	`q:before {content: open-quote;}`
`:after`	Used to generate content after a given element	`q:after {content: close-quote}`

Table 4. Pseudo Elements in CSS 2.1

→

Pseudo Elements, continued

First line and letter pseudo elements

Both the :first-line and :first-letter pseudo elements are typically used to add typographic features to a given set of text. The following HTML block shows what happens in the document:

```
<p>Let's be honest. We all make mistakes. Sometimes
we can be too hard on ourselves, or others, for those
mistakes. It makes me remember that long ago and far
away, someone very wise said:</p>

<blockquote>To err is human, to forgive divine.</
blockquote>

<p>Having both the capacity to be forgiving of others
and the ability to forgive yourself is part of
learning how to be wise.</p>
```

Using the decorative pseudo elements, here are the CSS examples from Table 4:

```
blockquote:first-letter {font-size: 250%}
p:first-line {font-weight: bold;}
```

Figure 15 shows the results.

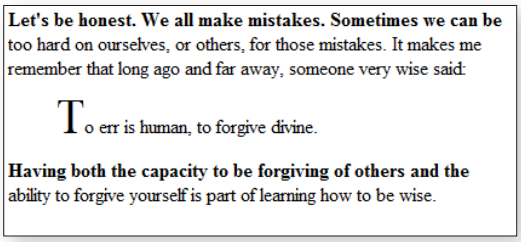


Figure 15. Using first letter and line pseudo elements to apply style. Notice that in the case of :first-line, the "line" is defined as whatever amount of characters make up the first line. Because this is not always the desired result, using `min-width` and `max-width` properties to limit line length wherever possible can address this issue.

Both the first letter and line pseudo elements have good support across browsers, including IE 6.0.

Generated Content

A fascinating if controversial portion of CSS is called generated content. This is when, using the pseudo elements `:before` and/or `:after`, you as the author can actually generate text, symbols and images. What's more, you can style them on the page. Consider the quote from earlier:

```
<blockquote>To err is human, to forgive divine.</
blockquote>
```

Now, let's generate quote marks and style them using CSS:

```
blockquote {font-size: 30px; font-weight: bold;}
blockquote:before {content: open-quote; color: red;
font-size: 120px;}
blockquote:after {content: close-quote; color: red;
font-size: 120px;}
```

Figure 16 shows the results in Firefox.

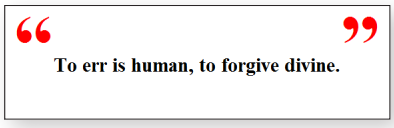


Figure 16. Using pseudo elements to generate and style the quote marks.

The caveat, and the cause of misuse and therefore controversy has to do with the fact that the content generated by pseudo elements results in pseudo content. In practical terms, this means the content never actually appears in the content layer, only the presentational layer!

In a situation where the generated content is largely decorative or practical in some sense but does not inhibit access to important data, this is fine. Take a look at the generated source by Firefox and you'll see the quotes called for do not appear in the code at all.

But what if we were to generate the message itself? In the HTML:

```
<blockquote></blockquote>
```

And in the CSS:

```
blockquote:after {content: "To err is human, to
forgive divine" font-size: 90px;}
```

Figure 17 shows the generated results.

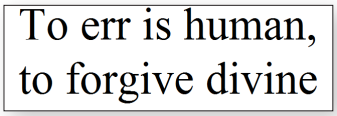


Figure 17. You can generate actual content, but it will only appear on the presentational surface.

However, when we look at the source code, we see that the generated content does not appear within the code (Figure 18).

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<html xmlns="http://www/w3/org/TR/xhtml1">
<head>
<title>Untitled</title>
</head>

<style type="text/css">

blockquote:after {content: "To err is human, to forgive divine"; font-size: 90px;}

</style>


<body>

<blockquote></blockquote>

</body>
</html>
```

Figure 18. While we can visually see the generated content on the screen, it does not appear within the actual body of the document.

Therefore, if you are generating important content to the desktop screen that must be comprehensible, generated content is not the way to go. It can cause problems for copying, printing, reading, saving, and for anyone using Internet Explorer IE7 or earlier, simply non-existent due to complete lack of implementation for the `:before` and `:after` pseudo elements.

COMBINING SELECTORS

Selectors can be combined, giving authors highly specific ways of working to style and manage documents.

Grouping

Selector grouping is simply placing a number of selectors that all share common properties separated by commas:

```
h1, h2, h3, h4, h5, h6, p, q, blockquote, td,
#content, .standard {color: #000; margin: 5px;}
```

Now all these selectors will share the declaration properties.

Hot Tip Grouping is useful when you have a lot of shared features between elements. You can group those elements as shown, and then create more specific rules for individual elements. You might have heard of "CSS reset" or "normalization" which uses this technique.

Combining Selector Types

As you've already seen in several of this refcard's examples, you can combine selector types in order to create what some designers and developers refer to as complex selectors. Table 5 →

Combining Selectors, continued

shows some examples as well as the selectors definition—and to help you practice—the selector's specificity. Read selectors from the right of the selector—it helps!

Combined Selector	Meaning	Specificity (CSS 2)
`#content div.module > p`	Selects child paragraphs descending from a element that has a class of "module" and is within the uniquely identified portion of the document that is identified as "content"	`1,1,2`
`#main-nav ul li ol > li:hover`	Selects only the first letter of text in a given element	`1,0,4 (CSS 2)` `1,1,4 (CSS 2.1)`
`tr > td+td+td > table`	Any table element that is a child of a table data element that is the third sibling from a table row element.	`0,0,5`
`#content ul > li + li a[href="http://molly.com/"]`	Any anchor with an href of http://molly.com/ that is the second child sibling from an unordered list element descending from an element with an ID of content.	`1,0,4 (CSS 2)` `1,1,4 (CSS 2.1)`

Table 5. Combining selectors to create highly specific rules

RESOURCES

The resources in Table 6 should help you get more information on the topics discussed in this card.

URL	Reference
http://www.w3.org/TR/CSS21/cascade.html#specificity	Specificity in CSS 2.1 explained
http://www.w3.org/TR/REC-CSS2/cascade.html#specificity	Specificity in CSS 2.0
http://gallery.theopalgroup.com/selectoracle/	SelectOracle: Free online tool to help you calculate selector specificity
http://developer.yahoo.com/yui/reset/	Yahoo! User Interface library reset
http://meyerweb.com/eric/thoughts/2007/05/01/reset-reloaded/	Eric Meyer's take on using reset or "normalization"

Table 6. Resources

More **Core CSS** Refcardz:
Core CSS: Part III—December 2008
Core CSS: Part I—Available Now!

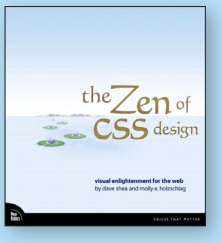
ABOUT THE AUTHOR

Molly E. Holzschlag

Molly E. Holzschlag is a well-known Web standards advocate, instructor, and author. She is an Invited Expert to the W3C, and has served as Group Lead for the Web Standards Project (WaSP). She has written more than 30 books covering client-side development and design for the Web. Currently, Molly works to educate designers and developers on using Web technologies in practical ways to create highly sustainable, maintainable, accessible, interactive and beautiful Web sites for the global community. She consults with major companies and organizations such as AOL, BBC, Microsoft, Yahoo! and many others in an effort to improve standards support, workflow, solve interoperability concerns and address the long-term management of highly interactive, large-scale sites. A popular and colorful individual, Molly has a particular passion for people, blogs, and the use of technology for social progress.

Web Site
http://www.molly.com

RECOMMENDED BOOK



Proving once and for all that standards-compliant design does not equal dull design, this inspiring tome uses examples from the landmark CSS Zen Garden site as the foundation for discussions on how to create beautiful, progressive CSS-based Web sites.

BUY NOW
books.dzone.com/books/zencss

Get More FREE Refcardz. Visit refcardz.com now!

Upcoming Refcardz:

Core Seam
Core CSS: Part III
Hibernate Search
Equinox
EMF
XML
JSP Expression Language
ALM Best Practices
HTML and XHTML

Available:

Essential Ruby
Essential MySQL
JUnit and EasyMock
Getting Started with MyEclipse
Spring Annotations
Core Java
Core CSS: Part II
PHP
Getting Started with JPA
JavaServer Faces
Core CSS: Part I
Struts2
Core .NET
Very First Steps in Flex
C#
Groovy
NetBeans IDE 6.1 Java Editor
RSS and Atom
GlassFish Application Server
Silverlight 2

Visit **refcardz.com** for a complete listing of available Refcardz.

Design Patterns
Published June 2008

DZone

DZone communities deliver over 4 million pages each month to more than 1.7 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513
888.678.0399
919.678.0300
Refcardz Feedback Welcome
refcardz@dzone.com
Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-22-6
ISBN-10: 1-934238-22-8
50795
9 781934 238226
$7.95

CONTENTS INCLUDE:

Core CSS: Part III

By Molly E. Holzschlag

ABOUT THIS REFCARD

To understand and manage attractive design and layouts, gaining an appreciation for the how to manage the CSS Box Model, Floats and Positioning is paramount. In this third and final refcard in the Core CSS series we turn our attention to the visual models that exist within CSS. You'll learn not only about how visual models work, but how to troubleshoot and repair common problems too.

THE CSS VISUAL MODEL

The path to master today's options for layout requires a significant study of the way that browsers work with the markup and style they're interpreting. Current Web browsers implement what is known as the "CSS Visual Model" and lay out content based on a foundation of lines and boxes.

Lines and Boxes

It's very likely you've heard the terms "inline" and "block" to describe HTML and XHTML elements. I'll review their meaning here in the context of the model. An inline element by default is one that lies on the line with no subsequent break, unless the line has come to the end of available browser space in which to flow. Boxes that fall on a line in this fashion are called line boxes (Figure 1).

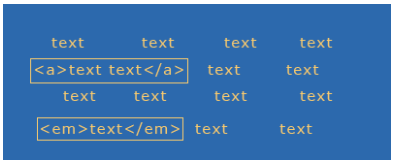


Figure 1. Imagine this is a series of text within a paragraph. Note how the inline elements do not cause a line break, and that they generate what is known as a "line box."

A block level element by default defines a block, and there is a break between it and the next element.

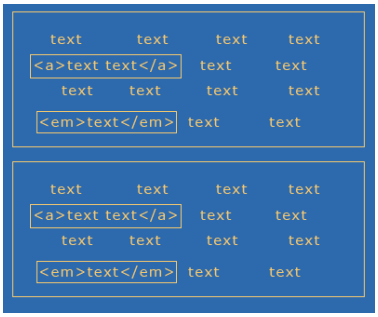


Figure 2. I've taken the original text and placed it within a paragraph element. Then, duplicated the element below, giving us two block boxes each containing inline boxes.

The CSS 2.1 Visual Model, continued

The most critical piece to understand is that ***every element creates a box***. This box is the foundation for what is known as the *box model* and is a critical component to understanding how to style elements within your document.

THE BOX MODEL

Every element box can be styled by the box components. In CSS 2.1, the Box Model consists of the content at center, and the top, right, bottom and left borders, padding and margin. The borders, padding and margin values for each or any side of the box are all optional and can be styled using CSS.

Hot Tip Boxes only go to the width and/or height of their content. You can control the width of block boxes by providing a specified width, but in the case of floats, you'll need to use clearing and other techniques to "stretch" element heights.

Figure 3. The CSS Box Model.

It's important to point out that because of flow issues within a browser, inline elements can't take a width. Therefore, while you can style other elements of an inline box, you can't set an explicit width.

Core CSS: Part III

www.dzone.com

The Box Model, continued

Understanding the box model is critical because it gives designers a greater sense of what they can actually style. For example, you can add backgrounds or styled borders to the content surrounded by the box. A bit of a warning, though, the Box Model in IE6 and below does not follow the CSS box model. Therefore, it is necessary to use the correct DOCTYPE declaration (See Core CSS: Part I Refcard) in IE6 and IE7 to ensure the proper box model is in use. Otherwise, width measurements will be significantly different.

Hot Tip Margin backgrounds in boxes are transparent.

About Normal Flow

Normal flow at its most simplistic is simply the default flow of elements within a document. Here is an example of normal browser flow when we diminish the width of the browser, content will flow down and to the left (Figures 4-6). If we reverse the sizing and expand the browser, the content will flow up and to the right (simply reverse viewing order of Figures 4-6).

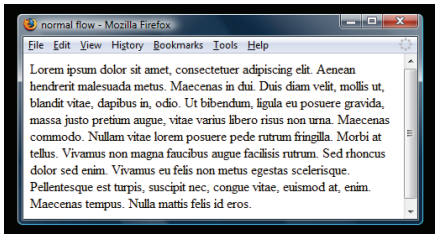


Figure 4. A paragraph in the normal flow.

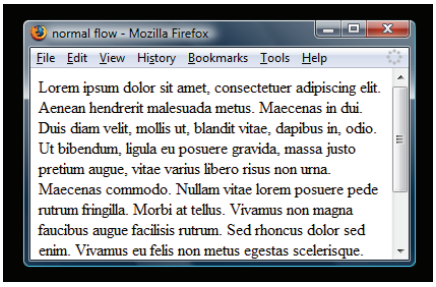


Figure 5. As the browser window is resized, text flows down and to the left. Notice the appearance of a scrollbar.

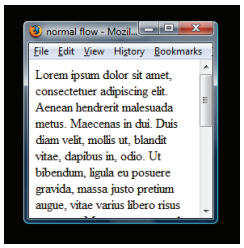


Figure 6. As we continue to make the browser window smaller, the paragraph reflows to fit the available width, while the text flows to the left in a downward manner, elongating the page.

Hot Tip A great way to gain an appreciation of normal flow is to simply create an HTML page with multiple paragraphs of text, and then size and resize the browser, watching how the flow behaves.

THE POWER AND PROBLEM OF FLOATS

Floats have become a very critical part of today's CSS approach to visual layout. At first glance, the concept is very simple. While floating elements can assist us in doing all kinds of visual magic, there are challenges inherent to using them, including the way they are rendered in browsers.

The Basic Intention of Floats

Interestingly, floats were not really envisioned as a means for creating columns, although that's often how we use them. Rather, they were developed to allow text to flow around another element which has been set to the left or the right, such as an image. This is something we see a great deal in design, as you can see in Figure 7.

Figure 7. Floating allows for text to flow around a floated box. In this case, the image is floated right, and the text flows around the image.

The more we use floats for layout, the more challenged we are to find ways of controlling and breaking the manipulation of flow. Floated elements are shifted to the left or right of the normal flow. In the case of Figure 8, the image element is shifted to the left and therefore the text flows to the right.

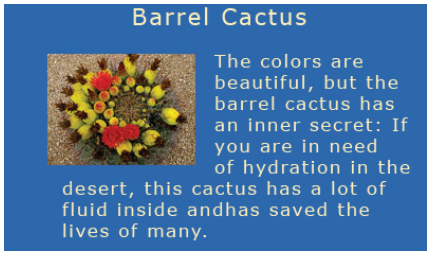


Figure 8. If the image is floated left, the text flows to the right.

It's important to note that the markup stays the same—it's only the CSS that changes. An example of the markup would be:

```
<h2>Barrel Cactus</h2>

<img src="b-cactus.jpg" alt="photo of blooming barrel cactus" />

<p>The colors are beautiful, but the barrel cactus has an inner secret: if you are in need of hydration in the desert, this cactus has a lot of fluid inside and has saved the lives of many.
```

The CSS values will of be different in terms of manipulating the location of the image and the text flow, in the first case, the image is floated right, and in the second, left. Other elements will simply flow around the floated element once a float has been applied.

The Power and Problem of Floats, continued

Float Behavior

Understanding how floats behave (or don't) within a browser environment is critical to troubleshooting a variety of issues, particularly when it comes to layout. While floats can be your best layout friend in many, many situations, knowing how floats behave in general will give you the upper hand!

The first thing to know is that floats are *not in the normal flow.* This means that other text and elements will continue to flow around the floated elements. In Figure 9, we see what happens if two elements are floated to the right and left. If not contained by another element that has an explicit width, the floated elements will continue to spread apart as the browser size increases, and come together as the browser size decreases.

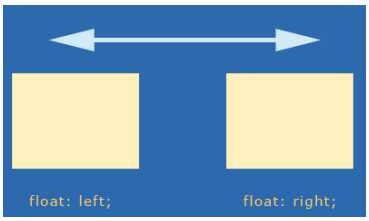


Figure 9. One box floated left, another right. If not contained, these boxes will continue to spread apart or get closer as the browser is resized.

Hot Tip If you've ever had text begin to unintentionally crawl up around the side of a floated box, the solution is float clearing (see "Clearing Floats" later in this reference).

In order to get floats to stack up nicely next to one another and avoid this behavior, we simply float them all in the same direction, typically left (Figure 10).

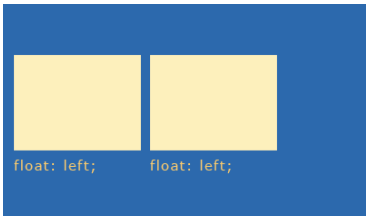


Figure 10. If we float elements in the same direction, they'll line up next to each other. This is one way of creating columnar designs.

What happens if there's not enough room within the browser (or the set width of a container element)? The subsequent floated element "falls off" the line and seeks the first available space (Figure 11).

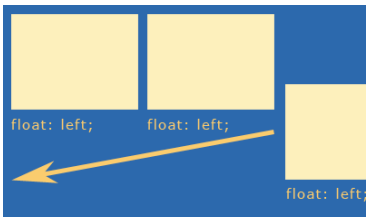


Figure 11. If an element cannot fit into the available space, the box will "fall off" the line and move to the first available space in the flow.

Float Behavior, continued

Hot Tip IE6 will drop the float down in if there is no space, incongruent with other browsers. If you're developing in a Firefox environment, be sure to test constantly in IE as well. This issue, referred to as "tolerance" can be remedied by modifying widths slightly to accommodate IE.

Floats can be contained by setting explicit widths on containing elements, and on the floated boxes themselves. This doesn't eradicate tolerance problems, but it gives you a more finite idea of how to mathematically arrange your elements (Figure 12).

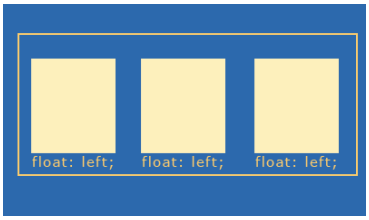


Figure 12. By containing elements and giving them a width, we can begin to create columns.

By containing elements, we can create contexts for interface components. If I wanted to create three columns as above, I could create three divs with specific widths, borders, padding and margins for each of the columns, and then set that within another div, which I would give an explicit width too, containing the three individual columns:

```
<div>
   <div>column 1</div>
   <div>column 2</div>
   <div>column 3</div>
</div>
```

Of course, using divs for everything is overkill, because many HTML elements actually have this sort of mechanism built right in! Consider an unordered list with three list items:

```
<ul>
   <li>column 1</li>
   <li>column 2</li>
   <li>column 3</li>
</ul>
```

As you can see, the is an element, and therefore *creates* a box. We can then style that box as a container for the list items residing within. If there's a logical reason to use a list, such as with navigation, or linked interface components, consider using semantic elements instead. This is called minimal markup and is an extremely helpful technique.

Hot Tip There's no such thing as "div" based design. That term is popular but highly inaccurate. In CSS, it's not that we use divs to create boxes, but that the element boxes are already there. Use divs only where necessary for containment and structure.

The Power and Problem of Floats, continued

Clearing Floats

So what happens when you want to stop the flow of text, or a floated box, and return to the normal flow? You have to clear your floats (Figure 13).

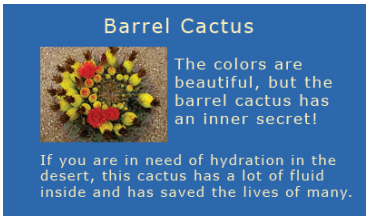


Figure 13. Clearing floats is the process of returning the next consecutive element to the normal flow.

In terms of floats for layout, if you want to be able to have content below floated elements, such as a traditional "footer" where site information will go, clearing is necessary. This allows the subsequent elements to return to the normal flow (Figure 14).

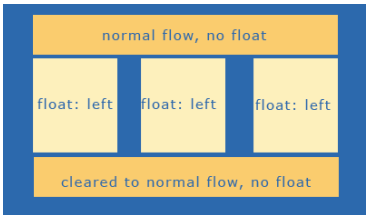


Figure 14. Once the float is properly cleared, we return to the normal flow, which is how we can create a footer beneath the floated columns.

There are numerous ways to clear floats. A very popular one is the creation of a clear class and the use of the property "clear" and a value of both:

```
.clear {clear: both;}
```

People will then use a div or a break element to clear the float:

```
<div class="clear"></div> or
<br class="clear" />
```

My preference of the two is the break element, as an empty div is a non-empty element with nothing inside it. While still slightly presentational, at least "break" has meaning in this case (Figures 15 and 16).

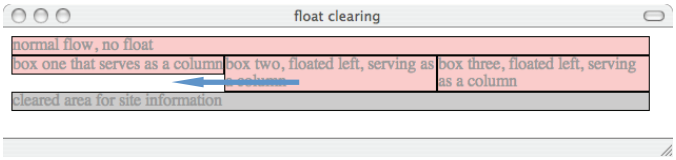


Figure 15. The float is properly cleared. Notice how the column to the left doesn't fill the entire available space—this is normal behavior. The important issue is that the bottom-most element is cleared and does not try to "creep up" into the available space.

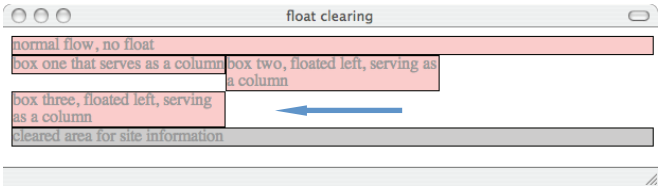


Figure 16. Even when a tolerance problem forces the far right column off the line, notice that it does not encroach upon the space above it. This is because it's not in the normal flow. Also, note that despite the tolerance issue, the bottom-most element remains clear of the floated elements.

CSS POSITIONING

CSS positioning is a powerful piece of CSS that allows us to position elements in a variety of ways. One of the problems with positioning is that it has not been well articulated, particularly relative positioning, over the years. It took me a long time to understand exactly what relative positioning really was, or was useful for. Therefore, I'll start with relative positioning and work from there.

Relative

If the fathers of CSS had been thinking about naming it a little more carefully relative positioning would be called "offset" positioning.

Essentials for relative positioning:

- A box with a position: relative; designation remains in the normal flow
- A box with position: relative; creates a new instance of normal flow within it
- Relative positioning is most useful for offsetting boxes or creating a positioning context (see *Relatively Absolute* later this reference)

Imagine three paragraphs of text in the normal flow (Figure 17). Using CSS I've hidden the visibility of the text in the second paragraph to make the visualization of relative positioning clearer.

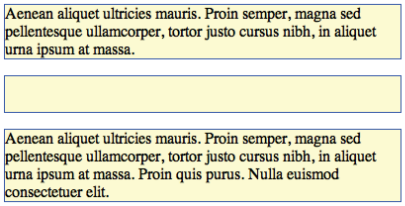


Figure 17. Three paragraphs in the normal flow.

Here's some CSS that offsets the paragraph using relative positioning:

```
p {width: 400px; border: 1px solid blue;}
p#rel {position: relative; top: 50px; left: 50px;}
```

The offset can be seen in Figure 18.

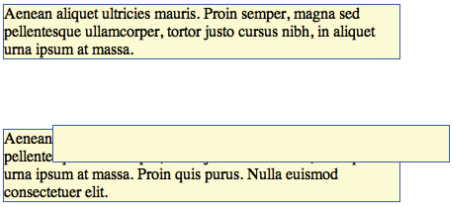


Figure 18. The second paragraph, positioned relatively.

At this point you're probably wondering, what is that paragraph being positioned to? This is why I say the term relative in this instance is so confusing. The answer is not what you'd expect. A relatively positioned element is positioned in relation to its location in the normal flow. In other words, the browser still "sees" the positioned element in the flow but on the design surface, it's visually moved from where it would have been. This is why the big space is left behind, almost a ghost of the paragraph. The browser interprets the element as being in the normal flow.

Absolute

Absolute positioning is a little clearer in its terminology, and perhaps a bit more logical in its behavior. An absolutely positioned element:

- Is removed from normal flow
- Is positioned in relation to
  - a positioned parent element
  - the root element of HTML
- Subsequent content flows into the now available space

CSS Positioning, continued

Here's the CSS used to position the box:

```
p#rel {position: absolute; top: 50px; left: 50px;}
```

If we briefly revisit normal flow (Figure 19), we can compare the before and after positioning.

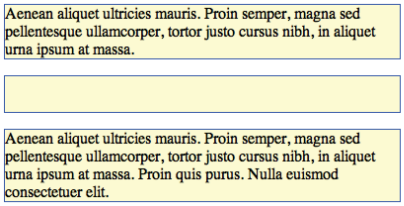


Figure 19. Normal flow, again.

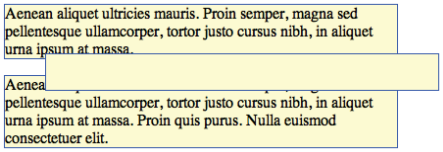


Figure 20. The second paragraph, positioned absolutely. Notice how the element is positioned absolutely from the top, left area (this can be the HTML element or another element, such as a div, that is positioned). The element is taken out of the normal flow, which is why the subsequent paragraph flows up into the available space, unlike relative positioning.

Fixed

Fixed positioning is actually a subset of absolute positioning and can be very useful in creating static elements on the design surface while other elements flow behind.

```
h1 {width: 100%; position: fixed; top: 0; left: 0;
border: 1px solid #999; background-color: #ccc;}
```

Elements that are fixed are done so in relation to the browser chrome (Figure 21).

Figure 21. The h2 gray box is fixed to the left and top of the browser chrome.

As I scroll the browser, the box remains fixed (Figure 22).

Figure 22. As I scroll, the text flows under the positioned box. True to its term, the box remains fixed.

Hot Tip

Fixed positioning isn't implemented in IE6, although IE7 does implement it, as does IE8. If we were viewing this in IE6, the fixed box would simply scroll off just as if it were a non-positioned element.

Creating a Positioning Context: "Relatively Absolute"

One technique that is very helpful to designers who are using a combination of floats and positioning to accomplish their layouts has been coined *"Relatively Absolute"*.

Creating a Positioning Context: "Relatively Absolute", continued

Remember, a relatively positioned box creates an instance of normal flow, while it itself remains in the normal flow of the document. Therefore, creating areas that stay together as a unit can be accomplished by creating a containing element that is relatively positioned without *offset values*:

```
ul#sub-nav {position: relative; width: 350px; padding:
40px; border: 1px solid black;}
```

I then styled the list items and links:

```
li {list-style-type: none;}
a {text-decoration: none;}
li#home {position: absolute; width: 100px; left: 10px;
border: 1px solid green;}
li#products {position:absolute; width: 100px;
absolute; left: 120px; border: 1px solid red;}
li#contact {position: absolute; width: 100px; left:
240px; border: 1px solid blue;}
```

This CSS removes the bullets from the list items. I then absolutely position each list item box as I see fit, resulting in a box in the normal flow with three links (Figure 23).

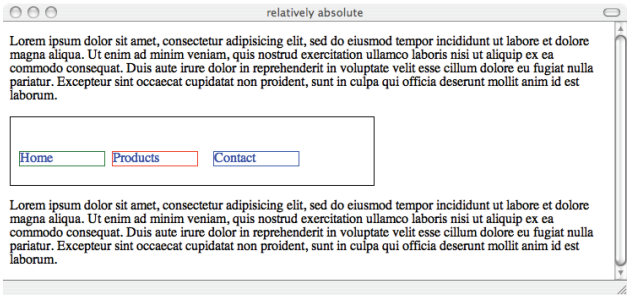


Figure 23. Relatively absolute: Using a combination of positioning to keep like items together within normal flow.

If the browser is resized, everything will flow normally, but the absolutely positioned elements will retain their position within the relative box.

Stacking Order: z-index

An important property in CSS is z-index. This property provides us a third visual dimension of depth with any positioned element (note that the element must be positioned to accept a z-index). This way, we can stack boxes and have them overlap as we'd like.

In Figure 24, I created 3 boxes and used the following CSS to cause them to overlap naturally, without using a z-index. This is the default stacking order behavior when no z-index is involved.

```
div#sample1 {position: absolute; left: 10px; padding:
50px; background-color: blue;}
div#sample2 {position: absolute; left: 65px; padding:
50px; background-color: red;}
div#sample3 {position: absolute; left: 110px; padding:
50px; background-color: yellow;}
```

Figure 24. Default stacking order.

I can reverse the stack order using the z-index:

```
div#sample1 {position: absolute; left: 10px; padding:
50px; background-color: blue; z-index: 3;}
div#sample2 {position: absolute; left: 65px; padding:
50px; background-color: red; z-index: 2;}
div#sample3 {position: absolute; left: 110px; padding:
50px; background-color: yellow; z-index: 1;}
```

Stacking Order: z-index, continued

Figure 25 shows the results.

Figure 25. Reversing the stack using z-index.

It is important to understand that the higher the number, the "closer" to the screen the item appears. Also, you do not have to use sequential numbers. In fact, designers who want to be very certain that a given element is always on top of a stack might use numbers significantly higher than those lower in the stack:

```
div#sample1 {position: absolute; left: 10px; padding:
50px; background-color: blue; z-index: 2;}

div#sample2 {position: absolute; left: 65px; padding:
50px; background-color: red; z-index: 10;}

div#sample3 {position: absolute; left: 110px; padding:
50px; background-color: yellow; z-index: 3;}
```

Figure 26 shows the results, with the red box coming to the foreground.

Figure 26. Bringing a box to the foreground with z-index.

Another point to make regarding z-index is that each positioning context allows for a new instance of z-index. This is why you might see several instances of the same numbers that result in different stacking orders—it just depends on what the designer or developer is attempting to achieve.

CORE CSS WRAP-UP

The goal of the Core CSS series is to ensure that readers are empowered to understand the areas of CSS that are unclear and confusing, as well as give insight into the way browsers behave. Understanding CSS and how browser behaviors influence the work you do empowers you to envision, create and code to a high quality that will allow the sites and applications you design to be manageable, scalable and attractive, too.

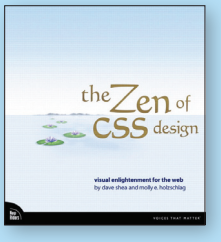
ABOUT THE AUTHOR

Molly E. Holzschlag

Molly E. Holzschlag is a well-known Web standards advocate, instructor, and author. She is an Invited Expert to the W3C, and has served as Group Lead for the Web Standards Project (WaSP). She has written more than 30 books covering client-side development and design for the Web. Currently, Molly works to educate designers and developers on using Web technologies in practical ways to create highly sustainable, maintainable, accessible, interactive and beautiful Web sites for the global community. She consults with major companies and organizations such as AOL, BBC, Microsoft, Yahoo! and many others in an effort to improve standards support, workflow, solve interoperability concerns and address the long-term management of highly interactive, large-scale sites. A popular and colorful individual, Molly has a particular passion for people, blogs, and the use of technology for social progress.

Web Site
http://www.molly.com

RECOMMENDED BOOK



Proving once and for all that standards-compliant design does not equal dull design, this inspiring tome uses examples from the landmark CSS Zen Garden site as the foundation for discussions on how to create beautiful, progressive CSS-based Web sites.

BUY NOW
books.dzone.com/books/zencss

Get More FREE Refcardz. Visit refcardz.com now!

Upcoming Refcardz:

Using XML in Java

Core Mule 2

Getting Started with Equinox and OSGi

SOA Patterns

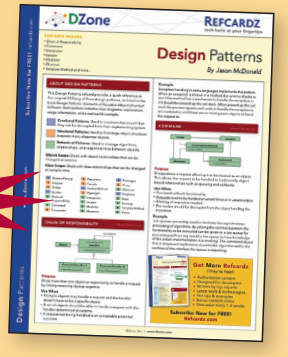
Getting Started with EMF

Available:

Getting Started with Hibernate Search
Core Seam
Essential Ruby
Essential MySQL
JUnit and EasyMock
Getting Started with MyEclipse
Spring Annotations
Core Java
Core CSS: Part II
PHP
Getting Started with JPA
JavaServer Faces
Core CSS: Part I
Struts2
Core .NET
Very First Steps in Flex
C#
Groovy
NetBeans IDE 6.1 Java Editor

Visit refcardz.com for a complete listing of available Refcardz.

FREE



Design Patterns
Published June 2008

DZone

DZone communities deliver over 4 million pages each month to more than 1.7 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheatsheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
1251 NW Maynard
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-23-3
ISBN-10: 1-934238-23-6
50795
9 781934 238233
$7.95

DZone Refcardz

CONTENTS INCLUDE:

CSS3: Basics

By: Jason Cranford Teague

Cascading Style Sheets (CSS) standard is a stylesheet language that allows web developers to add visual and temporal styles to Web pages using mark-up languages—typically HTML. CSS3 (or CSS Level 3) is the latest iteration of the CSS specification as defined by the World Wide Web Consortium's CSS Workgroup. It builds on features already defined in the CSS1, CSS2, and CSS2.1 specifications. CSS4 is also currently under development by the CSS Workgroup but its features are still experimental.

Because it is a progression of the previous standards rather than a replacement, there are few changes from the previous versions, mostly additions. There are a few cases, however, where older syntax is being depreciated in favor of new syntax. Those cases will be highlighted in this document.

Browser Support

CSS3 is a standard that is supported in part by all modern web browsers. However, web browsers have gradually implemented CSS3 over the last several years, so older browsers may not have properties implemented, have the properties fully implemented, or may be using a browser extension. Universal browser support should be assumed unless otherwise indicated.

Rendering Engines and Browser Extensions

The *rendering engine* (sometimes called layout engine or web browser engine) is a software component used to display and interpret HTML, CSS, JavaScript, or other code used to create web pages.

Extension	Rendering Engine	Browsers Include
-webkit-	Webkit	Webkit
-moz-	Mozilla	Firefox
-o-	Presto	Opera to v14
-ms-	Trident	Internet Explorer

Some of the features in CSS3 (and CSS4) are supported using *browser extensions*, which were used to test experimental features in rendering engines in advance of their inclusion in the full CSS specification. Although referred to as *browser* extensions, they are actually prefixes based on the specific company of origin or the rendering engine used by the browser, rather than the specific, name-brand browser.

Using Browser Extensions

To ensure maximum backwards - and cross-browser compatibility, always include the declaration using all possible browser extensions available for the specific property along with the CSS3 syntax.

```
.mybox {  -webkit-transform: scale(2);
          -moz-transform: scale(2);
          -o-transform: scale(2);
          -ms-transform: scale(2);
          transform: scale(2);
```

It is a good idea to place the W3C version of the property last in order, so that it will override the older browser extension versions if recognized.

Hot Tip

Will I always need browser extensions?
Eventually, after the property is added to the CSS standard, the browser extension is no longer needed, but it could take several years before all legacy browsers have been updated. In the meantime, the website caniuse.com provides an easy to reference chart of supported properties and extensions.

NEW SELECTORS IN CSS3

Selectors are used to indicate the element to be styled and the conditions by which it should be styled. One of the key areas that CSS3 has added to the standard is new pseudo-classes and pseudo-elements, allowing you more precise control over web designs.

Pseudo-class	Pattern Matching
:root	Styled if the element is parent element of the document. This is synonymous with the <html> tag, but has higher specificity.
:empty	Styled if the element has no children.
:last-child	Styled if the element is the last child of another element.
:only-child	Styled if the element is the only child of the parent element.
:first-of-type	Styled if the element is the first of its selector type in the parent element.
:last-of-type	Styled if the element is the last of its selector type in the parent element.
:only-of-type	Styled if the element is the only child of its type in the parent.
:nth-child(#)	Styled if the element is the specific nth child (#) of the parent.
:nth-last-child(#)	Styled if the element is the specific nth child (#) of the parent from the bottom.
:nth-of-type(#)	Styled if the element is the specific nth child (#) of the specific selector type in the parent.
:nth-last-of-type(#)	Styled if the element is the specific nth child of the specific selector of the parent from the bottom.

New Pseudo-Classes

Pseudo-classes are used to reference specific elements on the page that are not specific HTML selectors.

Structural Pseudo-classes

Structural Pseudo-classes are used to reference specific elements based on their location in the DOM (Document Object Model).

UI Pseudo-classes

UI pseudo-classes are used to reference specific elements based on their use to input information by the user.

Pseudo-class	Pattern Matching
:enabled	Styled if the element can be activated (selected, clicked on, or accept text input) or accept focus
:disabled	Styled if the element has the value disabled and cannot be selected, clicked on, accept text input, or accept focus.
:check	Styled if the element is a radio, check box, or option form element that is set to an on state.
:indeterminate	Styled if the element is an input checkbox that has had its indeterminate set to true using JavaScript
:default	Styled if the element is the default amongst a group of similar elements. Not supported in Internet Explorer.
:valid	Styled if the element is an input or form that contains content that validates correctly according to the input type setting.
:invalid	Styled if the element is an input or form that contains content that does not validate correctly according to the input type settings.
:in-range	Styled if the element is an input with a value range set and a value within that range.
:out-of-range	Styled if the element is an input with a value range set and a value out of that range.
:required	Styled if the element is an input with the required attribute.
:optional	Styled if the element is an input without the required attribute.
:read-only	Styled if the element cannot be edited by the user. **Not widely supported.**
:read-write	Styled if the element can be edited by the user. **Not widely supported.**

Other Pseudo-Classes

Pseudo-class	Pattern Matching
:not(x)	Styled if the element is not x where x is a valid selector.
:target	Styled if the element is the target of a link with the specified URI anchor as part of its href.

New Pseudo-Elements

Pseudo-elements are used to reference elements based on their use within the document. Four new ones have been added in CSS3, but have not been widely implemented yet.

Selector	Description
::value	Styled if the element is the value in an input element. **Not widely supported.**
::choices	Styled if the element is a choice in a form element. **Not widely supported.**
::repeat-item	Styled if the element is a single item from a repeating sequence. **Not widely supported.**
::repeat-index	Styled if the element is the current item of a repeating sequence. **Not widely supported.**

Change in pseudo-element Syntax

The only major change to the actual syntax of CSS is with pseudo-elements, which now uses two colons (::) instead of a single colon.

Example using CSS2.1

```
p:first-letter { font-size: 2em; }
```

Example using CSS3:

```
p::first-letter { font-size: 2em; }
```

Hot Tip

Do I have to use the double colon syntax?
The older syntax is still recognized by all browsers, and will likely be for several years. For best backwards compatibility, it is recommended to use the old syntax or use the old and new syntax redundantly.

COLOR STYLES

There are new ways to define color values as well as gradients and to set the opacity of individual color fills.

HSL

Hue, saturation, and lightness are common ways to define a color value, and are now available to web designs.

Value	Description
Hue	Value from 0-360 indicating the value of the color's hue on a color wheel.
Saturation	Styled if the element is the target of a link with the specified URI anchor as part of its href.
Lightness	Value from 0 100% indicating the color tone from black (0%) to white (100%) with 50% being the regular color.

Changing Hue

hsl(0, 100%, 50%)
hsl(90, 100%, 50%)
hsl(180, 100%, 50%)
hsl(270, 100%, 50%)

Changing Saturation

hsl(120, 100%, 50%)
hsl(120, 75%, 50%)
hsl(120, 50%, 50%)
hsl(120, 25%, 50%)
hsl(120, 0%, 50%)

Changing Lightness

hsl(240, 100%, 100%)
hsl(240, 100%, 75%)
hsl(240, 100%, 50%)
hsl(240, 100%, 25%)
hsl(240, 100%, 0%)

HSL is not available in IE until version 9.

Hot Tip

Why HSL?
Using HSL to define colors makes it easier to darken or lighten them by simply changing the lightness value rather than having to recalculate the entire value. This is especially useful for defining link colors, which are often simply different tones of the same color hue.

Opacity

Opacity is used to make an entire element translucent. This includes all of the elements children.

Property	Values	Description
opacity	<number>	Specifies a value between 0 (transparent) and 1 (opaque) for the element's opacity.

50% opacity would be:

```
element1 { opacity: .5 }
```

Color Opacity with Alpha Values

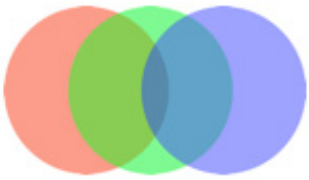
The alpha of a color refers to its opacity as a value between 0 (transparent) to 1 (opaque). CSS3 adds the ability to control opacity of individual color fills by adding an alpha value to RGB and HSL color values. This is done by changing the syntax for the color declarations slightly to rgba() or hsla().

Solid Color:

```
rgb(255, 0, 0);     hsl(0, 100%, 50%);
rgb(0, 255, 0);     hsl(120, 100%, 50%);
rgb(0, 0, 255);     hsl(240, 100%, 50%);
```

Color set to 50% Opacity:

```
rgba(255, 0, 0, .5);          hsla(0, 100%, 50%, .5);
rgba(0, 255, 0, .5);          hsla(120, 100%, 50%, .5);
rgba(0, 0, 255, .5);          hsla(240, 100%, 50%, .5);
```



Alpha values are not available in Internet Explorer until version 9.

Hot Tip

Where can I use the alpha value?
The alpha value can be used anywhere color is declared, including color (text), background, border, and shadows.

Gradients

A gradient allows you to set two or more color stop values that will then fill with color, smoothly transitioning between. Gradients can only be applied to backgrounds.

Linear Gradient

Selector	Description
Angle	Value in degrees (45deg) or a keyword (top, right, bottom, and/or left) to set the angle of the gradient within the element
Color Stop	The color value and location of a particular color stop. Location is optional

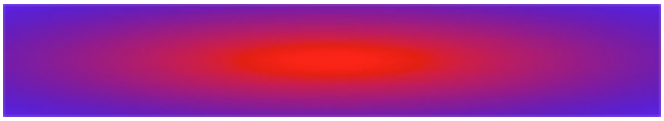
Currently, linear gradients still rely on browser extensions to ensure full cross-browser compatibility, but the syntax is identical.

```
background: -moz-linear-gradient(top, rgba(255,0,0,1) 0%,
rgba(0,0,225,1) 100%);
background: -webkit-linear-gradient(top, rgba(255,0,0,1)
0%,rgba(0,0,225,1) 100%);
background: -o-linear-gradient(top, rgba(255,0,0,1)
0%,rgba(0,0,225,1) 100%);
background: -ms-linear-gradient(top, rgba(255,0,0,1)
0%,rgba(0,0,225,1) 100%);
background: linear-gradient(top, rgba(255,0,0,1)
0%,rgba(0,0,225,1) 100%);
```

Sample Radial Gradient

Value	Description
Shape/Size	Keywords: circle or ellipse.
Origin Position	The position of the center of the radial gradient using distance, percentage, or keywords (top, right, bottom, left, or center).
Color Stop	The color value and location of a particular color stop. Location is optional

```
background: radial-gradient(center, ellipse cover,
rgba(255,0,0,1) 0%,rgba(0,0,225,1) 100%);
```



Currently, radial gradients rely on browser extensions to ensure full cross-browser compatibility, but the syntax is the same.

```
background: -moz-radial-gradient(center, ellipse cover,
rgba(255,0,0,1) 0%, rgba(0,0,225,1) 100%);
background: -webkit-radial-gradient(center, ellipse cover,
rgba(255,0,0,1) 0%,rgba(0,0,225,1) 100%);
background: -o-radial-gradient(center, ellipse cover,
rgba(255,0,0,1) 0%,rgba(0,0,225,1) 100%);
background: -ms-radial-gradient(center, ellipse cover,
rgba(255,0,0,1) 0%,rgba(0,0,225,1) 100%);
background: radial-gradient(center, ellipse cover,
rgba(255,0,0,1) 0%,rgba(0,0,225,1) 100%);
```

Hot Tip

How do I calculate gradients?
Since gradients will need to use browser extensions to be fully cross-browser compatible, complex gradients can be tedious to calculate. Use Colorzilla's Ultimate CSS Gradient Generator tool (http://www.colorzilla.com/gradient-editor/) to design your gradients and output the full cross-browser code.

FONT AND TEXT STYLES

Font styles affect the appearance of individual letters, while text styles generally are used to affect an entire block of text.

Property	Values	Description
font-size-adjust	none | auto | <number>	Specifies how a font should be sized based on a ratio of its lowercase letters, rather than uppercase.
font-stretch	normal | ultra-condensed | extra-condensed | condensed | semi-condensed | semi-expanded | expanded | extra-expanded | ultra-expanded	Specifies the version of the typeface to be used based on common type values for stretch.

Property	Values	Description
text-overflow	clip | ellipsis | <string>	One or two values that specify how overflow text should be treated in an inline element, either adding an ellipsis ( ) or a user defined string of characters.
text-shadow	[<color> <offset-x> <offset-y> <blur-radius>] #	Adds one or more shadows underneath all styled text with the specified color, offset, and blur.
text-align-last	auto | start | end | left | right | center | justify	Specifies the text alignment of the last line in a block of text. **Not widely supported.**
text-decoration-color	<color>	Specifies the color value for the text decoration. **Not widely supported.**
text-decoration-line	none | [underline || overline || line-through || blink]	Specifies the position of the text-decoration. One or more values can be specified, separated by spaces. **Not widely supported.**
text-decoration-skip	none | [images || spaces || link || all]	Specifies element types to skip over when adding a text decoration. **Not widely supported.**
text-decoration-style	solid | double | dotted | dashed | wavy	Specifies the style of the text decoration being used. **Not widely supported.**
word-wrap	normal | break-word	Specifies whether text is allowed to break within words or not.

@font-face

Although technically a part of the CSS2 specification, @font-face is commonly associated with CSS3, where it has come into its own with the availability of webfonts that allow the syntax to work.

Property	Values	Description
font-family	' <text-string>'	Name for the font to use in your font-family declarations. This name can be anything you want, but make sure to put it in quotes.
src	<url>	Path for the font files location on your server and the format the font is using, typically: truetype, opentype, eot, svg, and woff.
font-variant	<font-variant>	**Optional.** Specifies whether the font is normal or small-caps.
font-stretch	<font-stretch>	**Optional.** Specifies whether the font is a particular stretch style, such as condensed or expanded.
font-weight	<weight>	**Optional.** Specifies whether the font is a particular weight, such as bold, bolder, lighter, or 100-900 .
font-style	<style>	**Optional.** Specifies whether the font is a particular italic or oblique.

The basic @font-face syntax looks like this:

```
@font-face {
          font-family: 'font name';
          src: url('fonts/fontfile.ttf') format('truetype');
font-stretch: normal;
          font-style: italic;
          font-weight: bold;
}
```

Then use the font name in your CSS:

```
p { font-family: 'font name', times, serif; }
```

Cross-Browser Webfonts

Webfonts are simply font files that can be used by a particular browser to display type. However, different browsers use different file formats, so you will need different sources for each:

```
@font-face {
          font-family: 'font name';
          src:      url('fonts/fontfile.eot?') format('embedded-
opentype'),
                    url('fonts/fontfile.woff') format('woff'),
                    url('fonts/fontfile.ttf') format('truetype'),
          url('fonts/fontfile.svg#fontname') format('svg');
}
```

The browser will download the first file format it recognizes and use that.

Hot Tip Note the question mark (?) after the .eot in that url. This is to help overcome a bug in older versions of Internet Explorer, which would have kept this file from loading.

Hot Tip **Webfont icons**
You can use @font-face to load a font with icons instead of letters, numbers, and other symbols, and then use that instead of images to add icons in your interface For a demo, visit: http://cdpn.io/acsej

BACKGROUND STYLES

All elements have a background area that can be filled with a solid color, gradient or image. Although part of the CSS2 specification, with CSS3 you can now designate multiple background images on the same element, which has necessitated some slight changes to the properties syntax.

Property	Values	Description
background	[<bg-layer>] #	Specify multiple background colors or images on the same element in a comma-separated list.
background-attatchment	[inherit || scroll || fixed ||local] #	Specifies how the background scrolls with the element. `local` will only scroll with its content, and not with the parent element.
background-clip	[inherit || border-box || padding-box || content-box] #	Specifies whether an element's background, either the color or image, extends underneath its border. This is important when the border is transparent.
background-image	[<url>] #	Specify the path to one or more background images on the same element in a comma-separated list.
background-origin	[inherit || border-box || padding-box || content-box] #	Specifies the location of the top right corner of the background in relation to the element box.
background-position	<position> #	Specifies the initial position of the background in relation to its origin.
background-repeat	[inherit || repeat-x ||repeat-y || repeat || space || round || no-repeat] #	Specifies how the background is repeated in the element box.
background-size	[<length> || <percentage>|| auto || cover || contain] {1,2} #	Specifies a size for the background image. Two values will set width and height. `cover` fills the entire element, `contain` shows the entire image in the background.

Box-Shadow

Drop shadows are a common visual effect used to create a feeling of depth on a 2D screen.

Property	Values	Description
box-shadow	none | [inset? && [<offset-x> <offset-y> <blur-radius>? <spread-radius>? <color>?]]#	Specify one or more shadows of any color in a comma-separated list, behind and/or inside (inset) the element's box. You can also control the horizontal and vertical offset the sharpness of the shadow (blur) and the distance of the solid area from the edge before the shadow fade begins (spread).

```
box-shadow: 2px 3px 5px rgba(0, 0, 0, .5), 0 0 20px rgba(023, 234, 120, .75) inset;
```

Hot Tip box-shadow is only supported in IE 9+ and some older versions of Firefox, Safari, and Chrome may require the browser extension to work.

Border Image

CSS3 introduces the ability to use an image file to create customized border styles. To do this, you will first need to create a bit map image, saved in a web-compatible format (e.g., PNG, GIF, or JPEG).

The CSS slices the image into nine areas: the four corners, the four sides, and the center. The corners of the image will be used at the corners of the box, and each of the four sides are tiled horizontally or vertically

Then, use the properties below to set the image as the elements border.

Property	Syntax	Description
border-image-outset	[<length> | <number>]{1,4}	One to four values that specify how far beyond the edge of the box the border image can extend without being clipped.
border-image-repeat	[stretch | repeat | round]{1,2}	One or two values to specify how the image border behaves horizontally and vertically.
border-image-slice	[<number> | <percentage>]{1,4} && fill?	One to four values that specify the distance sliced into the image from its outer top, right, bottom, and left sides to create the image border. If fill is included, the central part of the border image is used as the element's background image.
border-image-source	<url>	Specifies the path to the image file being used as the border.
border-image-width	[<length> | <percentage> | <number> | auto]{1,4}	One to four values that specify the offset to use for dividing the border image in nine parts: the top-left corner, central top edge, top-right-corner, central right edge, bottom-right corner, central bottom edge, bottom-left corner, and central right edge.
border-image	<'border-image-source'> || <'border-image-slice'> [/ <'border-image-width'> | / <'border-image-width'>? / <'border-image-outset'>]? || <'border-image-repeat'>	Shorthand to specify multiple values for the border image.

In the border-image shorthand, the URL for the image file, the width and the height that the border should slice into the image from its edges, and then how the border image should repeat.

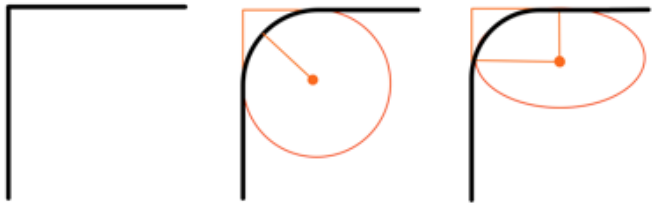
```
border-image:url("flora.png") 30 30 repeat;
```

The border image property is still not fully supported as a CSS standard, but is available in Firefox, Safari, Chrome, and Opera with the appropriate browser extensions:

```
.floralborder {
  border: 30px solid transparent;
  -moz-border-image:url("flora.png") 30 30 repeat;
  -webkit-border-image:url("flora.png") 30 30 repeat;
  -o-border-image:url("flora.png") 30 30 repeat;
  border-image:url("flora.png") 30 30 repeat;
}
```

Border Radius

Border radius allows for the inclusion of rounded corners. Although referred to as *border radius*, this property really rounds the corners of the box, whether there is a specific border on it or not. The radius can be a single value per corner for a circular radius or two values per corner for an elliptical radius. The border radius can be applied to all four corners or each individually as desired.



Property	Syntax	Description
border-radius	[<length> | <percentage>] {1,4} [/ [<length> | <percentage>]{1,4}]?	One to four values that specify the radius of the corners' curvature on each corner (top left, top right, bottom right, bottom left). Optionally, a slash (/) and one to four more values can be included to create elliptical curves on each of the four corners.

Property	Syntax	Description
border-top-left-radius, border-top-right-radius, border-bottom-right-radius, border-bottom-left-radius	[<length> | <percentage>] [/ [<length> | <percentage>]]?	A single value that specifies the radius for a specific corner.

```
border-radius: 20px;
```

```
border-radius: 20px 10px 30px 40px;
```

```
border-radius: 20px/10px;
```

```
border-radius: 20px 10px/10px 20px;
```

Hot Tip

Content in rounded corners is not cropped
Border radius does not crop the content inside the box, so any image or text in a corner will still appear. However, it will crop the background — including background images — allowing you to create a variety of shapes, including circles and ovals.

BOX & UI STYLES

Overflow allows you to control how content is viewed if it does not fit in the area defined. However, the new `overflow-x` and `overflow-y` allow you to set how overflowing content is controlled independently horizontally and vertically, while the `resize` property allows the user to change the size of the element's box.

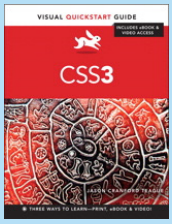
Property	Syntax	Description
overflow-y	visible | hidden | scroll | auto	Specifies how vertically overflowing content in an element should be treated for scrolling.
overflow-x	visible | hidden | scroll | auto	Specifies how horizontal overflowing content in an element should be treated for scrolling.
resize	none | both | horizontal | vertical	Specifies whether an element can be resized by the user.

ABOUT THE AUTHOR

Jason (www.jasonspeaking.com) combines creative and technical know-how to help people communicate online. He has worked with businesses and organizations, including USA TODAY, Marriott International, AOL, Virgin, Bank of America, The Aspen Institute, and The Solar Energy Industry Association. Jason is well know for being able to explain complex technical concepts to non-technical audiences. His most recent book is CSS3 Visual Quickstart Guide available in finer book stores everywhere.

Jason blogs regularly for GeekDad (www.geekdad.com),

RECOMMENDED BOOK



With CSS3: Visual QuickStart Guide readers can start from the beginning to get a tour of the stylesheet language or look up specific tasks to learn just what they need to know. This task-based, visual reference guide uses step-by-step ivnstructions, and plenty of screenshots to teach beginning and intermediate users CSS. Best-selling author Jason Cranford Teague takes readers step-by-step through today's CSS essentials and provides extensive coverage of CSS3 and CSS 2.1 techniques. The book outlines what can be done with CSS3 now and how the latest browsers have implemented many of the new features. Both beginning users, who want a thorough introduction to CSS, and more advanced users, who are looking for a convenient reference, will find what they need in straightforward language and through readily accessible examples.

Buy Here

DZone, Inc.
150 Preston Executive Dr.
Suite 201
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-936502-83-7
ISBN-10: 1-936502-83-6
50795
9 781936 502837
$7.95

Version 1.0