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

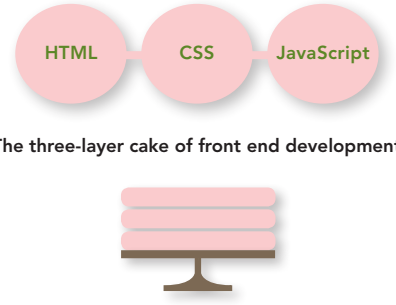
A Separate Piece, continued

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).

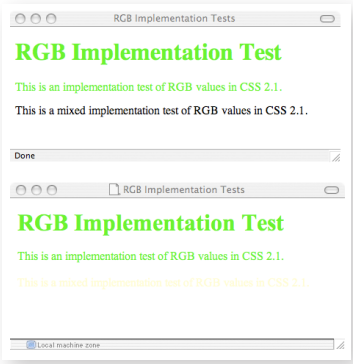


Figure 2. Error handling in browsers. In the first instance, if there's a mistake in the rule, the browser simply drops the erroneous rule and reverts to the browser style. In the second instance, the browser instead makes an attempt to find a nearest value and apply the color to nearly disastrous results.

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).

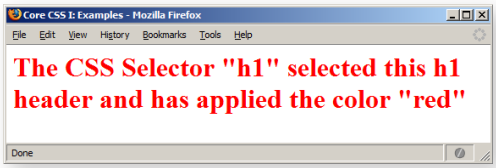


Figure 3. Using the h1 selector to apply the color "red" to a corresponding h1 element

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):

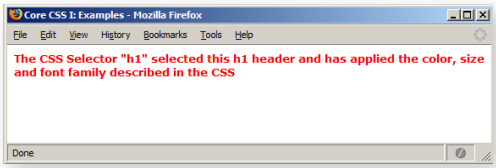


Figure 4. Adding additional declarations to the rule.

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

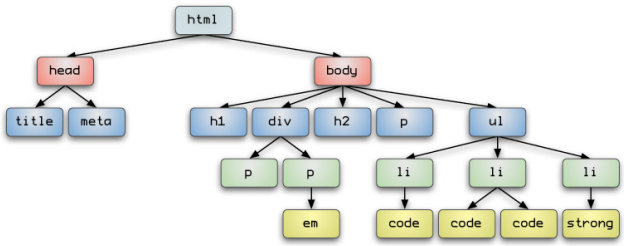


Figure 5. Inheritance. Imagine pouring a bucket of blue paint onto the body element. Because color is an inheritable property, all text descending from body will be blue until another style overrides the inherited color.

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.

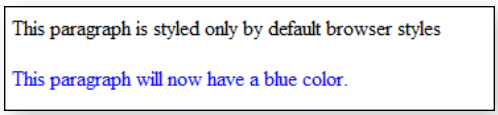


Figure 6. Applying style to a discrete element using inline style.

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

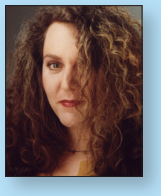
It should be clear that CSS has nuances that only time and experience can reveal. Well, that and good references! Look for the remaining "Core CSS" series and go into depth with selectors, the box model, floats, positioning and the z-index. Sound exciting? I think so too!

More **Core CSS** Refcardz:

Core CSS: Part II—October 2008

Core CSS: Part III—November 2008

ABOUT THE AUTHOR



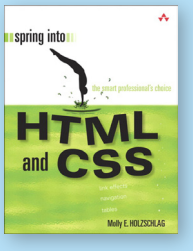
Molly E. Holzschlag

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