

**Sample Hands-On-Training™ Chapter – Review Copy Only**

Copyright ©2000-2003 by lynda.com, Inc. All Rights Reserved.  
Reproduction and Distribution Strictly Prohibited.

This electronically distributed Hands-On-Training™ document is for review purposes only and is intended for on-screen viewing only. Any printing, reproduction, copying, distribution, and/or transmission of this document are strictly prohibited without written consent from lynda.com, Inc.

**Contact Information**

Garo Green ([garo@lynda.com](mailto:garo@lynda.com))

Director, Publications

lynda.com, Inc.

PO Box 789

Ojai, California 93024

Phone: 805-646-7076

Fax: 805-640-9607

**Notice of Rights**

All rights reserved. No part of this book may be reproduced or transmitted in any form by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of the lynda.com, Inc. For information on getting permission for reprints and excerpts, contact [garo@lynda.com](mailto:garo@lynda.com).

**Notice of Liability**

The information in this book is distributed on an “As Is” basis, without warranty. While every precaution has been taken in the preparation of the book, neither the author nor Peachpit Press shall have any liability to any person or entity with respect to any loss or damage caused or alleged to be caused directly or indirectly by the instructions contained in this book or by the computer software and hardware products described in it.

**Trademarks**

Hands-On-Training is a registered trademark of lynda.com, Inc. Macromedia is a registered trademark. Macromedia Dreamweaver and Dreamweaver, and Macromedia Fireworks and Fireworks are registered trademarks of Macromedia, Inc., in the U.S. and/or other countries. QuickTime and the QuickTime logo are trademarks used under license. The QuickTime logo is registered in the U.S. and other countries.

# Cascading STYLE SHEETS

## INTRODUCTION

- > What Is a Style Sheet?
- > Declarations & Selectors
- > Using Comment Tags
- > Classes
- > IDs
- > Pseudo-Classes
- > Pseudo-Elements
- > Typography
- > Absolute Positioning
- > Layering Text & Images
- > WYSIWYG Editors

# 23

CSS (**C**ascading **S**tyle **S**heets) offers a means to separate “style” from “structure.” What exactly does that mean? At its inception, HTML was only intended to be a structural markup language. Structure is independent of the way something looks. Structure defines whether text on the page is a headline or body copy, whether a list should be alphabetized or numeric, and so on. Structure is very important when it comes to accessibility issues (when someone who is visually impaired uses an audio reader, for example) or when a web page is shown on a device other than a browser. Because HTML originally offered no means to style a document (change the font, the color, or format text and pictures precisely), many tags were introduced over time that dealt with the appearance of document. Many times, these tags that dealt with style interfered with structure.

To give an example of this interference between style and structure, imagine a headline on a web page. The HTML code could instruct this text to be a headline by virtue of an **H** tag. The **H1** tag will cause the type to be set in a large, bold, black, generic Times Roman font. Instead of using the **H1** tag, designers who didn't want their headline to look so generic might use the **FONT** tag instead. Through this tag, they could tell the text to be large, bold, and to be any font or color they chose. While both techniques produced something that looked to the end user like a headline, the **H1** approach kept the structure intact, while the **FONT** tag changed the appearance and didn't identify what the structure was.

Enter style sheets to the rescue. Style sheets can change the appearance of your web pages without affecting the structure. In other words, you can have your headline maintain its structure and have its style, too.

## IMPLEMENTATION VERSUS STANDARDS

While the **World Wide Web Consortium (W3C)** strongly advocates that you use style sheets, and offers a great deal of documentation and support, there is a problem with using them. The problem is that browsers “implement” style sheets with varying degrees of success. By this, I mean not all browsers interpret style sheets the same way. There are still major implementation differences between Netscape and Explorer, not to mention that different (especially older) versions of the same browser support style sheets differently.

The promise that style sheets hold is a designer’s best dream. Designers want pixel-precise control, better layout options, and more consistent and robust typographic control. HTML was designed to separate the structure of a document from the presentation so that HTML pages could be accessed by anyone, by any device, or on any browser. While this was started out as a goodwill theory, the passing of Section 508, which is described in detail in Chapter 8, “Accessibility Issues,” makes CSS a mandatory practice for many sites.

Sadly, until the browsers support style sheets in a uniform manner, that promise will not be totally met. While using style sheets is always better than using no style sheets, backward compatibility can pose a strong deterrent for some.



This is the site for [alistapart.com](http://alistapart.com), shown in a browser that properly supports Cascading Style Sheets.



Here is the same site shown in an older version of the browser that doesn't support style sheets. All the information is still here, but it doesn't look very good.

What this means is that you need to make a decision as to whether style sheets are appropriate for your site. If you know that your audience is likely to be using a specific browser, testing style sheet implementation is easy. If you think your audience will be viewing your site from older browsers, you might have to do a considerable amount of testing to ensure that you're getting the desired results from your code.

Be sure to check out WaSP (**Web Standards Project**) at <http://www.webstandards.org>. You'll find numerous articles on the benefits of using style sheets, as well as compelling arguments in their favor. The only reason style sheets pose any controversy at all is their lack of consistent implementation by browsers.

Another fantastic resource is found at [http://www.alistapart.com](http://www alistapart.com). Here you'll find numerous articles about style sheets, accessibility, and design with lots of practical code examples, and great ideas.

Cascading Style Sheets will be described in this chapter from a practical perspective in terms of what is supported and what is promised. However, in the end, choosing to use CSS on your pages will be a decision that you will have to make based upon your intended audience and the browsers you expect them to be using.

## note

### READ THE SPECS!

The W3C has released two recommendations for Cascading Style Sheets: CSS1 and CSS2. The CSS1 recommendation was formalized on December 17, 1996 and revised on January 11, 1999. The CSS1 recommendation contains about **50** properties. The CSS2 recommendation was formalized on May 12, 1998. The CSS2 recommendation contains about **120** properties, including those of the CSS1 recommendation. In this chapter I will refer to them collectively as CSS unless there is a reason to make a distinction between the recommendations.

If you want to read about the recommendations, you will find them at:

#### **W3C's CSS Recommendations**

<http://www.w3.org/TR/REC-CSS1>

<http://www.w3.org/TR/REC-CSS2>

## ANATOMY OF A STYLE SHEET

The anatomy of a style sheet includes some terminology that is likely new to you, such as declarations and selectors. Here are some examples of how these terms relate to style sheet programming.

### RULES

At the very core of CSS are rules. To the left is an example of a simple CSS rule and to the right is an example of a CSS declaration.

```
h1 { color: green; }
```

#### Rules consist of two parts

The selector and the declaration. In the example above, the **H1** tag is the selector (the object being modified), and the color is the declaration (what part—color—and how—the selector is being changed).

```
{ color: green; }
```

#### The declaration has two parts

The property and the value. In the example above, property is the color of the **H1** tag that is being modified. The value is set to green, which specifies how the property is being modified.

### note

#### HTML 101: WHAT IS AN ELEMENT?

In case you haven't read the HTML chapter or have forgotten what an "element" is, I thought it would be a good idea to revisit that term here before moving on to the details of Cascading Style Sheets: it's the tag or tagset plus content. `<table border="1">` is the opening tag plus an attribute of the element **TABLE**.

### note

#### CSS & ACCESSIBILITY

This chapter will show how Cascading Style Sheets play a critical role in making your web pages accessible. In Chapter 8, "Accessibility Issues," the role of CSS and accessibility are covered more fully. Make sure you review that section to learn more about this important issue facing web designers.

## Adding CSS TO HTML

Defining your CSS rules is only a part of adding CSS to your pages. Once you have defined your rules, you need to add the CSS rule to your HTML document so it is rendered properly by the browsers. This is done by simply adding a small bit of code within the **<head>** tag of your HTML document. This code will serve as a container for all the CSS within your document.

This is an example of a simple page that contains CSS:

### code

```
<html>
<head>
<title>Untitled</title>
1 <style type="text/css">
2 <!--
3 h1 {color: green;}
4 -->
5 </style>
</head>
<body>
6 <h1>The Art of Bonsai</h1>
</body>
</html>
```

### code deconstruction

- 1 The **STYLE** element is a container that goes in the **HEAD** section of your HTML document. The **TYPE** attribute specifies the type of style sheet being used, which in this case is **text/css**. The **STYLE** element can contain any number of different styles.
- 2 This is the opening tag of an HTML comment. By placing your styles within an **HTML** comment, users with older browsers not capable of viewing CSS will ignore the code instead of seeing the code displayed in error in their browser.
- 3 This style element contains one style, which applies to the **H1** tag, changing its color to green. **H1** is the selector, **color** is the property, and **green** is the value.
- 4 Closing HTML comment tag.
- 5 Closing style tag.
- 6 This line contains some text formatted with the **H1** tag, which has been redefined using the CSS code above.

## **GROUPING SELECTORS**

As you add more and more styles to your pages, you might find yourself making the same stylistic change to multiple HTML elements. In these cases, you might consider grouping your CSS selectors. This can shorten the amount of code and make for a quicker download. As you will see, grouped selectors must be separated by a comma in order to function properly.

For example, if you had these CSS selectors in your document:

### **code**

```
h1 {color: blue; font-family: verdana}
h2 {color: blue; font-family: verdana}
h3 {color: blue; font-family: verdana}
```

they could be optimized and grouped like this:

### **code**

```
h1, h2, h3 {color: blue;
font-family: verdana }
```

### **code deconstruction**

This rule specifies that all text within **h1**, **h2**, and **h3** tags will display in the Verdana font with a blue color.

## **CLASS SELECTORS**

In the preceding examples, you learned how to create styles based on HTML elements using selectors. Selectors are related to HTML elements. If you wanted to apply a style to something that wasn't a tag (let's say there was a certain sentence in your document that you wanted to be bold, but it wasn't anything other than text from a structural definition), this kind of situation is where Class selectors are of value.

Another situation in which a Class selector comes in handy is when you want to style the appearance of HTML tags differently. Earlier you learned how to redefine the appearance of an HTML element (the **H1** tag). Suppose you don't want to format every **H1** tag the same way? This is a perfect place to consider creating a Class selector. Consider the code on the following page:

**This is some text formatted with a class!**  
 This text has not been formatted with a class.

#### code

```
<head>
<style type="text/css">
<!--
1 .text1 {font-family: Verdana;}
-->
</style>
</head>
<body>
2 <h1 class="text1">This is some text
formatted with a class!</h1>
3 <h1>This text has not been formatted
with a class.</h1>
</body>
```

#### code deconstruction

- 1 Class selectors are written in this syntax: always beginning with a dot (.) and a unique name. The rest of the syntax will follow the conventions you learned earlier.
- 2 The Class selector is applied to the HTML element using the class attribute and the name you assigned to the class. In this example I gave the class a name of "text1". Notice that the dot (.) is not included in the class attribute!
- 3 For comparison, this line has not been formatted using the Class selector.

By creating a Class selector, I could now apply this formatting anywhere in my document independent of the HTML element. It will apply this format only where it encounters this class, not across every instance of an HTML element like previously shown.

Here is an example of a Class selector being applied only to a specific portion of a paragraph:

There are **five basic styles** of bonsai.

#### code

```
<head>
<style type="text/css">
<!--
1 .text1 {font-family: verdana;
font-weight: bold;}
-->
</style>
</head>
<body>
2 <p>There are <SPAN CLASS="text1">five
basic styles</SPAN> of bonsai.</p>
</body>
```

#### code deconstruction

- 1 The Class selector syntax is displayed on this line with the **font-family** and the **font-weight** properties applied.
- 2 The **<SPAN>** tag is used to designate the area of text that is to be formatted using the Class selector. Notice that there is a closing **</SPAN>** tag to end the formatting.

## BLOCK-LEVEL & INLINE-LEVEL ELEMENTS

Throughout this chapter and book, you will find many references to block-level and inline content, so it is important to understand what these terms mean.

**Block-level elements** act like boxes that start at the margin of one line of text and end so that the content after the closing element is forced to start on a new line of text. The content of a block-level element can be, and typically is, several lines long. Basically, block-level elements start and end a line of text. For example, the paragraph `<P>` element is a block-level element. It starts at one margin and anything that comes after the closing `</P>` element is forced to appear on a new line. Any formatting applied to a block-level element will affect everything within it. You cannot place one block-level element inside another block-level element. To do that, you would use an inline element, which I explain below. Let's consider the following example and code:

Mary Had a Little Lamb

Mary had a little lamb, its fleece  
was white as snow.

### code

```
<head>
<style type="text/css">
<!--
.body {font-family: Verdana, Arial,
Helvetica, sans-serif; font-size: 10px}
.lamb {color: #FF0000}
-->
</style>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<div class="body">
<p>Mary Had a Little Lamb</p>
<p>Mary had a little lamb, its fleece<br>
was white as snow.</p>
</div>
</body>
```

### code deconstruction

A `<DIV>` element is used to create a range, or invisible box if you will, around the two `<P>` elements. The body **CLASS** is attached to the `<DIV>` element, which causes both paragraphs to be formatted with the body **CLASS**. So, instead of applying the body **CLASS** to both paragraph elements, it was applied once to the `<DIV>` element. This results in less code, which is a good thing.

There are a bunch of block elements within HTML. I have listed some of them in the chart below.

Block Element	Definition
<b>BLOCKQUOTE</b>	Blockquote
<b>BODY</b>	Body
<b>BR</b>	Line break
<b>DD</b>	Definition description
<b>DL</b>	Definition list
<b>DIV</b>	Division
<b>DT</b>	Definition term
<b>H1 - H6</b>	Heading levels
<b>HR</b>	Horizontal rule
<b>HTML</b>	Well ... ;-)
<b>LI</b>	List item
<b>OBJECT</b>	Object
<b>OL</b>	Ordered list
<b>P</b>	Paragraph
<b>PRE</b>	Preformatted
<b>UL</b>	Unordered list

**Inline elements:** As described, inline-elements act like boxes as well, except they are used within block-level elements. This is the difference between the two. For example, if you have a **<DIV>** element that is formatting a paragraph of text with a style sheet and you want to format a single word within that **<DIV>** element, you would use the inline element **<SPAN>**. For example, consider the example and code below:

Mary Had a Little Lamb

Mary had a little lamb, its fleece was white as snow.

#### code

```
<head>
<style type="text/css">
<!--
.body {font-family: Verdana, Arial,
Helvetica, sans-serif; font-size: 10px}
.lamb {color: #FF0000}
-->
</style>
</head>
<body bgcolor="#FFFFFF" text="#000000">
<div class="body">
<p>Mary Had a Little Lamb</p>
<p>Mary had a little
<span class="lamb">lamb</span>,
its fleece<br>
was white as snow.</p>
</div>
</body>
```

#### code deconstruction

As you can see above, I used the **<SPAN>** element and `lamb` class to format the word `lamb`, which is within the **<DIV>** element. This is a good example of how the **<SPAN>** element is used.

There are also a bunch of inline elements within HTML. I have listed some of them in the chart below.

Inline Element	Definition
<b>A</b>	Anchor
<b>EM</b>	Emphasis
<b>I</b>	Italic
<b>IMG</b>	Image
<b>SPAN</b>	Span
<b>STRONG</b>	Strong
<b>TT</b>	Teletype

## PSEUDO-CLASSES

So far, you have learned about selectors, classes, and IDs. Things are about to get more interesting with pseudo-classes. These are class selectors that let the designer apply styles to elements that don't exist within the document. If you are scratching your head, don't worry. I had the same reaction when I was learning these. Thank goodness it wasn't dandruff. ;-)

Pseudo-classes let you apply styles to elements that you know will exist, but you just don't know when. For example, users are likely to move their cursor over a link at some point. The anchor tag is probably the most common pseudo-class. Let's look at a pseudo-class in context in the following code.



Here you can see an active link, being viewed in Netscape Navigator 4.77 for Windows, that has the line removed using a pseudo-class.

### code

```
<head>
<style type="text/css">
1 a:link {text-decoration:none;}
</style>
</head>
<body>
2 <a href="http://www.macromedia.com">
Click here to visit Macromedia.com</a>
</body>
```

### code deconstruction

- 1 This is the proper syntax to create a pseudo-class for the anchor tag. Because the decoration attribute has been set to none, there will not be an underline beneath this text link.
- 2 The pseudo-class is automatically applied to every anchor tag in the document. You do not have to add any additional code in the <BODY> portion of the document.

The following chart identifies the pseudo-classes available in the CSS2 recommendation. Note that pseudo-class selectors and links must be in a specific order to work properly (shown here).

Pseudo-Class	Definition
:link	Adds style to unvisited links.
:visited	Adds style to visited links.
:hover	Adds style to link when the mouse is over it.
:active	Adds style to link when it is clicked on.
:first-child	Adds style to an element that is the first child of another element.*
:lang	Can specify a language for an element.*

\* Using the **:first-child** and **:lang** pseudo-classes is really advanced. If you are interested in learning more about them, you should consider purchasing the following handy books:

#### Eric Meyer On CSS

Eric Meyer  
New Riders Publishing  
0-7357-1245-X

#### Designing CSS Web Pages

Christopher Schmitt  
New Riders Publishing  
0-7357-1263-8

## note

### COOL STUFF WITH PSEUDO-CLASSES

There are some neat things you can do with pseudo-classes, among them is the capability to remove or create text link effects. For example, you can use them to change the color of text when users move their mouse over the text link. You can also use them to remove the underline under text links altogether! Let's take a look at the code for some of these cool tricks.

```
a:link { text-decoration: none}
```

This pseudo-class will remove the underline from all links on the page.

```
a:hover { color: #FF0000; text-decoration: underline}
```

This pseudo-class will cause all text links on the page to turn red and underlined when the users move their mouse over the link.

Have fun and experiment with these pseudo-classes to see what kinds of effects you can come up with.

## tip

### BOOKS ON CSS

#### Cascading Style Sheets: Designing for the Web

Hakon Wium Lie and Bert Bos.  
Addison-Wesley  
ISBN: 0201596253

#### Eric Meyer on CSS

Eric Meyer  
New Riders Publishing  
ISBN: 0-7357-1245-X

#### Designing CSS Web Pages

Christopher Schmitt  
New Riders Publishing  
ISBN: 0-7357-1263-8

## PSEUDO-ELEMENTS

Pseudo-elements are pretty similar to pseudo-classes, except they do not affect the anchor tag of a document. Instead, pseudo-elements can be used to control the first letter or line of text. In the CSS1 recommendation, there are two pseudo-elements: first-letter and first-line. These are used to format the first letter and line of text respectively. It is important to note that pseudo-elements can only be applied to block elements, such as the <P> tag.

A pseudo-element has a general form of:

```
p.dropcap:first-letter { font-size: 200% }
```

The first character in this paragraph has been formatted using the first-letter pseudo-element. As you can see, this is an easy way to add a drop-cap to your paragraphs of text. Cascading Style Sheets can really make things look better!

Here you can see a drop-cap that has been added to this paragraph using the first-letter pseudo-element. That was quick and easy. :-)

### code

```
<head>
<style>
1 p.dropcap:first-letter {font-size: 200%;
float: left}
</style>
</head>
<body>
2 <p class="dropcap"> The first character
in this paragraph has been formatted
using the first-letter pseudo-element.
As you can see, this is an easy way
to add a drop-cap to your paragraphs
of text. Cascading Style Sheets can
really make things look better!</p>
</body>
```

**code deconstruction**

- 1 This is what the code looks like for this pseudo-element. In this example, the font size was increased, floated to the left, applied to the <P> block element, and called "dropcap".
- 2 The pseudo-element is applied to the paragraph tag, one of the many block tags.

Next, let's take a look at the first-line pseudo-element. As you will see, the code for the two pseudo-elements is very similar:

```
p:first-line { font-variant: small-caps }
```

THE FIRST LINE OF THIS PARAGRAPH HAS BEEN formatted using the first-line pseudo-element. As you can see, this is an easy way to add a style to your paragraphs of text. Cascading Style Sheets can really make things look better!

Here you can see that the entire first line of the paragraph has been formatted differently from the rest of the text.

The code for the first-line pseudo-element does not look a whole lot different from the first-letter pseudo-element:

**code**

```
<head>
<style>
p:first-line { font-variant: small-caps }
</style>
</head>
<body>
<p>The first line of this paragraph has
been formatted using the first-line pseudo-
element. As you can see, this is an easy
way to add a style to your paragraphs of
text. Cascading Style Sheets can really
make things look better!</p>
</body>
```

Here is a chart that identifies the pseudo-elements available in the CSS2 recommendation.

Pseudo-Element	Definition
:first-letter	Adds style to the first letter of a text line.
:first-line	Adds style to the first line of text block.
:before	Inserts the content before an element.
:after	Inserts the content after an element.

## HIDING STYLES FROM OLDER BROWSERS

If you enclose your style sheets inside HTML comments, browsers that don't understand the **STYLE** tag will be prevented from seeing it. Those browsers that do understand the style sheet will ignore the comment tag and apply the style sheet to the page.

### code

```
<head>
<title>CSS Examples</title>
<style type="text/css">
<!--
h1 {font-family: Verdana}
-->
</style>
</head>
```

Note: Remember just because the style sheet itself is enclosed in HTML comments in the code above, browsers that don't understand the **STYLE** tag are prevented from displaying the style sheet in the browser window (that's one of the rules of HTML—ignore the tags you don't understand). Browsers that do understand the style sheet will ignore the comment tags and apply the style sheet to the page.

H1 font-family: Verdana

## Working with Ducks in a Row Rubber Stamps

Because the style of art throughout the catalog is consistent, any of the designs may be used together side by side, or superimposed, making the possibilities for personal expression boundless.

You will find designs in our catalog which cover a vast array of categories, from floral images to humor, holiday stamps, border designs and decorative imagery, for all around use. You, as the stamping artist will be the one to create original art works from these designs, and nothing pleases us more than to see the beautiful and original ways in which our designs are being used.

Throughout the catalog you will find some helpful hints on some of the uses and techniques to enjoy your stamps, but we're sure that you as the stamping artist can teach us a thing or two! One thing for certain, the possibilities are endless!

Here is an example in Netscape 3 showing a style sheet that was not hidden within any comment tags.

## TYPES OF STYLE SHEETS

There are three popular types of style sheets: internal (embedded), external, and inline. The following sections provide you with an example and description of each. Knowing the different types and when to use them should make you feel much more comfortable working with style sheets. Here is a brief description of the three types of Cascading Style Sheets:

- > **Internal style sheets** are the most common type of style sheet used. These are constructed using the **<STYLE>** element and are part of the HTML document, which means all the formatting information is contained within the **<HEAD>** portion of the HTML document. These types of style sheets are easy to create and are useful for creating consistent styles within a single page.
- > **External style sheets** are the most powerful of the three. These style sheets exist as separate documents from the HTML page. This allows them to apply to multiple pages using the **<LINK>** element. With one change to the style sheet document, you can update hundreds or even thousands of pages! These types of style sheets are great for bringing design consistency to a large number of pages and can make updating multiple pages more efficient.
- > **Inline style sheets** exist within the **<HEAD>** element of the HTML page and are created using the **<STYLE>** element. These styles are useful for applying a style to a document that contains styles from an internal or external style sheet. The inline style will override the styles defined by the internal or external style sheet.

### INTERNAL

This is a code example of an “internal” style sheet. Notice the **<STYLE>** element is used inside the **<HEAD>** portion of the page. All of the formatting information is contained within the HTML page. These types of style sheets will only affect the appearance of this single page. They can be effective for creating design consistency within a single page. When would you use an internal style sheet? These are useful when you want the style to only apply to a single HTML page.

#### code

```

<html>                                     </style>
<head>                                     </head>
<title>Internal Style Sheet                 <body>
Example</title>                             <h1>Internal Style Sheets are cool!
<style type="text/css">                     Woo-hoo!</h1>
<--                                         </body>
h1 {font-family: Verdana}                   </html>
-->
```

## EXTERNAL

Here's an example of an "external" style sheet. Notice the `<LINK>` element is using the `<HREF>` attribute to point to the "external.css" file. The `external.css` file is the document that contains all the formatting information for this HTML page. Because the formatting information exists independent of the HTML page, it can easily be applied to several HTML pages and updated more efficiently.

### code

```
<html>
<head>
<title>External Style Sheet
Example</title>
<link rel="stylesheet" href="external.css"
type="text/css">
</head>
<body>
<h1>Check it Out!</h1>
<p>My style is cool, daddy-o</p>
</body>
</html>
```

### code deconstruction

The file this document references is called `external.css`. You can give your style sheet any file name you want; just be sure to save it in text-only mode (just like HTML), upload it to the same directory as your HTML (or set the link to whichever directory in which it lives), and give it a `.css` extension. You don't have to put an external style sheet inside a comment tag because no style information is stored within the HTML code.

Your `external.css` document could look like this:

### code

```
h1 {color: FFCC33; font-family: sans-serif}
p {background: black; font-family: verdana}
```

### code deconstruction

The external style sheet document should not contain any HTML whatsoever—just CSS rules.

## INLINE

An "inline" style sheet only applies to the parts of an HTML document that are specified and will override any style settings being applied by an external or internal style sheet.

### code

```
<html>
<head>
<title>Inline Style Sheet Example</title>
<style type="text/css">
<!--
h1 {font-family: Verdana}
-->
</style>
</head>
<body>
<h1 style="font-family: Times New
Roman">Wow - Inline Style Sheets are
really powerful!</h1>
</body>
</html>
```

### code deconstruction

In the example above, even though the "internal" style sheet formats all `H1` tags with Verdana font, the "inline" style will override that setting and format the text using Times New Roman.

When would you use an inline style sheet? Whenever you want a style to affect only a portion of an HTML page and/or you want to override the settings of an external or internal style sheet.

The disadvantage to this method is that you have to add the inline style code every single time you want to use it. The next `H1` text after this one would revert back to the default browser display unless you added yet another `STYLE` attribute or have an external style applied that defines the appearance of the `H1` element.

## MEASUREMENT UNITS FOR TYPE

In order to work with style sheets, it's important to understand the types of measurement units that can be specified. Measurement units in electronic type are tricky, because when you specify a "twelve-point" typeface, it probably won't mean the same thing across computer platforms. Anyone who has ever looked at type on the same web page inside a Windows and Macintosh browser can attest that the same code does not produce the same results across platforms. Type size can even change between the browsers on the same computer!

With that in mind, let's look at some measuring units. Below, you'll see a list with the English name, the CSS name, and a short description for each measurement unit. I have also identified which are absolute and relative units, which is an important thing to know as you choose which unit to use with your designs. Here is a chart that identifies CSS length units.

English	Type	CSS	Description
Pixel	Relative	Px	In CSS, a pixel is the distance from one grid unit on a computer screen.
Point	Absolute	Pt	A point is 1/72 inch. How many pixels that is depends on the resolution of your screen, but it is 1 pixel on 72 dpi systems.
Pica	Absolute	Pc	6 picas equal 1 inch.
Em	Relative	Em	The size of the text is relative to the size of the parent element. A 2em type size applied to a body of text with a size of 12pt would be 24pt, or twice as large as the parent element.
Ex	Relative	Ex	1/2 em. The height of a lowercase "x."
Inch	Absolute	in	Same as the good, old ruler on your desk. ;-) 72 points equal 1 inch.
Centimeter	Absolute	cm	2.54 cm equal 1 inch.
Millimeter	Absolute	mm	25.4 mm equal 1 inch.

Most web-based graphics are a fixed number of pixels wide and a fixed number of pixels high, making their sizes consistent between operating systems. Text is not consistent between operating systems; therefore, I recommend that you use "pixels" as the measurements for your text. That way, the text will have a relative size to other text on the page.

## WORKING WITH TYPOGRAPHY

CSS defines a number of different properties for working with text. The **font-family** property, for example, lets you specify a particular font family, either by name (like Verdana) or by generic type (like sans-serif).

**Property:** font-family

**Values:** Name of the font family, for example, Verdana or Georgia, or one of the following generic names:

- serif
- sans-serif
- cursive
- fantasy
- monospace

You can also give a list of values for font-family:

```
H1 {font-family: Verdana, Helvetica, sans-serif}
```

In that case, the system would try to use Verdana, or if that's not available, Helvetica. Otherwise, it will use whatever sans-serif font "is" available.

Another useful text-related property is **line-height**. This property affects the space between lines, also referred to as "leading" in the print world, after the strips of lead used to provide the spaces between lines of metal type.

**Property:** line-height

**Values:** number

default

Sets the amount of space between lines of text

The **line-height** is measured from the baseline of one line to the baseline of the next. The amount of space above and below the text is the same. For example, a **line-height** of 12pt would add 6pt of space above the text and 6pt of space below the text.

```
P {  
  font-family: Georgia;  
  font-size: 12px;  
  line-height: 1.5em;  
}
```

This will set the **line-height** to **1.5 em**. (1em is the same as the font-size, so 1.5 em is one-and-a-half times the body size of the type, which would be 18px in this example). This would add 3px to space above and below (a total of 6px, or the difference between the **font-size** and the **font-height**) the text.

## ABSOLUTE POSITIONING

Designers have long complained that it's not possible to absolutely position a graphic on a page using HTML. CSS offers a solution for this complaint, allowing designers the first real ability to position objects accurately inside the web browser window.

Absolute positioning can be achieved using either a Class selector or ID selector. Most web designers prefer to use an ID selector because an ID selector can only be used once. It is, however, possible and technically correct to use a Class selector—either will work just fine. If you were asking me, I would recommend that you use an ID selector in most cases.

In the example below, I created an ID selector to position the selected element in the uppermost left corner of the browser window.

### code

```
1 #absolute {
2 position: absolute;
3 left: 0px;
4 top: 0px;
5 z-index: 1;
}
```

### code deconstruction

- 1 The ID selector always begins with a hash mark (#) and must be unique within the document. That means that you can only define it once, and you can only use it once. It works almost like the class selector (.), but is usually used for absolute positioning of objects on a page, where only one object will have any single position.

- 2 The **position** property can be either **absolute** or **relative**. Use **absolute** for objects that you want to have at a specific position on the page and **relative** for objects that you want to position relative to wherever they would have otherwise fallen. Absolute positioning was used in this example to position the graphic at a specific point on the page. An example of relative will be given a little later.

- 3 The **left** and **top** properties are used to position the object. These measurements are absolute measurements from the top-left corner of the screen, leaving no padding between the graphic and the edge of the screen. This style will place the graphic in the absolute upper-left corner of the screen.

- 5 Objects that are put on the screen with absolute positioning can be laid on top of each other. The **z-index** property tells the browser which objects should be on top of (or under) which other objects. Higher-numbered objects will overlay lower-numbered objects.



Here's a page that uses the absolute positioning I just defined in the code to your left.

**code**

```

<html>
<head>
<title>Buddah Gallery</title>
<style type="text/css">
<!--
#absolute { position: absolute; left:
0px; top: 0px; }
-->
</style>
</head>
<body bgcolor="#FFFFFF" text="#000000"
background="buddha3.jpg">
I 
</body>
</html>

```

**code deconstruction**

- I Notice the **ID** attribute at the end of the **IMG SRC** line. This tells the browser to use the "**absolute**" ID selector to format this element. In this example, I have set the position of that selector to 0, 0, which will place the selected element in the upper-left corner of the browser window.

**note****ID VERSUS CLASS**

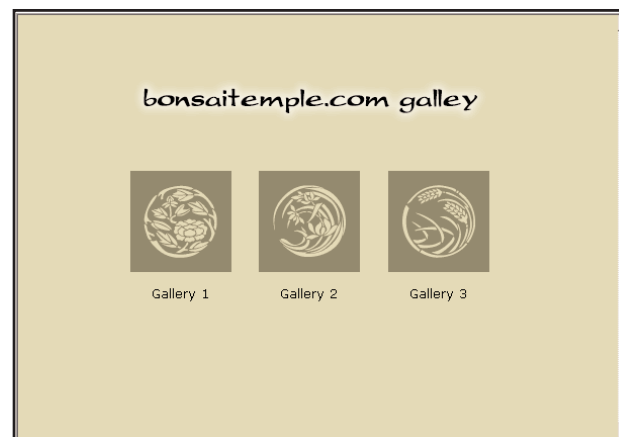
At this point, you might be wondering about the difference between a Class and ID selector. Functionally, it doesn't really matter because they both accomplish the same thing. However, there are a few nuances that make them different.

For starters, IDs start with a # symbol instead of a dot (.). Also, it is considered an error to use the same ID selector more than once. In fact, IDs should be used only once within a given document, and they should always have a unique name different from other IDs in the document. This makes them great for absolute positioning, where they would only want to use them once. Validation engines—and some future browsers—may flag errors on duplicate ID selectors. Classes are great for formatting text because they can be used multiple times on a web page. Because of this, you will find yourself probably using Classes most of the time.

**DIV & ABSOLUTE POSITIONING**

In the previous section, I set the absolute position of an image by formatting it using an ID selector. There are other ways to format the elements in your web pages. The **<DIV>** tag is an important part of positioning with cascading style sheets. The **<DIV>** tag acts like a container to hold other elements, much like a box would hold a pair of shoes.

The **<DIV>** element can be used to set up entire page layouts that would normally require HTML tables to achieve. In fact, the CSS **<DIV>** tag is what is intended to replace the HTML **<TABLE>** tag. It can take some time getting used to the **<DIV>** tag and how to manipulate it to get the page layouts you want. However, by using the **<DIV>** tag, you are making a significant separation between the structure and the formatting of your page, which is the goal of style sheets in the first place!



Here is an example of a simple page layout that you would normally create using tables, but instead was created using the **<DIV>** tag and classes.

In this example, you can't tell whether the layout was created using tables or absolute positioning with CSS. However, the code on the following page reveals this secret.

code

```

<html>
<head>
<title>Buddah Gallery</title>
<style type="text/css">
<!--
1 .text {font-family: Verdana, Arial,
Helvetica, sans-serif; font-size: 10px}
-->
</style>
</head>
<body bgcolor="#CCCC99" text="#000000">
2 <div id="gallery1"
style="position:absolute; left:111px;
top:154px; width:100px; height:100px;
z-index:1">
</div>
3 <div id="gallery2"
style="position:absolute; left:238px;
top:154px; width:100px; height:100px;
z-index:2">
</div>
4 <div id="gallery3"
style="position:absolute; left:366px;
top:154px; width:100px; height:100;
z-index:3"><IMG SRC="icon3.gif"
width="100" height="100">
</div>
5 <div id="pagetitle"
style="position:absolute; left:109px;
top:56px; width:360; height:50;
z-index:4"><IMG SRC="title.gif"
width="360" height="50">
</div>
6 <div id="text1" style="position:absolute;
left:111px; top:267px; width:100px;
height:20px; z-index:5" class="text">
<p align="center">Gallery 1</p>
</div>
7 <div id="text2" style="position:absolute;
left:238px; top:267px; width:100;
height:20; z-index:6" class="text">
<p align="center">Gallery 2</p>
</div>
8 <div id="text3" style="position:absolute;
left:366px; top:267px; width:100;
height:20; z-index:7" class="text">
<p align="center">Gallery 3</p>
</div>
</body>
</html>

```

code deconstruction

- 1 This is an internal class selector used to format the HTML text in this document. I used pixels for my text size to make sure it's viewed properly between the Mac and Windows operating systems.
- 2 This <DIV> tag is holding the first of the three gallery icons on the page. Notice that the position has been set to *absolute*, and the *top* and *left* values are defined to set its exact location within the browser window.
- 3 This <DIV> tag is holding the second gallery image and is defined the same as the first <DIV> tag.
- 4 This <DIV> tag is holding the third gallery image and is formatted like the others.
- 5 This <DIV> tag is holding the page title image. Notice that this code appears beneath the three images, but the actual image appears above the gallery images on the page. This occurs because the page title's location is defined in the absolute positioning properties, not by its position in the HTML code!
- 6 This <DIV> tag is holding the first block of text. Notice that this block of text has been formatted using the *text* class selector.
- 7 This <DIV> tag is holding the second block of text.
- 8 This <DIV> tag is holding the third block of text.

While this code might seem a bit complex and hard to navigate, it is the way of the future and knowing how to work with it will be essential to your success as a web designer. Don't panic—most of the WYSIWYG HTML editors will let you work in a visual environment and automatically create the code behind the scenes. Phew!

## WYSIWYG EDITORS

You have the choice to author CSS from scratch or use an HTML editor such as Dreamweaver or GoLive. These editors make using CSS much easier, and the code they write is updated and current to support the most current CSS recommendations. If you do decide to use an HTML editor to write your CSS, make sure you are using a current version of Dreamweaver or GoLive. The current versions are most certain to write the best CSS code.

Whether you choose to use Dreamweaver or GoLive, you still have to know what you are doing, however, because there is no way for the HTML editor to decide for you whether to choose between an ID or a CLASS, pixels or inches. CSS is another challenge in the web publishing curve. Just when you thought HTML was finally manageable!

This chapter should familiarize you with CSS so that you can make these decisions within HTML browsers. Here are some other helpful references for CSS:

### A List Apart's CSS Articles

<http://www.alistapart.com/stories/indexCSS.html>

### WebReview's Style Sheet Reference Guide

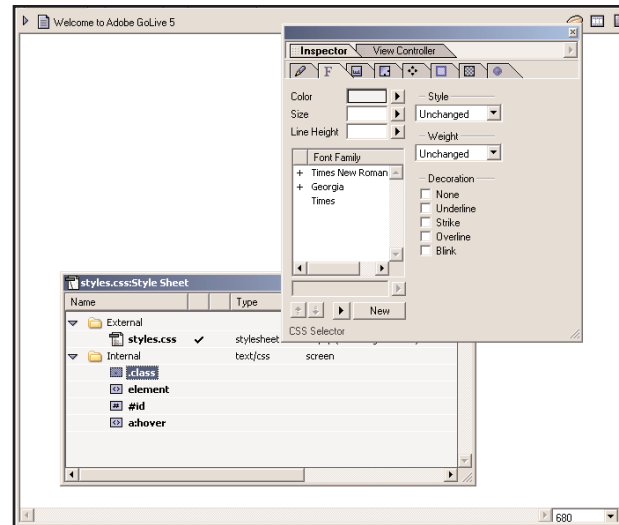
<http://www.webreview.com/style/index.shtml>

### TopStyle WYSIWYG CSS Editor

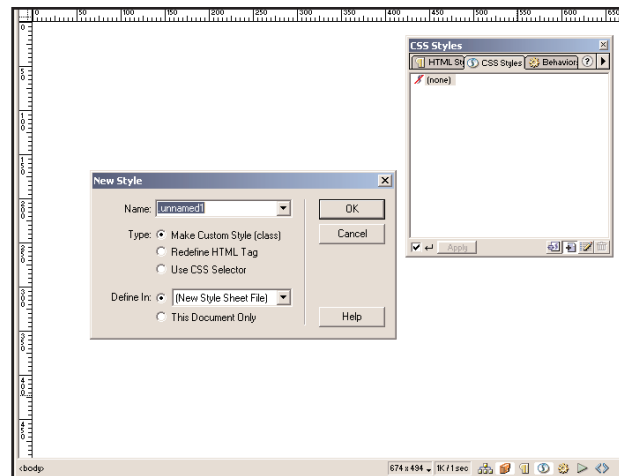
<http://www.bradsoft.com/topstyle/index.asp>

### StyleMaster WYSIWYG CSS Editor

[http://www.westciv.com/style\\_master/index.html](http://www.westciv.com/style_master/index.html)



GoLive supports style sheets, but you definitely have to have a base knowledge of how they work to use it.



Macromedia Dreamweaver supports style sheets, too, but you must know what you are doing to use the WYSIWYG capabilities.

# cascading STYLE SHEETS

## SUMMARY

CSS style sheets are a powerful mechanism for enhancing the presentation quality of web text. Though not appropriate for all audiences due to browser compatibility (especially Netscape 4.x), all recent popular web browsers have excellent CSS support. Depending on your audience, CSS is now—and most certainly will become—an extremely useful method of designing web pages.

- > Using CSS, you will be able to format your text in manners much closer to that of the printed page, and expect your layouts to work consistently—within certain limits—across platforms, browsers, and differing sizes and resolutions of displays. This chapter covered the basic principles of style sheets, so that you can use them now or wait until they are supported more widely.
- > Internal style sheets are used within a single HTML document.
- > External style sheets are used by multiple HTML documents.
- > Inline style sheets only apply to part of an HTML document.
- > It's best to use pixel measurements for specifying type, since they will produce the most reliable results.
- > Be sure to put your style sheet rules within comment tags, so browsers that don't support CSS will ignore the code.
- > While WYSIWYG editors support style sheets, you still need to understand what to specify within the various choices.