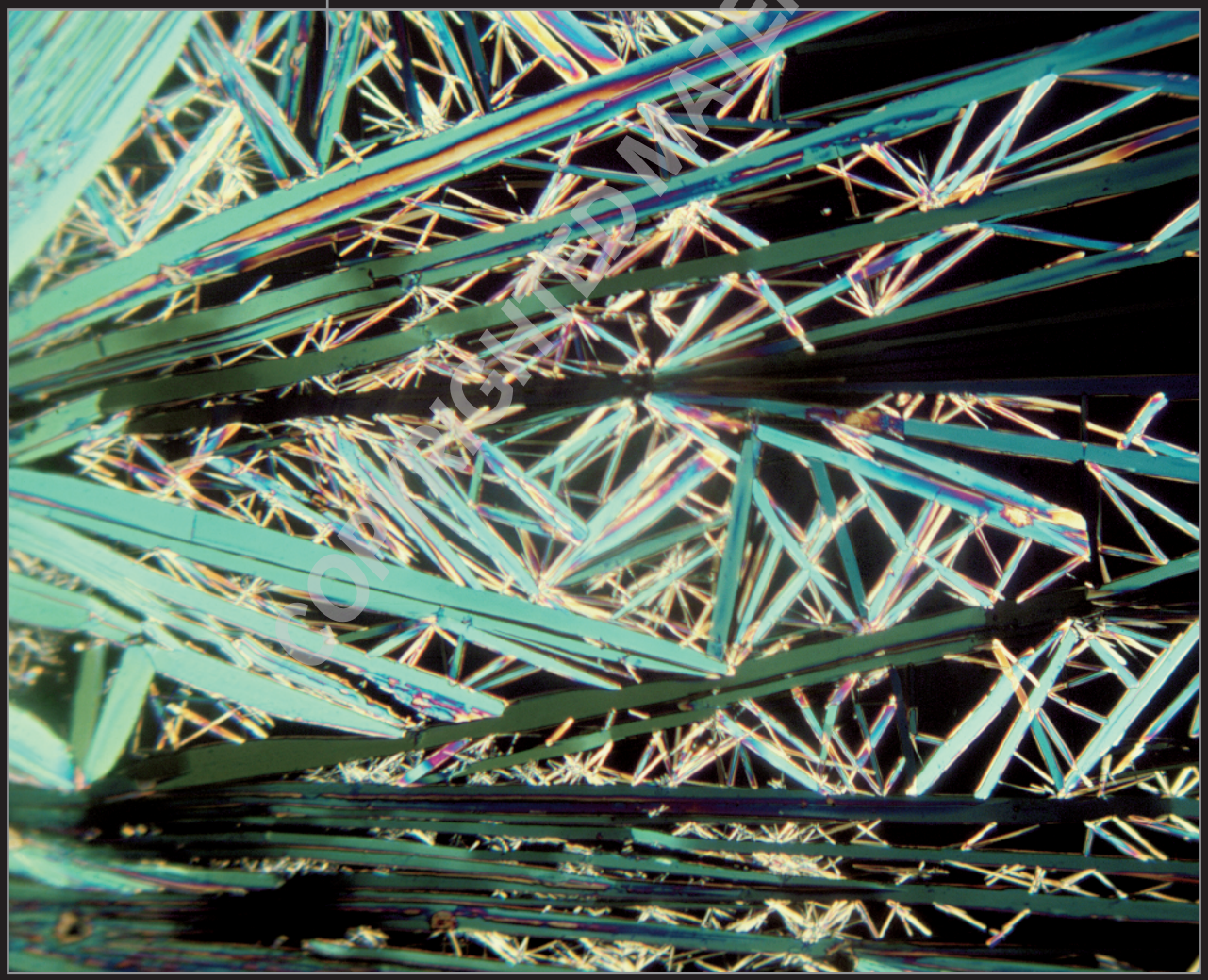
Chapter 1

MAKING CSS WORK FOR YOU



Cascading Style Sheets (CSS) is an open and freely available standard developed to separate presentation and content on a Web site. The standard is developed and published by the World Wide Web Consortium (W3C), a group made up of organizations, industries, manufacturers, and others involved in the development of the Internet.

CSS came on the scene in 1996 with the release of CSS Level 1. CSS Level 2 was released in 1998 and builds on CSS Level 1, supporting more styles and expanding to include different types of media output. CSS 3 is currently in development, and is destined for release in a number of modules addressing different categories of styles, such as selectors, color, or print.

The point of separating presentation and content is that updating and changing the appearance needs to be done only once to a style sheet, and the changes are automatically passed to the browser when the page is displayed. If you can remember the "old days" of Web page building before the time of CSS, you'll appreciate how valuable it is to have a centralized function to update the styles used on a page. Read how guest contributor Margaret Werdermann used a consistent method for producing customized style sheets for her clients.

CSS is paired with the XHTML standard. XHTML reformulates HTML with XML syntax and without any presentation attributes. CSS is supported to different degrees by different browsers, and different browser versions.

Describing Web Page Content

When you write CSS styles you use numerous properties for defining the placement of content on the page. CSS uses the *box model* as the basis for describing everything displayed on a Web page.

You don't have to understand the intricacies of the box model to write CSS, but appreciating how it works will improve your ability to predict how a style looks, and save you tweaking and adjusting time.

NOTE

For full details on the box model, check out the World Wide Web Consortium's CSS2 or CSS3 specifications. Both of these documents make terrific bedtime reading.

There is always one box on a page; each element added to the page adds another box. As you can see in figure 1-1 where I have drawn an overlay of the boxes used in this simple Web page, boxes can be nested within other boxes within other boxes . . . .

Delete and Insert or Replace?

What's the difference? It depends on what else is on the page.

Suppose you are using a page with a large number of comments, links, or form fields on it, like the sample form page shown in Example 1.

Example 1: The active form fields are highlighted in pale purple.

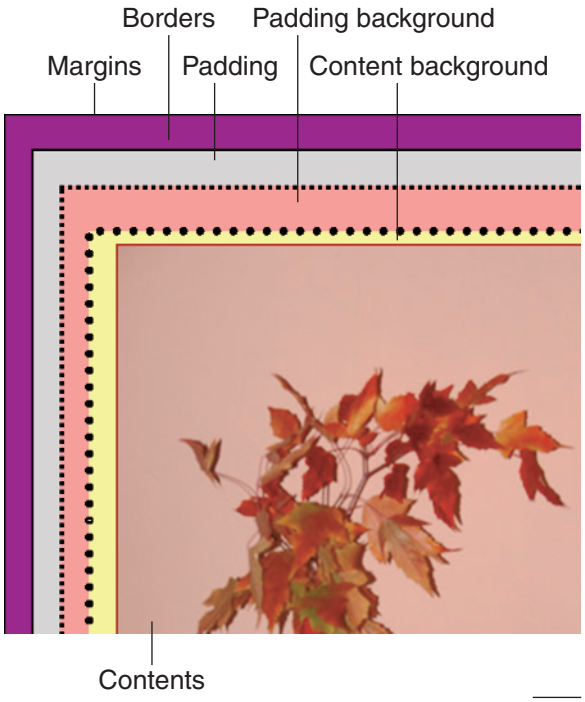
1-1

Identifying the box components

Each box has a content area holding text, images, or other material. The content optionally has padding, border, and margin areas that you specify by assigning style properties.

The characteristics of a box are shown in figure 1-2, and include:

- **Margins.** A margin is the outermost element of a box.
- **Borders.** The border is inside and adjacent to the margin.
- **Padding.** Padding is the space inside the border and surrounding the content.

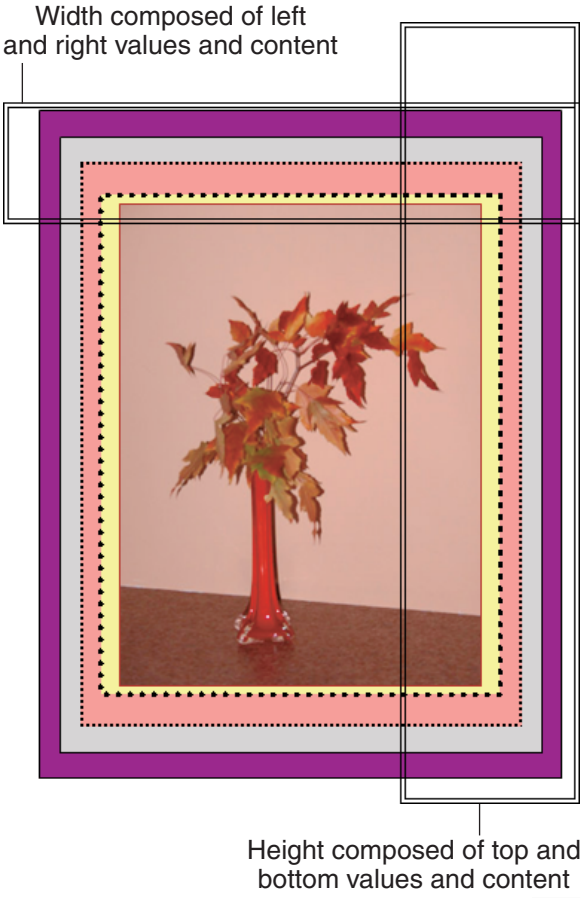


1-2

- **Padding background.** The color or image assigned to the padding space.
- **Content background.** The color or image assigned to the background surrounding the content.
- **Content.** The actual content viewed on the Web page is the innermost element in a box.

Defining the box's size

Planning layouts means you have to know what makes up the overall height and width of a box. In figure 1-3, you can see that:



1-3

- The overall width of the box is made up of the left margin, left border, left padding, content width, right padding, right border, and right margin.
- The overall height of the box is the sum of the top margin, top border, top padding, content height, bottom padding, bottom border, and bottom margin.

Box Model Issues

The current CSS standards identify the `width` and `height` properties of a box as the content area of the box only. The total box width is made up of the content area, as well as padding, borders, and margins. Unless the `width` property is specified, the total box width is the same as the `content area` of the surrounding container element.

Unfortunately, all CSS-enabled versions of Internet Explorer before IE6 function under a different box model where the padding and borders are included as part of a width or height total. Only if the elements don't use borders or padding do the two models jive.

Page layout problems and the headaches accompanying them arise when a box uses an assigned `width` and either (or both) `borders` or `padding` is defined. The standard box model causes the overall box width to increase, and in the IE model the `content area` is decreased by the same amount.

For example:

```
{width:400px; padding:20px;
border:5px;}
```

If you look at this object in a standards-compliant browser, the box from border edge to border edge is 450px, the sum of 400 + (20*2) + (5*2).

In older versions of IE, all the values are combined within the width value for a total of 400px, and the actual width of the content area shrinks to 350px, the remainder of 400px – (20*2 + 5*2).

Locating CSS Information

Style Sheets are templates, very similar to templates in desktop publishing applications that contain rules defined for page elements, or *selectors*.

Placing code inline

Using a style inline is comparable to how HTML was written in the past, where presentation details were included along with the page's content in the body of the page.

The style is written as an attribute for the tag. For example, if I want to use a `<p>` style in a page, like the brown text shown in figure 1-4, the tag is written as:

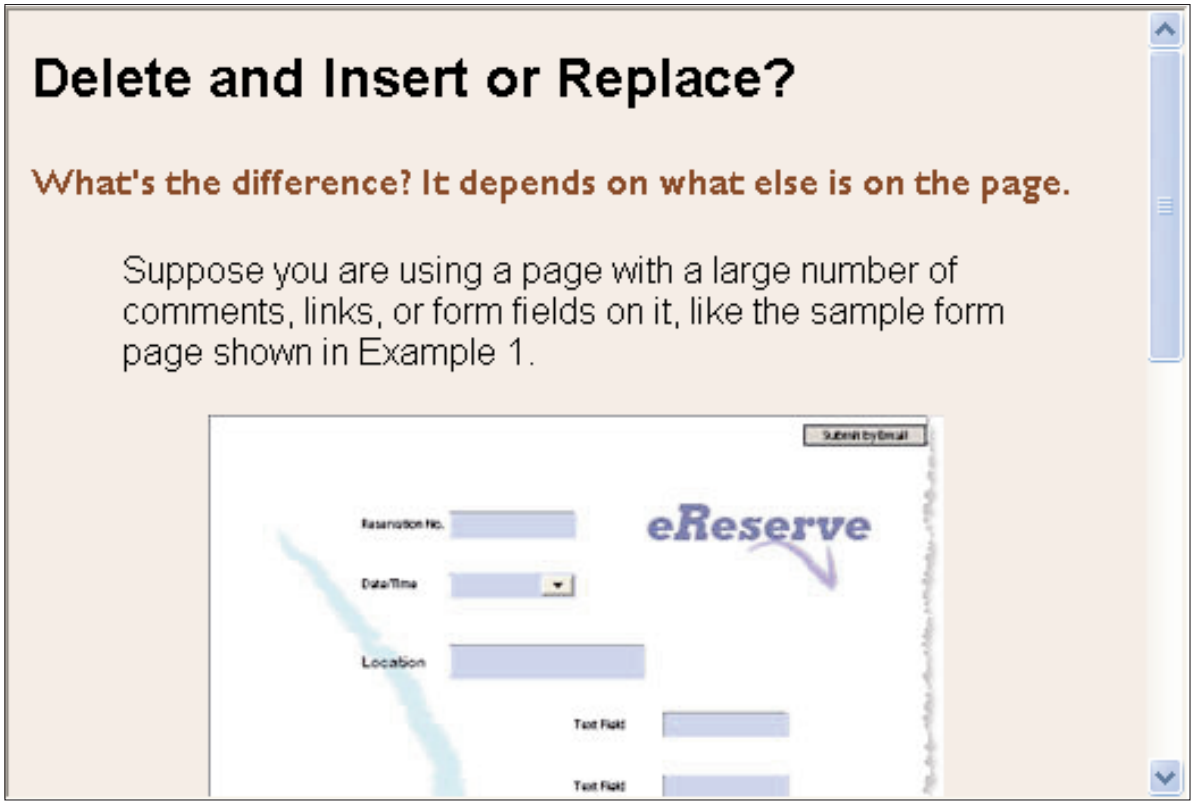
```
<p style= "font-family: 'Gill Sans
 Extra Bold', 'Arial Black',
 sans-serif; color:#663300;"> What's
 the difference? It depends on what
 else is on the page. </p>
```

On the upside, inline CSS lets you:

- Add style information quickly to a specific tag. If the paragraph scripted above were the only instance of content that used that particular `color` and `font-family` properties, then it's easy to add it to the `<p>` tag.
- Decrease clutter in your site's code. A style applied to only one element isn't included in other style listings, which makes organizing and locating your other styles simpler.

On the downside:

- Using inline CSS is a real pain to maintain because you have to scan a page's code for the style's information.
- Often you work with multiple paragraphs on the page that use the same style. Using inline CSS means repeating the style information for each paragraph, as shown in figure 1-5, which hardly contributes to streamlined code.



1-4

```
8  <body style="background-color:#Eee8dd;">
9  <h2> Delete and Insert or Replace?</h2>
10 <p style= "font-family: 'Gill Sans Extra Bold', 'Arial Black', sans serif;
11     color:#663300;"> What's the difference?
12     It depends on what else is on the page.</p>
13 <p style= "font-family: 'Gill Sans Extra Bold', 'Arial Black', sans serif;
14     color:#663300;"> Suppose you are using a page with a large number
15     of comments, links, or form fields on it. In the form page shown
16     in Example 1 the active form fields are highlighted in pale purple. </p>
17 <p style= "font-family: 'Gill Sans Extra Bold', 'Arial Black', sans serif;
18     color:#663300;"> If you merely inserted an edited version of the page
19     and deleted the one you want to remove, you'd lose all your comments
20     and links, as you can see in Example 2. </p>
```

1-5

> You can't reuse common elements. For example, if you have a heading that uses the same color and font family as a type of paragraph, you have to define that heading separately, rather than naming a single style and applying it to different elements.

PRO TIP

In general, I find inline CSS difficult to maintain and hard to troubleshoot, but it does have its place. I have used an inline tag when I am creating a "one-off" page where the content is specific to the page as opposed to a site, and where the style is specific to a single instance of a tag.

Embedding the styles

Rather than adding styles to each individual element on a page, you can include them as a `style` element within the page's `<head>` tags.

The `<p>` example described in the previous section can be moved to a page's `<head>` section, as shown in figure 1-6.

On the upside:

- Embedding the styles on a page is the best way to ensure your styles' information is always included with your page's code, like the example shown in figure 1-6.
- Including the styles in a single block is simple to maintain and troubleshoot on a page-by-page basis.

```
<head>
<title>Tips for Managing PDF Pages</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
<style type="text/css">
.intro {
    font-family: "Gill Sans Extra Bold", "Arial Black", sans-serif;
    color: #663300;
}
</style>
</head>
```

1-6

On the downside:

- Styles can't be shared across pages on a site, meaning you have to copy and paste the code from one page to another.
- When styles need updating, you have to update the code on each page containing a copy of the styles.

PRO TIP

Hide the styles from non-CSS browsers by commenting out the `<style>` tag on the HTML page.

Linking to an external file

Rather than including any of the style information in the page, write a style sheet as a separate file, linked to your Web pages by tags included in the `<head>` tag.

A style sheet is linked to a Web page using this code:

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

The link requires three attributes including:

1. `type`. The `type` attribute defines the `MIME_type` of the target URL. In the case of an attached style sheet, the type is `text/css`; other types can include `text/javascript` or `image/jpg`.
2. `rel`. The `rel` attribute defines the relationship between the current document and the targeted document, such as a style sheet.
3. `href`. The `href` attribute uses the URL of the CSS file as its value.

On the upside:

- Maintaining all the styles in a single file is a convenient way to keep track of a page or site's presentation.
- Incorporating your styles in a single file shows you what you have named different styles, preventing duplication in naming and possible errors.
- Collecting all the styles in a single file lets you see the relationships among the styles, making it simpler for you to identify and take advantage of the existing styles. For example, figure 1-7 shows the same page as that displayed in figure 1-4. In the external style sheet, the brown text is defined as a style, which is then applied to the main heading on the page. The caption under the image, which uses the default `<h5>` tag, also has the style applied.

Delete and Insert or Replace?

What's the difference? It depends on what else is on the page.

Suppose you are using a page with a large number of comments, links, or form fields on it, like the sample form page shown in Example 1.

Example 1: The active form fields are highlighted in pale purple.

1-7

NOTE

For interest and a more cohesive page I also assigned a style to the default `<blockquote>` tag to add padding and a border at the left and bottom edges of the block.

On the downside:

- A style sheet must be accessible by the pages to which it is linked. If a viewer takes a page offline for example, they won't have the linked styles — so much for your carefully crafted page presentation.
- Having all the styles for a site in a single file can be difficult to organize logically.
- Collecting all of a site's styles in a single file can make it difficult to pinpoint a style you want to modify.

Applying an imported style sheet

You can import as many style sheets files as you want and override imported styles using embedded styles, but I don't recommend it except in an environment using a structured style sheet scheme.

When you are trying to find the source of an error and have to pass through numerous hierarchical layers, you'll see why. It is probably well worth the time it takes to rebuild the sheet for the page or site.

You can add more than one style sheet to a Web page by linking a style sheet using an *at-rule*. The rule begins with the @ symbol (hence the name) followed by an alphanumeric keyword, which can also include dashes or underlines.

If you want to add an additional style sheet to an existing one, write:

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

Instead of using a linked style sheet on a Web page, use the at-rule. On your Web page, the `<style>` tag is written similar to:

```
<style type="text/css">
@import url(extrastyles.css);
</style>
```

NOTE

At-rules have other uses as well, such as defining media — for example, `@print` for a printer. Other forms include `@font-face` to define and embed an external font and `@page` to apply styles to printed pages.

Maintaining page styles offline

You can use both linked and embedded styles in a page. You may want to provide both options in cases where a file is viewed offline, such as a lesson or tutorial article.

To include an embedded style sheet within the Web page that lists the basic elements of the page design, follow these steps:

1. Attach and test the page using your site's style sheet.
2. Embed a subset of the styles using a `<style>` element within the `<head>` tag.
3. Delete or comment the code that defines the style sheet. In the example shown in figure 1-8, the code is hidden with comments.

```
3  <head>
4  <title>Tips for Managing PDF Pages</title>
5  <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1" />
6  <!-- <link href="ch01.css" rel="stylesheet" type="text/css" /> -->
7  <style type="text/css">
8  .intro {
9      font-family: "Gill Sans Extra Bold", "Arial Black", sans-serif;
10     color: #663300;
11 }
12 </style>
```

1-8

4. Test the page to make sure the applicable styles are embedded.
5. Reinsert the code defining the style sheet, or remove the comments.

Writing CSS Syntax

CSS syntax, as anyone reading this book already knows, is made up of a selector, a property, and a value, and is expressed as:

```
selector {property: value}
```

The *selector* is usually the HTML element or tag you are defining, the *property* is the attribute you are modifying, and the *value* is the description you are applying to the defined property. Together the selector and the property/value are known as a *ruleset*.

Keep yourself straight! Naming styles functionally is usually easier to understand than naming them according to appearance. Naming a style `schedulebottomrow` is simpler to figure out than `doublelinedarkblue`, especially if your site's color scheme is dark blue and you use several table and form structures.

NOTE

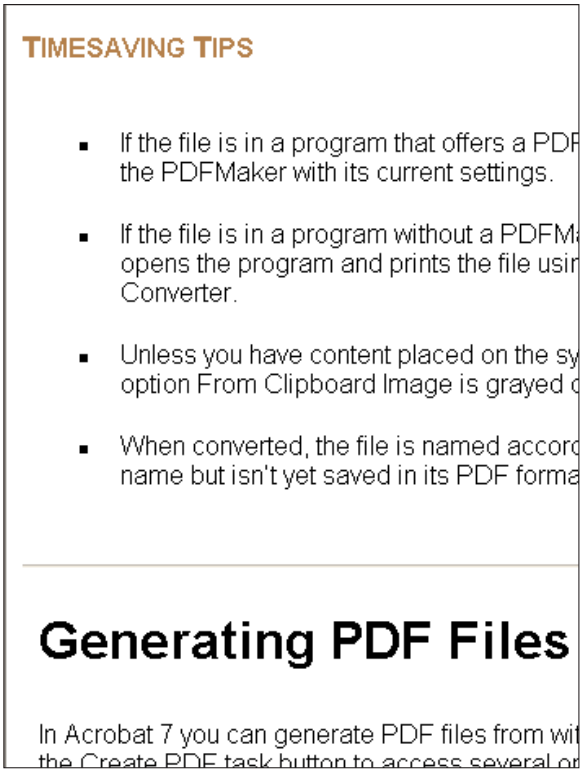
Don't start a class or ID name with a number because Mozilla/Firefox browsers won't recognize the style.

Styling Elements

Instances of the same element, such as `<h4>` tags, can be assigned a unique style. In this case, use the name of the element as the name of the style on the style sheet.

For example, my style for the `<h3>` tag shown on the Web page in figure 1-9 uses a different font-variant and color than the other heading styles, and is written as:

```
h3 {
color:#996633;
font-variant: small-caps;
}
```



1-9

Writing and Using Multiple Styles for an Element

More than one style can be associated with an element. For example, in figure 1-10 you can see two paragraph styles, one introducing the Samples section of the page, and the other describing the first sample. On the style sheet associated with the page the styles for the paragraphs are named `p.basic` and `p.glossaryrow`.

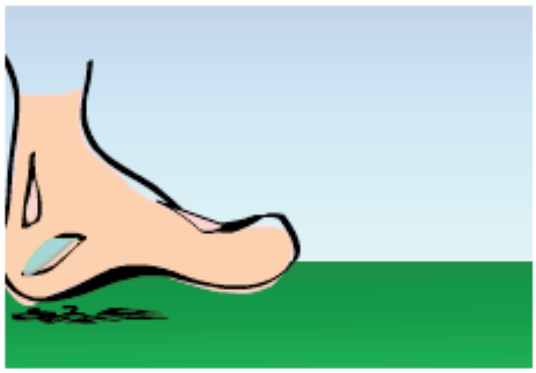
Using the element followed by a period and the style name identifies the style as specific to the tag.

Applying a Rule to Specific Instances

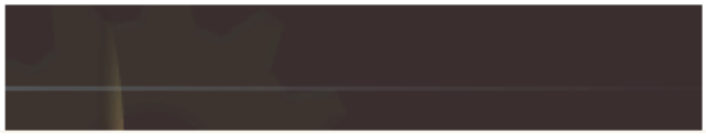
To zero in on page content and apply styles with razor-like precision, take a look at *contextual* selectors. A contextual selector is a string of individual selectors that produces a search pattern.

Samples

Here are sample SWF movies from the book. If you can't see the sample movies, and are using Internet Explorer, check your security settings in the Tools > Internet Options > Security settings dialog box:



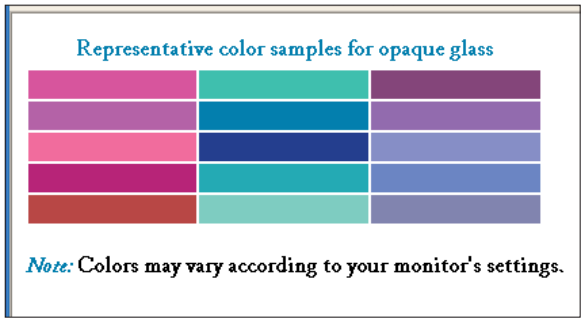
Learn to customize a Move effect's motion path by overriding keyframe angle settings, creating a more realistic sense of motion in Chapter 12.



1-10

Only the last element in the pattern is modified, and only when it meets the listed criteria. You are free to define contextual selectors in a number of ways. The price of this freedom is that contextual selectors can be difficult to write and apply, and errors that occur can be very hard to locate.

Here are two variations on the contextual selector theme. The appearances on a Web page are shown in figure 1-11.



1-11

The first example is written as:

```
div table caption {color:#006699}
```

The element <caption> is shown on the page as a medium blue, and styles the table's caption at the top of the page. However, the style only applies in the context of the element <table> which occurs in the context of the element .

Specify two or more contextual situations separated by commas, written as:

```
div table caption, h4 em
 {color:#006699}
```

The element <caption> is shown in medium blue, and the element is also medium blue when it occurs in the context of <h4>. The line below the table uses the <h4> style; the tag applies only to the word "Note"; the rest of the heading uses the page's basic style.

Writing ID selectors

With the id selector you can define the same style for different HTML elements using a single rule.

For example, a rule can be applied as an id attribute to any element that I choose, such as the figure caption, part of the numbered list, and the horizontal rule shown in figure 1-12.

The style shown in the figure is written as:

```
#accent {
color: #3399CC;
background-color: #E9E9E9;
}
```

Before using an id selector, determine whether you want it to apply to one element or use it more generally. Either write a generic attribute that is applied like the example, or specify the elements to which it can apply. For example, the selector p#accent can be applied only to <p> tags having the rule id="accent".

NOTE

In addition to regular classes and elements, you can also work with and write styles for pseudo-classes and pseudo-elements. Read about pseudo-classes in Chapter 10, and pseudo-elements in Chapter 2.

Figure 1-1. Choose an option from the task button's menu.

Click the Create PDF task button to display the menu and follow these steps:

1. Click the first option, From File, to display the Open dialog box.
2. Locate the file you want to convert to PDF.
3. Click Open. You'll see a progress bar window as Acrobat executes a macro that opens the file in the native program and converts it to a PDF.
4. The converted file opens in Acrobat. Choose File > Save to save the file as a PDF.

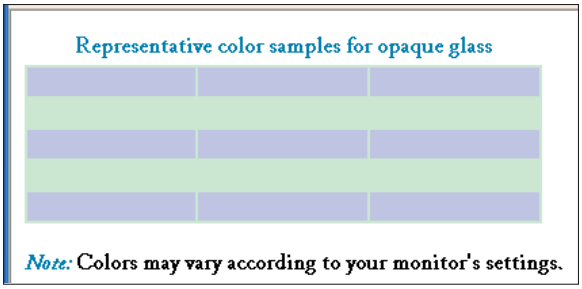
1-12

Claiming an Inheritance

Web pages are structured hierarchically. The `<html>` element is the top of the heap, also known as the *ancestor*. The rest of the page descends from this top-level element.

Like any family, child elements can inherit properties from their parents. In the case of a Web page, the inherited properties result in the default style for the element. For example, a `<table>` element's style is inherited by `<tr>` and `<td>` tags.

If you don't want inherited properties to be assumed by a child element, specify a style for the child element. In the example shown in figure 1-13, alternate `<tr>` tags use unique styles.



1-13

To capitalize on inheritance, you can use a *parent-child* selector to apply properties in defined parent-child relationships. Write the selector by listing two or more selectors separated by a tilde (~).

An example is shown in figure 1-14. In the sample page, the `<em>` element is used within both a `<p>` and an `<h2>` tag.

The style is written as:

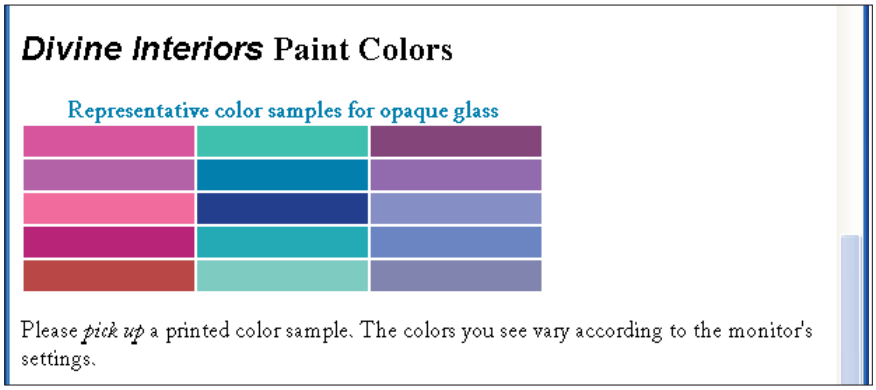
```
body ~h2 ~em {font-family:Arial,
 Helvetica, sans-serif;}
```

As a result, only the element within the `<h2>` tag is styled.

PRO TIP

A property may be assigned an `inherit` value, which means that it uses the same value as that displayed by its parent.

You can use the `inherit` value for properties that aren't normally inherited, such as backgrounds.



1-14

Cascading Styles

You may have experience using CSS and building Web pages without understanding how the cascade rules are applied.

PRO TIP

Like playing cards, you will win some of the time regardless of whether you are familiar with the rules or not, but your odds of winning are sure to improve when you learn the intricacies of the game.

I can't promise any big-money prizes or flashy jewelry, but I can promise you will find that the way you approach designing and using relationships in a style sheet is streamlined, as well as easier to maintain and troubleshoot when you appreciate how the CSS cascade process works.

The CSS cascade is a sorting system made up of rules that organize declarations so there are no conflicts in presentation. When several rules apply, the one with the greatest importance, or *weight,* takes precedence, allowing the browser to solve conflicting rules and display the content correctly.

Style sheets come from three sources, including:

- The author writing the style sheet, who defines the styles either in the document or linked from an external file.
- Users, who can specify style information as well, such as applying a contrasting color scheme-based style sheet.
- User agents, such as Web browsers, who apply a default style sheet before any other style sheets.

Weight assignments

Each *style rule,* or property and value combination, you assign to a style is assigned a weight. More than one rule can apply to the same style and are applied in order of weight.

Sorting strategies

Aside from sorting styles by origin, there are other types of sorts upon which the final displayed style is based. In ascending order, the other types of sorting include:

- **Specified selector.** A selector such as `img.left` overrides a general selector such as `img`.
- **Specified order.** If two rules have the same origin, weight, and specificity, the latter specified rule is used.
- **Defined importance.** Declarations with increased weight take precedence over declarations with normal weight.

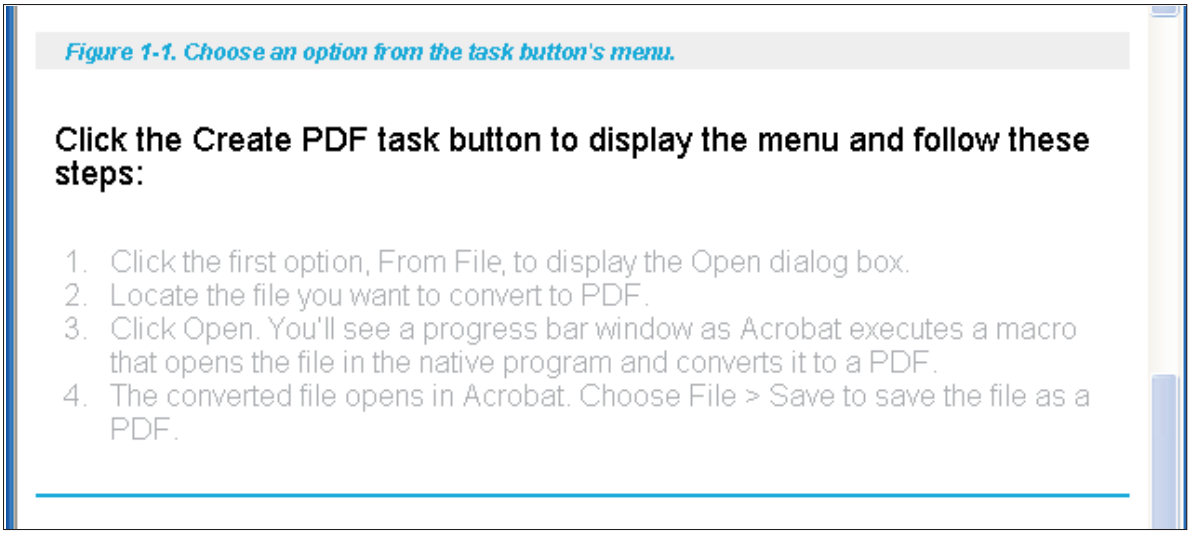
Redefining importance

By default, rules in an author's style sheet override those in a user's style sheet. As listed previously, an `!important` declaration is more important than a normal declaration, and can be used by both author and user style sheets.

The ability to control appearance is extremely useful when designing for those viewers with special visual requirements, such as using high-contrast colors schemes or very large font sizes.

For example, the Web page shown in figure 1-15 shows the application of my styles to the numbered list.

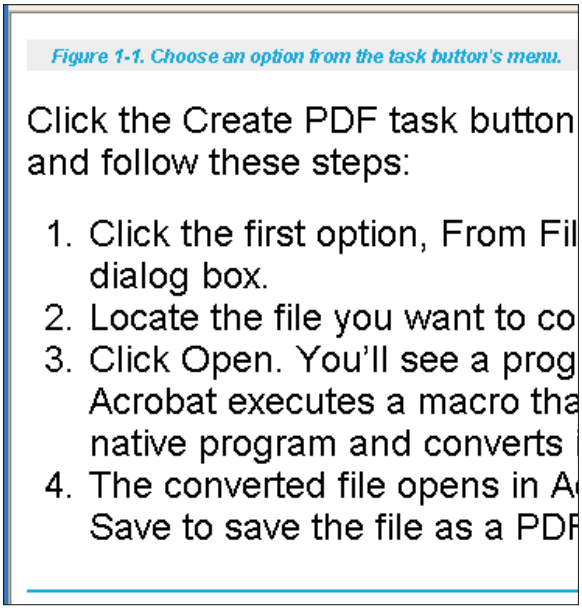
If you look at the same page in figure 1-16, you see the text is much larger and black, resulting from a user's style defining the size and color as `!important`.



1-15

PRO TIP

If you declare a shorthand property as `!important` you declare all its component properties as `!important` as well.



1-16

Building a Business Suite of Styles

Designing styles for your own site is a problem, but how about designing suites of different types of online education materials for numerous clients? Education consultant and training developer Margaret Werdermann has developed a plan that works for her business.

For my business, I have one basic style sheet that I build on and customize for each project. I use this same sheet, not to make all my sites look the same — in fact, the sites I create are always very different from each other — but, instead, as a mental checklist.

In the design phase of a project, I'll go through the basic style sheet, discussing each style and the attributes I commonly set for it. I ask myself, "What do I want to do with this style to make it work best for this design and layout?"

This method was really born of necessity. It took only a couple of projects "reinventing the wheel" to realize that starting from scratch with a blank style sheet might sound terribly creative; but, in fact, it's just a colossal waste of time. Now, I start with my basic

"menu" of styles, each given its own "flavor" for a particular project. Then, because every project ends up having some unique twist, I add to that project's basic sheet. Some of the additions are only really relevant to that project, and they just stay on that sheet. Others, I can see a use for future projects, and they get added to the master style sheet to be used again.

So now I bet you're thinking, "If one master style sheet is a good idea, wouldn't it be a good idea to just prefabricate, say, half a dozen different style sheets and let your customers choose from them?" I mean, it would probably be a lot easier to say, "Choose A, B, or C" than to go through the work of customizing the basic style sheet for each project individually, right? Well, that's true, and I know there are places out there that do exactly that; but I never have and never will.

My customers expect customized sites that express the personalities of their organizations. I feel I owe it to them to get to know them well and make sure their sites work perfectly with their content and appeal to their individual target audiences. You can't do that with a cookie-cutter design.

Sorting Elements

It can be confusing to keep track of what you are working with sometimes, particularly if you are using multiple style sources, like an inline style and an external style sheet.

To keep track, remember the order of priority:

1. An inline style attribute overrides all other styles.
2. A `style` element embedded in a page overrides linked and imported sheets.
3. The `link` element attached as an external style overrides imported styles.
4. The `@import` statement has lowest priority. Imported style sheets cascade with each other in the order in which they are imported.

PRO TIP

You can use the same method as a browser employs to evaluate and test your site's style sheet. Think like a machine. Look for any instances where you can't make a definitive assignment for a style — these instances are the style rules that can cause errors and headaches.

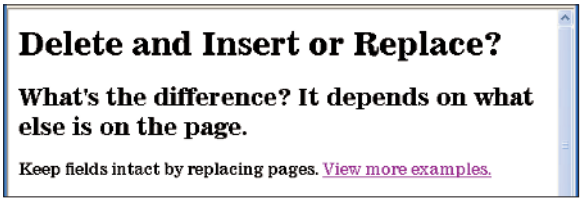
Figuring Out Property Values

Contrary to the title, this isn't a discussion on real estate! A Web browser is more complex software than it may appear at first glance.

To display what you see on a Web page, a browser follows these steps:

1. The document is parsed and a document tree is constructed.
2. The value for each property is calculated.
3. The value is assigned to every property applicable in the target media type.

All properties on a style sheet have a default *initial* value assigned to the root element of the document tree. Common examples are the link colors, underline decoration, and the appearance of headings, shown in the example in figure 1-17.



1-17

The displayed value for a property results from a calculation that may include up to four steps depending on the circumstances of the style sheets in use. The calculations are based on these values, listed in order:

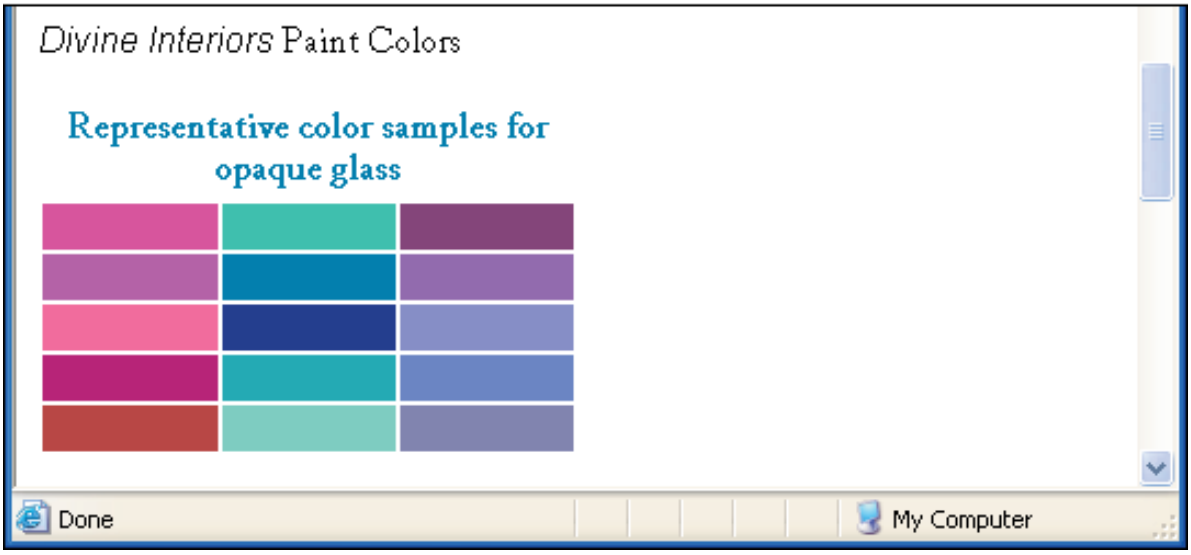
Using Attribute Selectors

Attribute selectors are more specific than general selectors and may be useful in situations using multiple style sheets, like the sets of style sheets written for sites that include a storefront, technical information, and other presentation requirements.

The selector can be defined in varying degrees of specificity as:

- **An attribute assigned to an element. A style is applied only if both the element and attribute are present. For example, `p[title]` would apply the style only to a `<p>` tag that also used the `title` attribute.**
- **The attribute and value. A style is applied only if the element uses a specific class. For example, `p[class=intro]` would assign the style only to those `<p>` tags that also use the `intro` class attribute.**
- **The attribute and value parts. A style is applied only if the attribute value is an exact match of the specified value. For example, the style named `p[class="marcom phase2 intro"]` assigns the style only to those paragraphs using the specific named class containing the words `"marcom phase2 intro"`. Writing the style name as `p[class~="phase2"]` applies the style to those paragraphs using the `"marcom phase2 intro"` class as well as others using the `"phase2"` term as part of the class name.**

1. The *specified* value the CSS specification defines is assigned by a browser based on this order:
   a. If the cascade results in a value, use the value.
   b. If the property is inherited and isn't part of the document tree, use the computed value of the property's parent element.
   c. If neither case is true, use the property's initial value.
2. The *computed* value is determined by inheritance. For example, `em` lengths are computed to pixel or absolute lengths; URI locations are made absolute.
3. The *used* value results from converting to an absolute value. If you are writing a style that uses percentage values for a table width, like the table examples in figure 1-18, the width of the table depends on the width of the browser window. Regardless of the browser window's size, the table is displayed at 50 percent of its width.
4. The *actual* value displays the property using local settings. Ordinarily, the used value is the same as the actual value. If you are using a black-and-white monitor, for example, regardless of the color settings specified in the style sheet, you'll see only shades of gray.



1-18

Taking Control of Your Style Sheet

Writing styles and using the principles of CSS is both fascinating and complex. There are a few methods you can use to make your style sheet as clear and usable as possible.

Although there isn't one best way to proceed, after writing a batch of styles for a new project, I evaluate my style sheet using these steps as a basic checklist:

1. Combine rulesets to omit directions or locations such as "top," "bottom," or "left." If I have written styles in "longhand" for an element, such as a `<blockquote>`, I combine the rulesets to condense the style.

2. Group selectors where styles are repeated. If I realize that `<h2>` and `<h4>` use the same properties and values, I group their styles.

3. Write shorthand properties to simplify the style sheet. Once a style is written and I have seen all the elements both in writing and applied to the page, I usually replace the long style with a shorthand version.

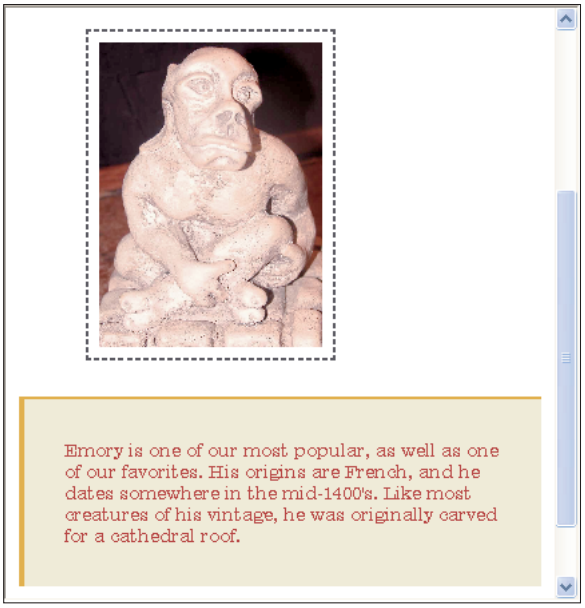
4. Add comments. Lots of comments. I add comments as reminders for myself, such as why I am using one method rather than another, or as headings if I am sorting the styles according to their use in the site.

Combining Rulesets

The CSS box model lets you specify elements on different sides, such as the left or top, right or bottom.

Here's an example. Figure 1-19 shows a paragraph that uses padding and borders. The style `p.callout` is written as:

```
p.callout {
color:#993333;
background-color:#E6E6CC;
padding-top: 30px;
padding-right: 30px;
padding-bottom: 30px;
padding-left: 30px;
border-top: 2px;
border-right: 0px;
border-bottom: 0px;
border-left:4px;
border-color: #CC9933;
border-style: solid;
}
```



1-19

When rulesets are applied uniformly to all margins, for example, I prefer to condense the length of the style by omitting the location in the property. Condensing the style `p.callout` shortens it to:

```
p.callout {
color:#993333;
background-color:#E6E6CC;
padding: 30px;
border-left:4px;
border-top: 2px;
border-bottom: 0px;
border-right: 0px;
border-color: #CC9933;
border-style: solid;
}
```

Grouping selectors

Grouping selectors is an easy way to keep track of similar types of styles such as headings, and saves time spent repeating the styles' rules and troubleshooting.

There are three categories of groups. Check to see if you are using the grouping methods as often as you could:

1. An element, such as a margin or border, can be condensed into a single line by removing the side designation in the property's name, such as "left" or "right" in the property and specifying the dimensions for each side of the object in this order: top, right, bottom, left. For example, the `p.callout` style listed earlier can be condensed further by combining the border widths separated by spaces, written as:

```
p.callout {
color:#993333;
background-color:#E6E6CC;
padding: 30px;
border: 2px 0px 0px 4px;
border-color: #CC9933;
border-style: solid;
}
```

2. A number of selectors that share a style, such as headings, are separated by commas. For example, if you want all the headings in a page to use the same text color, write:

```
h1, h2, h3, h4, h5, h6 {
  color: #0066CC
}
```

3. The most common way to group selectors is to write two or more declarations attached to the same style separated by a semicolon. For example, to define font characteristics for a paragraph, write:

```
p {
   font-family: Verdana, Arial, san
    serif;
   font-weight: bold;
   color: #999999;
}
```

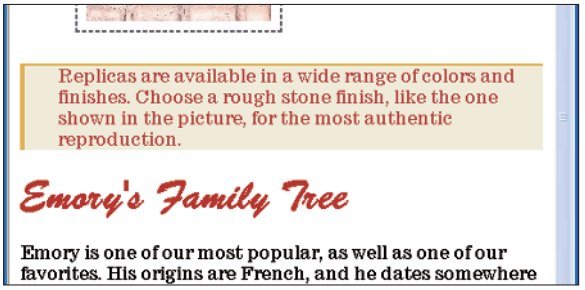
Writing shorthand properties

Once you get into the swing of writing CSS, as in many other things in life, you start to look for ways to streamline your work.

Fortunately, you can specify styles using shorthand properties that decrease the length of your style sheet and save keystrokes and bandwidth. The requirement is that the properties apply to the same style.

Figure 1-20 shows a heading with the following style applied:

```
h1 {
   font-family:"BrushScript
    BT","Times New Roman", serif;
   font-size: 36px;
   font-weight: bolder;
   color:#990000;
}
```



1-20

The shorthand version of the style separates the values by spaces. It is condensed into a single `font` property written as:

```
h1 {
   font:"BrushScript BT","Times New
    Roman",serif
   bolder 36px #330066;
}
```

PRO TIP

You can use shorthand versions for several categories of styles, including background, font, margin, border, padding, and list styles. In the case of the border styles, the `border` property defines a style that applies to all border values. Differentiate sides of the border and apply a shorthand style to the individual sides using `border-top`, `border-bottom`, and so on.

Commenting on style sheets

You may not realize how important comments are until you are faced with revising work you haven't dealt with in recent memory. It's even worse if you are starting from someone else's work.

Insert comments anywhere in a style sheet by enclosing the code or notes you want to hide using `/* COMMENT */` on the style sheet. A comment can be added anywhere you can insert white space. In fact, the comments themselves are treated as white space with one exception: You can't nest a comment within another comment.

If you aren't sure how or when to use comments, imagine you are printing a page of data that has no headings, no references, and no instructions. Any or all of these situations may be perfect places to use comments.

Q & A

Is there a specific order that's best to use for writing a style? Does it matter? When?

For the most part, writing a style is a matter of convenience, habit, and workflow. If it works for you, that's the best method. A browser reads a style in order — the last command or line read is considered the newest.

There are a couple of examples of situations where the order is critical. The most common example is using pseudo-element styles for `<a>` states. The order must list the `a:link` and `a:visited` states first, then the `a:hover` state, and finally the `a:active` state. Explore more in Chapter 10.

Devising workarounds for some browser inconsistencies also requires ordering style properties in a specific way.

What takes precedence — HTML attributes or CSS properties?

CSS properties take precedence over HTML attributes. If both properties and attributes are specified, the HTML attributes are used in browsers without CSS support, but have no effect in CSS-enabled browsers.

How can the box model display issue described in this chapter be resolved?

Several fixes have been developed for preventing the IE box model resizing issue described in this chapter in the sidebar "Box Model Issues." One method produces a workaround based on two `width` properties that must be written in order. The first is a `width` read by all browsers; all those except IE 5x will display the `width` property's value.

```
box {
width:400px;
padding:20px;
border:5px;
width/**/:/**/ 350px;
}
```

By placing empty comment tags (`/**/`) before the colon, IE5.0 ignores the command. Likewise, by placing these empty comment tags after the colon, IE5.5 will ignore the command. By using these two rules in conjunction with each other, the command is hidden from IE5.x.syntax.