Selectors

The *selector* is the part of the style rule that identifies the element (or elements) to which the presentation instructions are applied. For instance, if you want all of the `h1`s in a document to be green, write a single style rule with `h1` as the selector. But that's just the beginning. CSS provides a variety of selector types to improve flexibility and efficiency in style sheet authoring. This chapter introduces the selectors included in the CSS 2.1 specification, including:

- Type (element) selectors
- Contextual selectors (descendant, child, and adjacent sibling)
- Class and ID selectors
- Attribute selectors
- Pseudoclasses
- Pseudoelements

Not all of these forward-thinking selectors are supported by today's browsers, so if a particular selector is not quite ready for prime time, it will be noted.

The W3C Selectors specification introduces additional selectors above and beyond those in CSS 2.1, which modern browsers are still in the process of implementing. This book does not describe them. For more information on those new selectors in particular, see the W3C Selectors specification (*www.w3.org/TR/css3-selectors*).

Type (Element) Selector

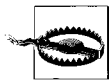
The most straightforward selector is the *type selector* that targets an element by name, as shown in these examples:

```
h1 {color: blue;}
h2 {color: blue;}
p {color: blue;}
```

Type selectors can be grouped into comma-separated lists so a single property will apply to all of them. The following code has the same effect as the previous code:

```
h1, h2, p {color: blue;}
```

CSS 2 introduced a universal element selector (*) that matches any element (like a wildcard). The style rule `* {color: gray}` makes every element in a document gray. The universal selector may be a useful tool when used in relation to other elements, as discussed in the next section.



The universal selector causes problems with form controls in some browsers. If your page contains form inputs, the safest bet is to avoid the universal selector.

Contextual Selectors

Type selectors, such as those in the previous example, apply to all instances of that element found in a document. By contrast, *contextual selectors* allow you to apply style properties to select elements, based on their context or relation to another element. There are several types of contextual selectors: descendant, child, and adjacent sibling. This is where being familiar with the document tree structure of your document comes in handy.

Contextual selectors use a specific character to signify the type of relationship between the elements in the selectors. This character is known as the *combinator*.

Descendant Selector

Descendant selectors target elements that are contained within (therefore descendants of) another element. They are indicated in a list separated by a character space (the combinator for descendant selectors), starting with the higher-level element. For example, the following rule specifies that `em` elements should be olive, but only when they are descendants of a list item (`li`). All other `em` elements are unaffected by this rule.

```
li em {color: olive;}
```

Like simple type selectors, contextual selectors can be grouped together in comma-separated lists. The following code makes emphasized (`em`) text red only when it appears in the context of a first-, second-, or third-level heading:

```
h1 em, h2 em, h3 em {color: red;}
```

Descendant selectors may also be nested several layers deep, as shown in this example that targets only emphasized text (`em`) within anchors (`a`) within ordered lists (`ol`).

```
ol a em {font-weight: bold;}
```

Child Selector

A *child selector* is similar to the descendant selector, but it targets only direct children of a given element. In other words, the element must be contained directly

within the higher-level element with no other element levels in between. Child selectors are separated by the greater-than symbol (>). The rule in the following example makes the background of emphasized text gray, but only when it is the child of a paragraph:

```
p > em {background-color: gray;}
```

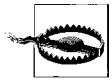
Therefore, in the following markup example, only the first instance of `em` receives a gray background, because the second one is the child of an intervening `strong` element:

```
<p>I've got <em>laser</em> eyes, and <strong>I know what you're <em>
thinking.</em></strong></p>
```

Adjacent Sibling Selector

The *adjacent sibling selector* is used to target an element that comes immediately after another element with the same parent element. The combinator for adjacent sibling selectors is a plus (+) sign. For example, if you wanted to give special presentation treatment to the first paragraph following a first-level heading, the resulting rule would look like this:

```
H1 + p {padding-left: 40;}
```



Browser alert: Child selectors and adjacent sibling selectors are not supported by Netscape 4 or Internet Explorer Version 6 and earlier. Support in Internet Explorer 7, in beta as of this writing, is not yet documented.

Class and ID Selectors

So far, all of the selectors we've seen have been tied to specific elements. *Class selectors* and *ID selectors* give you the opportunity to target elements that you've named yourself, independent of the document element.

Elements are named using the `class` and `id` attributes. They can be applied to all XHTML elements except `base`, `head`, `html`, `meta`, `script`, `style`, and `title`. In addition, `class` may not be used in `basefont` and `param`. Class and ID selectors work in slightly different ways.

class Selector

Use the `class` attribute to identify a number of elements as being part of a conceptual group. Elements in a class can then be modified with a single style rule. For instance, you can identify all the items in a document that you classify as "special":

```
<h1 class="special">Attention!</h1>
<p class="special">You look marvelous today.</p>
```

To specify the styles for elements of a particular class, add the class name to the type selector, separated by a period (`.`).

```
h1.special {color: red;}
p.special {color: red;}
```

To apply a property to all the elements of the same class, omit the tag name in the selector (be sure to leave the period—it is the character that indicates a class):

```
.special {color: red;}
```

Note that `class` names cannot contain spaces; use hyphens or underscores instead if necessary (although underscores are discouraged due to lack of support in some browsers).

When choosing a name for a class, be sure that it is meaningful. For example, naming a class `redtext` merely reintroduces presentational information to the document and does nothing to describe the type of information in the element. It may also be confusing if in a future redesign, the color of those elements changes to blue.

Authors should resist going overboard with class creation (a syndrome commonly referred to as "class-itis"). In many cases, other types of selectors with higher specificity, such as contextual or attribute selectors, may be used instead.

id Selector

The `id` attribute is used similarly to `class`, but it is used for targeting a single element rather than a group. `id` must be used to name an element uniquely (in other words, two elements can't have the same `id` name in the same document). It is not a problem for an `id` value to be used in multiple documents across a site; it only needs to be unique within each document. If you have several elements that need a similar treatment, use `class` instead.

The following example gives a paragraph a specific ID (note that the value of an `id` attribute must always begin with a letter):

```
<p id="j042801">New item added today</p>
```

ID selectors are indicated by the octothorpe (#) symbol within the style sheet as follows:

```
p#j061998 {color: red;}
```

The element name may be omitted:

```
#j061998 {color: red;}
```

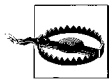
In modern web design, `id` attributes are frequently used to identify main sections (usually `div`s) within a page. Some common `id` values for this purpose are `content`, `header`, `sidebar`, `navigation`, and `footer`. Establishing sections of the page makes it easier to create contextual selectors so that elements can be styled based on where they appear on the page without the need to create extra classes.

Like `class` attributes, `id` names should be chosen based on the semantic role of the element, not its presentation. For example, for a sidebar on the left side of the page that contains news, it is preferable to name the `div id="sidebar-news"` rather than `id="sidebar-left"`.

The `id` attribute is also used in scripting to identify and access unique page objects.

Attribute Selectors

CSS 2 introduced a system for targeting specific attribute names or values. This may be useful for XML languages other than XHTML that may not contain `class` and `id` attributes. There are plenty of uses for attribute selectors within XHTML as well, but unfortunately, attribute selectors are not widely supported at this time.



Browser alert: Attribute selectors are not supported by Internet Explorer Versions 6 and earlier. As of this writing, support is rumored in IE 7, but it has not been documented. Gecko-based browsers (Mozilla and Netscape 6+), Safari, and Opera 7 do support them, but represent a smaller portion of browser usage.

There are four variations on attribute selectors:

Simple attribute selection

The broadest attribute selector targets elements with a particular attribute regardless of its value. The syntax is as follows:

```
element[attribute]
```

Example: `img[title] {border: 3px red;}`

Specifies that all images in the document that include a `title` attribute get a red border.

Exact attribute value

This selects elements based on an attribute with an exact attribute value.

```
element[attribute="exactvalue"]
```

Example: `img[title="first grade"] {border: 3px red;}`

Only images with the title value `first grade` are selected. The value must be an exact character string match.

Partial attribute value

For attributes that accept space-separated lists of values, this attribute selector allows you to look for just one of those values (rather than the whole string). The tilde (~) in the selector differentiates this selector from those that match an exact value.

```
element[attribute~="value"]
```

Example: `img[title="grade"] {border: 3px red;}`

This selector looks for images that contain the word `grade` in the list of title values. Images with the attributes `title="first grade"` or `title="second grade"` would be selected by the example selector.

Hyphen-separated attribute value

This selector is intended to target hyphen-separated values. The selector matches the value you specify, or that value followed by a hyphen. This type

of attribute selector is indicated by a vertical bar (|). This will make more sense in the example.

```
element[attribute|="value"]
```

Example: `*[hreflang|="es"] {color: red;}`

This selector looks for all elements in which the `hreflang` attribute is `es` or begins with `es-`. Elements with the language of their target `href` identified as `es`, `es-ar`, or `es-es` would be selected (in other words, it finds all variations on the Spanish language). Selecting language subcodes is a common use for this type of attribute selector (e.g., to put language flags next to hyperlinks that link to sites and pages of a different language), but by no means its only application.

Pseudoselectors

Style rules are normally attached to elements in the document tree structure, such as those we've discussed in the chapter so far. But some elements are not necessarily found in the document markup, such as which links have been visited or the first line of a paragraph. To apply style to these instances in a document, CSS provides several pseudoselectors. Instead of targeting a particular element in the document, pseudoselectors are interpreted by the browser based on context and function. Pseudoselectors are indicated by the colon (`:`) character. Pseudoselectors are divided into pseudoclasses and pseudoelements.

Pseudoclasses

As the name implies, *pseudoclasses* work as though there is a class applied to a group of elements, most often the anchor (`a`) element. These "phantom" classes (to use Eric Meyer's term) do not appear in the markup, but rather are based on the state of those elements or the document itself.

Anchor pseudoclasses

There are several pseudoclasses that can be used to apply styles to various states of a link:

```
a:link {color: red;}
a:visited {color: blue;}
a:hover {color: fuchsia;}
a:active {color: maroon;}
```

Similar to their `body` attribute counterparts in the `body` element, `:link` applies to hypertext links that have not been visited, `:visited` applies to links to pages that have been visited, and `:active` applies during the act of clicking. The difference is that you can do much more than just change color with CSS. Following are popular rules for turning off the underline under linked text.

```
a:link {color: red; text-decoration: none;}
a:visited {color: blue; text-decoration: none;}
```

The :hover selector is used to create rollover effects in which the link changes in appearance when the mouse pointer moves over it. The examples above turned off underlines for links. The following rule uses :hover to make the underline appear as a rollover.

```
a:link {color: red; text-decoration: none;}
a:hover {color: red; text-decoration: underline;}
```

According to CSS 2, :active and :hover may be used with elements other than anchors, but this use is not supported in Internet Explorer (through Version 6) or Netscape 4.

Love, HA!

Anchor pseudoclasses need to appear in a particular order in a style sheet in order to function properly. The initials LVHA (or according to a popular mnemonic, *love, HA!*) remind developers that the correct order is :link, :visited, :hover, :active. This has to do with order and specificity. Putting :link or :visited last would override the :hover and :active states, preventing those styles from appearing.

Other CSS 2.1 pseudoclasses

In addition to the anchor pseudo-classes, the CSS 2 specification introduced additional pseudoclass selectors. Be warned, however, that they are not well supported at this time.

:focus
: This targets elements that have focus, such as a form element that is highlighted and accepting user input. Although CSS 2 permits :focus to be applied to any element, it is currently only supported for use with the form elements. Netscape 6 supports :focus with a, input, textarea, and select.

  Example: input:focus {background-color: yellow;}

:first-child
: This targets an element that is the first occurring child of a parent element. It allows you to select the first paragraph of a div or the first li in a ul, for example.

  Example: li:first-child {font-weight: bold;}

:lang( )
: This targets an element that targets elements for which a language has been specified. It functions the same as the lang|= attribute selector, but may be more robust.

  Example: p:lang(de) {color: green;}