

# Cascading Style Sheets Level 2

---

**T**he Cascading Style Sheets Level 2 (CSS2) specification was published by the W3C in 1998, surpassing CSS Level 1 to make the formatting of XML and HTML documents more powerful than ever. Of course, CSS2 fights the same backwards-compatibility battles with HTML that CSS1 fought. However, with XML, CSS2 can format content on both paper and the Web almost as well as a desktop publishing program like PageMaker or Quark XPress.



Caution

Most of the rules discussed here are not yet implemented by the common browsers. Internet Explorer 5.0 and Mozilla 5.0 browsers should begin implementing some of these styles, but full implementation is still some time away.

## What's New in CSS2?

CSS2 incorporates many features that Web developers and designers have long requested from browser vendors. The specification has more than doubled in size from CSS1, and is not only a compilation of changes and new features, but a redraft of the original specification. This makes this specification a single source for all Cascading Style Sheet syntax, semantics, and rules.



On the CD-ROM

The complete CSS Level 2 specification is available on the Web at <http://www.w3.org/TR/REC-CSS2> and on the CD in the specs/css2 folder. This is possibly the most readable specification document ever produced by the W3C and is well worth rereading.

As with all new specifications it takes some time for the popular software to support them fully, and CSS2 is no exception. As you will discover while reading through this



### In This Chapter:

What's New in CSS2?

Selecting elements

Formatting a page

Visual formatting

Boxes

Counters and automatic numbering

Aural style sheets



chapter, both Internet Explorer 5.0 and Mozilla 5.0 are just starting to implement these properties. The ones that have not yet been implemented have been noted for your convenience.

The many new features of CSS2 enable you to more precisely select and format elements in your document. New pseudo-classes and pseudo-elements enable you to select the first child of an element, adjust an element when it receives focus, or control the placement of other elements automatically around specified element selections. Media types let you apply different styles to documents that will appear in different media such as printed pages, computer monitors, and radio broadcasts. Support for paged media like printouts and slide shows has been drastically improved with much stronger control over page breaks. Elements can now be formatted in tables as well as block and inline boxes. Sequences and lists can be automatically numbered and indented. More support is provided for non-Western languages like Arabic and Chinese. And for the first time you can apply aural styles that specify not how a document is rendered, but rather how it is read. In addition, CSS2 changes the implementation of some of CSS1's features.

## New Pseudo-classes

Pseudo-classes select elements that have something in common but do not necessarily have the same type. The `:hover` pseudo-class, for example, refers to whichever element the cursor is currently over, regardless of the element's type. CSS2 has seven new pseudo-classes, which are outlined below:

- ♦ `:first-child`: The `:first-child` pseudo-class selects the first child of an element.
- ♦ `:focus`: The `:focus` pseudo-class selects the object that has the focus; that is, the one into which input will go if the user types a key on the keyboard.
- ♦ `:hover`: The `:hover` pseudo-class selects a designated, but not activated object.
- ♦ `:lang`: The `:lang` pseudo-class selects those elements written in a specific language as identified by the `xml:lang` attribute.
- ♦ `:first`: The `:first` pseudo-class selects the first page of a document when it is being printed.
- ♦ `:left`: The `:left` pseudo-class selects the left-hand pages (normally these are the even-numbered pages) of a document printout, as if the hard copy material were going to be in a book.
- ♦ `:right`: The `:right` pseudo-class selects the right-hand pages (normally these are the odd-numbered pages) of a document printout, as if the hard copy material were going to be bound.

## New Pseudo-Elements

Pseudo-elements identify specific elements by information other than what's readily available from the XML input. For example, in CSS1 and CSS2, `:first-line` and `:first-letter` are pseudo-elements that select the first line and letter of an element, even though these aren't necessarily represented by any element.

CSS2 adds two new pseudo-elements, `:after` and `:before`. The `:after` pseudo-element enables you to insert objects after the specified element. These objects can be images, automatic counters, or text. The `:before` pseudo-element enables you to insert objects before a specified element. These objects can also be images, automatic counters, or text.

## Media Types

CSS2 defines ten media types in which information is presented such as Braille, computer displays, ink on paper, and television. CSS2 lets you specify different styles for different media. For example, it's more important to use larger fonts for low-resolution computer displays than for 1200 dpi printing.

## Paged Media

CSS2 provides control over page breaks and methods of identifying individual pages in a document so that designers can format printed documents, without affecting the appearance of the documents on screen.

## Internationalization

As the Internet expands beyond the English-speaking world, more advances are being made in supporting the thousands of languages spoken and written both currently and throughout history. CSS2 adds support for Unicode and bi-directional text so you can style Chinese and Hebrew as easily as English and French.

## Visual Formatting Control

CSS2 adds more formatting properties to provide more precise control over the objects that make up a document. You can now specify the absolute positions and dimensions of elements. There are also more display styles to use when creating elements. Shadows can be applied to text. Fonts and colors can be specified as "the same as" a user interface element like a menu item or an icon label. You can change the cursor shown when the pointer moves over different elements.

## Tables

Improvements in the `display` property make it easy to treat XML elements as table-like structures, better controlling their alignment.

## Generated Content

Automatically generated counters, numbering systems, and list markers enable document authors to force applications to create information on the fly, as the document is being rendered. Numbers can be recalculated on the fly whenever a document changes, rather than having to be painstakingly inserted by hand.

## Aural Style Sheets

In an effort to make information dispersal friendlier for all individuals, CSS2 has incorporated specific properties that cover the features of a speech-synthesizing system. These properties enable the document author to control the richness, pitch, and other properties of the speaker's voice for each element within the document.

## New Implementations

The CSS2 specification also changes the implementation of some features originally included in CSS1. These include the cascade mechanism, pseudo-classes, and a variety of other properties.

### Pseudo-classes and Elements

The `:link`, `:visited`, and `:active` pseudo-classes no longer have to be designated independently of each other, and can be used together.

### Inheritance

In CSS1, only some properties were able to inherit values from their parents. In CSS2, all properties can inherit their value from their parent element by setting the value to the keyword `inherit`. When a property is inherited, the property takes on the same value as the nearest parent element.

Note

Because every property can have the value `inherit`, I will omit any explanation of this value in the discussions of the individual properties that follow.

### Cascade Mechanism

In CSS1 the `!important` designator can force an author's style sheet to take precedence over a reader's style sheet. CSS2 reverses this precedence so that reader preferences take precedence over author preferences. The default result, when working with both author and reader style sheets, is that the user's style sheet overrides the author's. However, if the author declares a property `!important`, this adds more force to the specification, making it override the reader's style sheet. However, if the reader also declares a rule `!important`, this overrides a `!important` declaration in the author's style sheet. In other words, the reader gets the last word.

## Display Property

The default value of the `display` property is now `inline` rather than `block`.

## Margins and Padding

In CSS1, some of the margin properties were ignored when other properties were set, for example, `margin-right` would be ignored if both `margin-left` and `width` were set. This decision was independent of the direction of the text and the alignment of the object. CSS2 makes the decision between altering the left or right margin dependent on the direction of the text of the object.

# Selecting Elements

Browsers that support CSS2, such as Internet Explorer and Mozilla, can more specifically select an element or object to which a style rule is applied. Using CSS2 you can select elements based upon the pattern they create in the document tree, by simply designating their element name, id, or through a combination of element and attribute settings.

## Pattern Matching

CSS2 pattern matching identifies specific elements in the document tree. The syntax of the pattern-matching selector can be anything from a simple element name to a complex system of contextual patterns like those shown in Table 13-1. An element matches a pattern if it meets all of the requirements of the specified pattern. In XML this includes case-sensitivity.

Table 13-1  
CSS2 Selector Syntax for Pattern Matching

<i>Syntax</i>	<i>Meaning</i>
*	This is the universal selector, and matches any element.
X	Matches any element by the name of "X".
X Y	Matches any element with the name "Y" that is a descendent of an element with the name "X". For example: all VERSE descendents of SONNET elements.
X > Y	Matches any "Y" element that is a child of an element "X". For example: all VERSE children of a STANZA element.
X:first-child	Matches all "X" elements that are the first child of their parents. For example: the first STANZA element in a SONNET element.

*Continued*

Table 13-1 (continued)

<i>Syntax</i>	<i>Meaning</i>
X:link	Matches all "X" elements in a link whose target has not yet been visited.
X:visited	Matches all "X" elements in a link whose target has been visited.
X:active	Matches all "X" elements that are currently selected.
X:hover	Matches all "X" elements that currently have the mouse hovering over them.
X:focus	Matches all "X" elements that currently have the focus of the user either through selection by a mouse, or by being ready to input textual data.
X:lang( <i>i</i> )	Matches all "X" elements that are designated to use the human language <i>i</i> using the <code>xml:lang</code> attribute.
X + Y	Matches all "Y" elements whose immediate sibling is an "X" element. For example: a REFRAIN element that is immediately preceded by a STANZA element.
X[attr]	Matches all "X" elements with the "attr" attribute set, no matter what the value of the attribute is. For example: an AUTHOR element with a NAME attribute.
X[attr="string"]	Matches all "X" elements with whose "attr" attribute has the value "string". For example: an AUTHOR element with the DATE attribute with the value 19990723.
X[attr~="string"]	Matches any "X" element whose "attr" attribute is a space-separated list of words of which one is "string".
X[lang="langcode"]	Matches all "X" elements with the "lang" attribute set to a specific "langcode".
X#myname	Matches any "X" element whose <code>id</code> attribute has the value "myname".

## The Universal Selector

The `*` symbol selects all elements in the document. This enables you to set default styles for all elements. For example, this rule sets the default font to New York:

```
* { font-face: "New York" }
```

You can combine `*` with attribute, pseudo-class, and pseudo-element selectors to apply styles to all elements with a specific attribute, attribute value, role, and so forth. For example:

```
*:before      { content: ". " counter(pgraph) ". ";
                counter-increment: pgraph; /*Add 1 to pgraph*/
*[onmouseover] { text-decoration: blink }
```



Tip

If you are using the universal selector with just one other property specification, the \* can be omitted. For example,

```
before { content: ". " counter(pgraph) ". ";
        counter-increment: para }
[onmouseover] { text-decoration: blink }
```

## Descendant and Child Selectors

You can select elements that are children or descendants of a specified type of element with child and descendant selectors. For instance, you can select any VERSE element that is contained within a SONNET element, or only those VERSE elements that are direct children of a STANZA element. Consider Listing 13-1, which shows Shakespeare's 21<sup>st</sup> sonnet in XML.

### Listing 13-1: Shakespeare's 21<sup>st</sup> sonnet

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/css" href="shakespeare.css"?>

<SONNET>
  <AUTHOR>William Shakespeare</AUTHOR>
  <TITLE>Sonnet 21</TITLE>
  <STANZA id="st1">
    <VERSE>So is it not with me as with that Muse</VERSE>
    <VERSE>Stirr'd by a painted beauty to his verse,</VERSE>
    <VERSE>Who heaven itself for ornament doth use</VERSE>
    <VERSE>And every fair with his fair doth rehearse;</VERSE>
  </STANZA>
  <STANZA id="st2">
    <VERSE>Making a couplement of proud compare</VERSE>
    <VERSE>With sun and moon, with earth and sea's rich
      gems,</VERSE>
    <VERSE>With April's first-born flowers, and all things
      rare</VERSE>
    <VERSE>That heaven's air in this huge rondure hems.</VERSE>
  </STANZA>
  <STANZA id="st3">
    <VERSE>O, let me, true in love, but truly write,</VERSE>
    <VERSE>And then believe me, my love is as fair</VERSE>
    <VERSE>As any mother's child, though not so bright</VERSE>
    <VERSE>As those gold candles fix'd in heaven's air.</VERSE>
  </STANZA>
</REFRAIN>
```

Continued

### Listing 13-1 (continued)

```

    <VERSE>Let them say more that like of hearsay well,</VERSE>
    <VERSE>I will not praise that purpose not to sell.</VERSE>
  </REFRAIN>
</SONNET>

```

All `VERSE` elements are descendants of the `SONNET` element, but not immediate children. Some `VERSE` elements are immediate children of `STANZA` elements and some are immediate children of the `REFRAIN` element. Descendant selectors are made up of two or more element designators separated by a space. A descendant selector of the form `SONNET VERSE` matches a `VERSE` element that is an arbitrary descendant of a `SONNET` element. In order to specify a specific layer of descendant, you need to use the form `SONNET * VERSE` which forces the `VERSE` element to be at least a grandchild, or lower descendent of the `SONNET` element.

To specify an immediate child element, you use the form `STANZA > VERSE`. This applies the rule only to `VERSE` elements that are a direct child of a `STANZA` element, and therefore won't affect any `VERSE` children of a `REFRAIN` element.

You can combine both descendant and child selectors to find specific elements. For example the following selector finds all `VERSE` elements that are the first child of a `REFRAIN` element that is in turn a descendant of a `SONNET` element.

```
SONNET REFRAIN>VERSE:first { padding: "2cm" }
```

Applied to Listing 13-1, this rule selects the verse “Let them say more that like of hearsay well,”.

## Adjacent Sibling Selectors

Adjacent sibling selectors use a `+` sign between element designators to identify an element that follows another element at the same level of the hierarchy. For example, the following code selects all `REFRAIN` elements that share a parent with a `STANZA` element and immediately follow the `STANZA` element.

```
STANZA+REFRAIN {color:red}
```

## Attribute Selectors

Attribute selectors identify specific element/attribute combinations. Place the name of the attribute being matched in square brackets after the name of the element. For example, this rule turns all `STANZA` elements with a `NUMBER` attribute red:

```
STANZA[NUMBER] { color: red }
```

This rule turns all STANZA elements that have a NUMBER attribute red, regardless of the value of that attribute. This includes elements that have a default NUMBER attribute provided by the DTD, but not STANZA elements that don't have a NUMBER attribute.

To test attribute values, you use the same syntax you use to set an attribute value; that is, the name followed by an equals sign, followed by the value in quotes. For instance, to specify that only STANZA elements whose NUMBER attribute has the value 3 should be turned red; you would use this rule:

```
STANZA [NUMBER="3"] { color: red }
```

## @rules

@rules do something other than select an element and apply some styles to it. There are five of them:

1. @page: applies styles to a page (as opposed to elements on the page)
2. @import: embeds an external style sheet in the current style sheet
3. @media: groups style rules for attributes that should only be applied to one kind of media
4. @font-face: describes a font used elsewhere in the style sheet
5. @charset: defines the character set used by the style sheet

## @page

The @page rule selects the page box. Inside it the designer can specify the dimensions, layout, orientation, and margins of individual pages. The page box is a rectangular area, roughly the size of a printed page, which contains the page area and the margin block. The page area contains the material to be displayed, and the edges of the box provide a container in which page layout occurs between page breaks. Unlike other boxes, page boxes do not have borders or padding, only margins.

The @page rule selects every page of a document. You can use one of the page pseudo-class properties, :first, :left, or :right, to specify different properties for various classes of pages.

Because the @page rule is unaware of the page's content including its fonts, it can't understand measurements in ems and ex's. All other units of measurement are acceptable, including percentages. Percentages used on margin settings are a percentage of the total page box. Margins can have negative values, which place content outside of the area normally accessible by the application or printer. In most cases, the information is simply truncated to the visible or printable area.

## @import

The @import rule embeds a specified external styles sheet into an existing style sheet. This enables you to build large style sheets from smaller, easier-to-

understand pieces. Imported style sheets use a .css extension. For example, the following rule imports the poetry.css file.

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

@import rules may specify a media type following the name of the style sheet. If no media type is specified, the @import rule is unconditional, and will be used for all media types. For example, the following rule imports the printmedia.css file. The declarations in this style sheet will only be applied to print media.

```
@import url(printmedia.css) print;
```

The next rule imports the continuous.css file that will be used for both computer monitors and/or television display:

```
@import url(continuous.css) tv, screen;
```

Style sheets that are imported into other style sheets rank lower in the cascade than the importing style sheet. For example, suppose shakespeare.css styles a VERSE in the New York font while shakeprint.css styles a VERSE in the Times font. If shakespeare.css imports shakeprint.css, then the verses will be styled in New York. However, if shakeprint.css imports shakespeare.css, then verses will be styled in Times.

## @media

Many types of media are used to impart information to readers, and each media type has its own customary styles and formats. You can't very well have a speech synthesizer reading Shakespeare in a monotone, now can you? And italics don't make much sense on a monospaced terminal.

CSS2 allows you to specify different styles for the same element displayed in different media. For example, text is easier to read on the screen if it uses a sans serif font, while text on paper is generally easiest to read if it is written in a serif font. You can enclose style rules intended for only one medium in an @media rule naming that medium. There can be as many @media rules in a document as there are media types to specify. For example, these rules format a SONNET differently depending on whether it's being printed on paper or displayed on a screen.

```
@media print {
  SONNET { font-size: 10pt; font-family: Times, serif }
}
@media screen {
  SONNET { font-size: 12pt;
           font-family: New York, Times New Roman, serif }
}
@media screen, print {
  VERSE { line-height: 1.2 }
}
```

The first two rules define styles specific to the print and screen media types respectively. Since modern computer displays have much lower resolutions than modern printers, it's important to make the font larger on the screen than on the printout and to choose a font that's designed for the screen.

The third rule provides styles that apply to both of these media types. To designate style instructions for multiple media types simultaneously, you simply list them following the @media rule designator, separated by a comma.

Browsers that support CSS2 allow the document author to provide rules governing how a document will be displayed for a particular type of media. For instance, it would likely apply different rules when showing a document on the screen than when sending it to a printer. CSS2 identifies ten media types. These are:

1. **all** (all devices)
2. **aural** (continuous, aural): speech synthesizers
3. **braille** (continuous, tactile): Braille tactile feedback devices for the sight impaired
4. **embossed** (paged, tactile): paged Braille printers
5. **handheld** (visual): PDAs and other handheld devices such as Windows CE palmtops, Newtons, and Palm Pilots.
6. **print** (paged, visual): all printed, opaque material
7. **projection** (paged, visual): presentation and slide shows, whether projected directly from a computer or printed on transparencies
8. **screen** (continuous, visual): bitmapped, color computer displays
9. **tty** (continuous, visual): dumb terminals and old PC monitors that use a fixed-pitch, monochromatic character grid
10. **tv** (aural/visual): television-type devices, i.e. low-resolution, analog display, color

Browsing software does not have to support all of these types. In fact, I know of no single device that does support all of these. However, style-sheet designers should probably assume that readers will use any or all of these types of devices to view their content.

Of course, the characteristics of individual media change over time. My first printer was 144dpi, but such low-resolution printers should be relatively rare in the 21<sup>st</sup> century. On the other hand, monitors will eventually reach resolutions of 300 dpi or more; and color printing is rapidly becoming accessible to more and more users.

Some properties are only available with specific media types. For instance, the *pitch* property only makes sense with the aural media type. CSS2 does not specify an all-inclusive list of media types, although it does provide a list of current values for the @media rule. These values are not case-sensitive.

## @font-face

The `@font-face` rule provides a description of a typeface used elsewhere in the style sheet. It can provide the font's name, a URL from which the font can be downloaded, and detailed information about the metrics of the font that allow a reasonable facsimile to be synthesized. The `@font-face` rule also controls how the software selects the fonts for a document with author-specified fonts. You can suggest identical font matching, intelligent font matching, synthesizing the requested font, downloading the fonts from the server, or rendering the font. These methods are described below:

- ♦ **Identical Font Matching:** The user's software chooses the local system font with the same family name. Fonts with the same name may not necessarily match in appearance. The font the client is using could have originated from a different source than the font located on the server.
- ♦ **Intelligent Font Matching:** The software chooses a font that is available on the client system and is closest in appearance to the requested font. This is not an exact match, but it should be close. The font is matched based on font type, whether it uses serifs, its weight, the height of its capital letters, and other font characteristics.
- ♦ **Font Synthesis:** The Web browser builds a font that closely resembles the designated font, and shares its metrics. When a font is synthesized, it will generally be a closer duplicate than a font found by matching. Synthesis requires accurate substitution and position information in order for all the font characteristics to be preserved.
- ♦ **Font Download:** The browsing software downloads the font from a specified URL. The process is the same as downloading an image or sound to be displayed with the current document. Users that download fonts will experience delays similar to those that occur when downloading images.
- ♦ **Font Rendering:** The last alternative for managing fonts is progressive rendering. This is a combination of downloading and matching which enables the browser to create a temporary font so a document's content can be read while the original font downloads. After the "real" font has been downloaded, it replaces the synthesized font in subsequent documents. To avoid having a document rendered twice, your font description must contain the metric information describing the font. The more complete a font's metric information, the less likely a document will need to be re-rendered once the download is complete.

CSS2 enables the document author to specify which of these methods, if any, are used when a designated font is not available on the reading system. The `@font-face` rule provides a font description, created out of a series of font descriptors, defining detailed information about the fonts to be used on the page. Each font descriptor characterizes a specific piece of information about the font. This description can include a URL for the font, the font family name, and the font size.

Font descriptors are classified into three types:

- ♦ Those that provide a link between the style sheet usage of the font and its description.
- ♦ Those that provide a URL for the location of the font or its pertinent information.
- ♦ Those that provide character information for the font.

The `@font-face` rule applies only to the fonts specified within the style sheet. You will need one `@font-face` specification for each font in the style sheet. For example:

```
@font-face { font-family: "Comic Sans";
             src: url(http://metalab.unc.edu/xml/fonts/comicsans)}
@font-face { font-family: "Jester"; font-weight: bold;
             font-style: italic}
TITLE      { font-family: "Comic Sans"}
AUTHOR     { font-family: "Jester", serif}
```

As the software reads this style sheet, it will try to find a set of rules that specify how each element should be rendered. The style sheet sets all `TITLE` elements to the Comic Sans font family, at the same time it sets all `AUTHOR` elements to the Jester font. A Web browsing application that supports CSS1 will search for the Comic Sans and Jester font families. If it can't find them, then it will use its default text font for the Comic family, and the specified fall-back serif font for the Jester family. The `@font-face` rule's font descriptors will be ignored. CSS1 software will be able to safely skip over this command without encountering an error.

Applications that support CSS2 will examine the `@font-face` rules in an attempt to match a font description to the Comic Sans and Jester fonts. In the above example, the browsing software will find a URL from which it can download the Comic Sans font. If Comic Sans were found on the client system, the software would have used that instead of downloading the font. In the case of Jester, the users software will use one of the matching rules, or the synthesis rule to create a similar font from the descriptors provided. If the Web browser could not find a matching `@font-face` rule for the font family specified, it would have attempted to match the fonts using the rules specified for CSS1.

CSS2 allows any font descriptor that is not recognized, or useful to the browser, to be skipped. This provides a built-in means for increasing the descriptors in an effort to improve the font substitution, matching, or synthesis rules being used.

## @charset

There are three ways to specify the character set in which a style sheet is written, and they take precedence in the following order:

1. An HTTP “charset” parameter in a “Content-Type” field
2. The @charset rule
3. Attributes and properties associated with the document, such as HTML’s charset attribute used with the LINK element

Each style sheet can contain a single @charset rule. The @charset rule must appear at the very beginning of the document, and can not be preceded by any other characters. The syntax for using @charset is:

```
@charset "character set name"
```

The character set name specified in this statement must be a name as described in the IANA registry. You can see a partial list of character sets in Table 7-7 in Chapter 7. To specify that a style sheet is written in Latin-1, you would write:

```
@charset "ISO-8859-1"
```



Character sets are discussed in great detail in Chapter 7, *Foreign Languages and non-Roman Text*.

## Pseudo Elements

Pseudo-elements are treated as elements in style sheets but are not necessarily particular elements in the XML document. They are abstractions of certain parts of the rendered document after application of the style sheet; for example, the first line of a paragraph. Pseudo-elements are not case-sensitive, and may only appear directly after the subject of a style-sheet selector. CSS2 introduces two new pseudo-elements: :after and :before.

The :before and :after pseudo-elements select the location immediately before and after the element that precedes them. The content property is used to put data into this location. For example, this rule places the string — between STANZA objects to help separate the stanzas. The line breaks are encoded as \A in the string literal:

```
STANZA:after {content: "\A——\A"}
```

As well as a literal string, you can use one of these four keywords as the value of the content property:

1. open-quote
2. close-quote
3. no-open-quote
4. no-close-quote

The `open-quote` and `close-quote` keywords insert the appropriate quote character for the current language and font (for example, “ or ’). The `no-open-quote` and `no-close-quote` keywords do not insert any characters, but increment the level of nesting as if quotes were used. With each level of nesting, the quote marks switch from double to single or vice versa.

You can also use the `attr(X)` function as the value of the content property to insert the value of the X attribute before or after the identified element.

Finally, you can insert the current value of an automatic counter using either the `counter()` or `counters()` function. This has two distinct forms: `counter(name)` or `counter(name, style)`. The default style is decimal.

## Pseudo Classes

Pseudo-class selectors select elements based on aspects other than the name, attributes or content of the element. For example, a pseudo-class may be based on the position of the mouse, the object that has the focus, or whether an object is a link. An element may repeatedly change its pseudo-classes as the reader interacts with the document. Some pseudo-classes are mutually exclusive, but most can be applied simultaneously to the same element, and can be placed anywhere within an element selector. When pseudo-classes do conflict, the cascading order determines which rules are activated.

### **:first-child**

The `:first-child` pseudo-class selects the first child of the named element, regardless of its type. For example, in Listing 13-1 the `VERSE` element whose contents are “So is it not with me as with that Muse” would be the first child of the `STANZA` element and would be designated by this rule:

```
STANZA:first-child {font-style: bold}
```

### **:link, :visited, :active**

In CSS1 `:link`, `:visited`, and `:active` pseudo-classes are mutually exclusive. In CSS2, `:link` and `:visited` are still mutually exclusive (as they logically have to be), but you can use either of these in conjunction with `:active`. For example, the following code fragment assumes the `AUTHOR` element has been designated as a link, and alters the colors of the text depending upon the current state of the link. In the following code fragment, an unvisited link is set to red, a visited link will be displayed as gray, and an active link will be shown as lime green while the cursor is being placed over it.

```
AUTHOR:link      { color: "red" }  
AUTHOR:visited  { color: "gray" }  
AUTHOR:active   { color: "lime" }
```

**:hover**

The `:hover` pseudo-class selects elements which the mouse or other pointing device is pointing at, but without the mouse button depressed. For instance, this rule colors the `AUTHOR` element red when the cursor is pointing at it:

```
AUTHOR:hover { color: "red" }
```

The `AUTHOR` element returns to its normal color when the cursor is no longer pointing at it.

**:focus**

The `:focus` pseudo-class refers to the element that currently has the focus. An element has the focus when it has been selected and is ready to receive some sort of text input. The following rule makes the element with the focus bold.

```
:focus { text-style: "bold" }
```

**:lang()**

The `:lang()` pseudo-class selects elements with a specified language. In XML this is generally done via the `xml:lang` attribute and/or the `encoding` attribute of the XML declaration. The following rule changes the direction of all `VERSE` elements written in Hebrew to read right to left, rather than left to right:

```
VERSE:lang(he) {direction: "rtl" }
```

**:right, :left, :first**

The `:right`, `:left`, and `:first` pseudo-classes are only applied to the `@page` rule. They enable you to specify different styles for the first page of a document, for the left (generally even-numbered) pages of a document, and for the right (generally odd-numbered) pages of a document. For example, these rules specify very large margins:

```
@page:right { margin-top: 5cm;
              margin-bottom: 5cm;
              margin-left: 7cm;
              margin-right: 5cm }
@page:left  { margin-top: 5cm;
              margin-bottom: 5cm;
              margin-left: 5cm;
              margin-right: 7cm }
@page:first { margin-top: 10cm;
              margin-bottom: 10cm;
              margin-left: 10cm;
              margin-right: 10cm }
```

The only properties you can set in a rule for these pseudo-classes are the margin properties.

## Formatting a Page

The `@page` selector refers to a page. It's used to set properties that apply to the page itself rather than an individual XML element on the page. Each page of a document has a variety of properties applied to it, including the page size, orientation, margins, and page breaks. These properties cascade to any element placed on the page. Optional pseudo-classes can specify different properties for the first page, right-facing pages, and left-facing pages.

CSS2 makes the reasonable assumption that pages are rectangular. Given that assumption, a page can possess the box properties you're familiar with from CSS1 including margins and size. However, a page box does not have borders or padding since these would naturally fall outside the physical page.

### Size Property

In an `@page` rule, the `size` property specifies the height and width of the page. You can set the size as one or two absolute lengths or as one of the four keywords `auto`, `portrait`, `landscape`, or `inherit`. If only one length is given, the page will be a square. When both dimensions are given, the first is the width of the page; the second is the height. For example,

```
@page { size: 8.5in 11in }
```

The `auto` setting automatically sizes to the target screen or sheet. `landscape` forces the document to be formatted to fit the target page, but with long sides horizontal. The `portrait` setting formats the document to fit the default target page size, but with long sides vertical.

### Margin Property

The `margin` property controls the margins of the page — the rectangular areas on all four sides in which nothing is printed. This property is used as a shorthand for setting the `margin-top`, `margin-bottom`, `margin-right`, and `margin-left` properties separately. These properties are the same as they are for boxes in CSS1. For example, this rule describes an 8.5 by 11 inch page with one-inch margins on all sides.

```
@page { size: 8.5in 11in; margin: 1.0in }
```

### Mark Property

CSS2 offers the `mark` property to make marks to appear on a page delineating where the paper should be cut and/or how pages should be aligned. These marks appear outside of the page box. A page box is simply the viewable area of the document that can be affected by the `@page` rule. If you were to look at a printed 8 1/2" x 11"

piece of paper, the page box would be everything inside the printable region on that paper, what we normally think of as the space inside the printer margins. The software controls the rendering of the marks, which are only displayed on absolute page boxes. Absolute page boxes cannot be moved, and are controlled by the general margins of the page. Relative page boxes are aligned against a target page, in most cases forcing the marks off the edge of the page. When aligning a relative page box, you are essentially looking at the page in your mind's eye, and using `margin` and `padding` properties to move the printed area of that page about the physical paper.

The `mark` property has four possible values—`crop`, `cross`, `inherit`, and `none`—and can only be used with the `@page` element. Crop marks identify the cutting edges of paper. Cross marks, also known as registration marks, are used to align pages after printing. If set to `none`, no marks will be displayed on the document. The following rule specifies a page with both crop and cross marks:

```
@page { mark: crop cross }
```

## Page Property

As well as using the `@page` selector to specify page properties, you can attach page properties to individual elements using the `page` property. To do this you write an `@page` rule that specifies the page properties, give that `@page` rule a name, and then use the name as the value of the `page` property of a normal element rule. For example, these two rules together say that a `SONNET` will be printed in landscape orientation.

```
@page rotated { size: landscape }
SONNET       { page: rotated }
```

When using the `page` property, it's possible for different sibling elements to specify different page properties. If this happens, a page break will be inserted between the elements. If a child uses a different page layout than its parent, the child's layout will take precedence. For instance, in the following example the two tables are rendered on landscape pages, possibly on the same page if space allows. Because of the layering of the elements in the document, the assignment of the rotated page to the `SONNET` element is over ridden, and not used.

```
@page narrow { size: 9cm 18cm }
@page rotated { size: landscape }
STANZA       { page: narrow }
SONNET       { page: rotated }
```

## Page-Break Properties

The `page-break-after` property forces or prohibits the insertion of a page break after the current object. The `page-break-before` property forces or prohibits the insertion of a page break before the current object. The `page-break-inside` property allows or prohibits the insertion of a page break inside the current object. These can be used to keep together paragraphs of related text, headings and their body text, images and their captions, or to keep complete tables on the same page.

When either of these properties is set to `auto`, a page break is neither forced nor prohibited after the current box. A setting of `always` forces a page break. The `avoid` setting prevents a page break from appearing. The `left` and `right` settings force the insertion of either one or two page breaks as necessary in order to force the next page to be either a left- or right-hand page. This is useful at the end of a chapter in a book in which chapters generally start on right-hand pages, even if it leaves blank pages.

The following rule inserts a page break before and after every `SONNET` element in a document but not inside a sonnet so that each sonnet appears on its own page.

```
SONNET { page-break-before: always;
         page-break-after:  always;
         page-break-inside: avoid }
```

## Visual Formatting

CSS2 adds many new formatting features that provide more control over the layout of your XML document. The `display` property has many new values that expand on the basic block and inline types of CSS1. The `cursor` property enables you to identify what sort of cursor to display over your object. You can control the height and width of all object boxes. CSS2 also gives you the ability to modify your document objects' visibility, clipping size, color, font, text shadows, alignment, and control how an object's contents are dealt with if overflow should occur.

### Display Property

The expansion of the `display` property in CSS2 provides more complete layout options, most notably tables. In CSS2, there are 17 possible values of the `display` property:

```
inline
block
list-item
```

```
run-in
compact
marker
table
inline-table
table-row-group
table-header-group
table-footer-group
table-row
table-column-group
table-column
table-cell
table-caption
none
```

Block elements are drawn by breaking out space around the objects forcing buffer of space around their contents. Inline elements work without setting aside separate space. Table elements are various parts of a grid. Inline elements are like a word in a sentence. Their position moves freely as text is added and deleted around them. Block objects are more fixed and at most move up and down but not left and right as content is added before and after it. Block items include such items as tables, lists, and list items. Most display types are just modifications of the main block or inline types.

### Inline Objects

Inline object boxes are laid horizontally in a row starting from the top of the containing box of the surrounding page or block element. Between these boxes the variety of horizontal margins, borders, and padding spaces are implemented. You can also align these types of boxes vertically in a variety of ways including character baselines, box bottoms, or box tops.

**Note**

In CSS1, the `block` value was the default display type of all objects, but that has changed in CSS2. Elements are now automatically displayed as `inline` unless otherwise designated.

### Block Objects

Block objects are laid out vertically, one on top of the other. The first block is laid in the top left corner of the containing block, then the second block is placed below it, also flush against the left edge of the containing block. The vertical distance between each block is defined by the individual block's margin and padding properties. For example, this rule identifies the VERSE, STANZA, and REFRAIN

elements as individual blocks. Figure 13-1 shows Listing 13-1 when this rule (and only this rule) is applied. Note that the `AUTHOR` and the `TITLE` are on the same line because they are inline by default. However, when a block element follows an inline element, a line break is required after the block element.

```
VERSE, STANZA, REFRAIN { display: block }
```

**Figure 13-1:** When displayed as block elements, Shakespeare's sonnet starts to take on a more normal appearance.

## None

The value of `none` forces the element to not generate a display box of any kind for formatting the content of the element. In other words, the element will not have any effect on the layout of the document. Child and other descendant elements don't generate boxes either, even if the `display` property is set for them. When `display` is `none`, the box is not just invisible; it actually does not exist.

## Compact and Run-in Values

The `compact` and `run-in` values of the `display` property identify an element as either a block or an inline box depending on context. Properties used on items declared as these types will be effective based upon their final rendered status. A `compact` box is placed in the margin of the block box that follows it if it will fit. If the box that follows it is not a block box, or the `compact` box will not fit in the margin, then it is rendered simply as another block box.

The `run-in` value enables you to format normal block elements as the first inline block of the next block element in the code. If the next element is not a block element, then the `run-in` element is formatted as a block element.

## Marker Value

Setting the `display` property to the `marker` value identifies a block that's formed by content generated in the style sheet rather than copied in from the XML document. This value is only used with the `:before` and `:after` pseudo-elements that have been attached to block-level elements.

## Table Display Values

One of the most important new features in CSS2, especially for XML developers who often create tabular structures with tags that look nothing like HTML's table tags, is support for table layout of elements. CSS2 adds support for styling elements as parts of tables using these ten values of the `display` property:

1. `table`
2. `inline-table`
3. `table-row-group`

4. `table-header-group`
5. `table-footer-group`
6. `table-row`
7. `table-column-group`
8. `table-column`
9. `table-cell`
10. `table-caption`

For example, setting the `display` property to `table` indicates that the selected element is a block-level container for various smaller children that will be arranged in a grid. The `inline-table` value forces the table to function as an inline object, enabling text to float along its side, and for multiple tables to be placed side by side. The `table-caption` value formats an element as a table caption. The `table-row-group`, `table-header-group`, and `table-footer-group` values create groups of data cells that work as a single row, as if it was defined using the `table-row` value. The `table-column-group` creates a group of data cells that work as a single column that was defined using the `table-column` value. XML elements that appear in table cells have — naturally enough — a `display` property with the value `table-cell`.

For example, if you were to configure a sonnet in a table-like structure, you might set each `STANZA` and `REFRAIN` to be a table and each `VERSE` to be a table row. The style sheet to create this effect might include these three rules:

```
STANZA { display: table }
REFRAIN { display: table }
VERSE  { display: table-row }
```

## Width and Height Properties

The default height of a box in which each element appears is calculated from the combined height of the element's contents. The default width of each element's box is calculated from the combined width of the element's contents or the width of the viewable area on the page or the screen. Inline elements and table elements that contain text always have these automatically calculated dimensions. However, the style-sheet designer can change these defaults for block-level elements and replaced inline elements by specifying values for six properties:

1. `min-width`
2. `max-width`
3. `min-height`

4. `max-height`
5. `height`
6. `width`

The `min-height` and `min-width` properties specify the smallest dimensions that the object can be displayed with. The maximum properties are a maximum size for the box regardless of the total size of its contents. The Web browser is free to adjust the size of the box within these limits. However, if `height` and `width` are set, then they determine exactly the size of the box.

```
STANZA { width: 100px;
          Height: 100px }
```

## Overflow Property

When the size of a box is precisely specified using the `width` and `height` properties, it's entirely possible that its contents may take up more area than the box actually has. The `overflow` property controls how the excess content is dealt with. This property can be set to one of four values:

1. `auto`
2. `hidden`
3. `scroll`
4. `visible`

If `overflow` is set to `auto`, scroll bars are added if necessary to enable the user to see excess content. If `overflow` is set to `hidden`, the excess content is simply truncated. If `overflow` is set to `scroll`, scroll bars are added whether there's overflow or not. Finally, if `overflow` is set to `visible`, the complete contents are shown, if necessary by overriding the size constraints that were placed on the box. Figure 13-2 shows the sonnet when the `STANZA`'s `overflow` property is set to `scroll` with this rule:

```
STANZA { overflow: scroll }
```

The Shakespeare sonnet's stanzas with scroll bars

## Clip Property

The `clip` property identifies the portion of an object's content that will be visible when rendered by the user's software. Generally the clipping region will match the outside borders of the element's box, but the region can be altered. This property applies only to elements with an `overflow` attribute that is set to a value other than `visible`.

In CSS2, you can only clip to rectangular regions. Set the `clip` property to `rect(top, bottom, left, right)` where `top`, `bottom`, `left`, `right` are the offsets on each side. If the clipped object still exceeds the viewable area of the browser's window; the contents will be further clipped to fit in the window. The following rule uses the `clip` property with a `STANZA` block element:

```
STANZA { clip: rect(5px, 5px, 5px, 5px);
          overflow: auto }
```

## Visibility Property

The `visibility` property controls whether the contents of an element are seen. The four possible values of this property are:

1. `visible`
2. `hidden`
3. `collapse`
4. `inherit`

If `visibility` is set to `visible`, the contents of the box, including all borders are shown. If `visibility` is set to `hidden`, the box's contents borders are not seen. Invisible boxes still take up space and affect the layout of the document. Setting `visibility` to `hidden` is not the same as setting `display` to `none`.

If `visibility` is set to `collapse`, it is the same as `hidden` for any object, except a table row or column. However, for table rows and columns, it completely hides (as with `display: none`) the row or column.

## Cursor Property

The cursor is the arrow/hand/insertion bar/other icon that indicates the position of the pointer on the screen. A cursor is the visible representation of your mouse's logical position that is displayed on the viewable area of your computer monitor. The `cursor` property specifies the cursor a user's software should display when a reader moves the mouse over a particular object. CSS2 allows these 16 cursor values:

1. `auto`: the browser chooses a cursor based on the current context. This is the default value
2. `crosshair`: a simple cross-hair cursor
3. `default`: the platform-dependent default cursor, usually an arrow
4. `hand`: a hand

5. `move`: crossed arrows indicating something to be moved
6. `e-resize`: east-pointing arrow (up is north)
7. `ne-resize`: northeast-pointing arrow
8. `nw-resize`: northwest-pointing arrow
9. `n-resize`: north-pointing arrow
10. `se-resize`: southeast-pointing arrow
11. `sw-resize`: southwest-pointing arrow
12. `s-resize`: south-pointing arrow
13. `w-resize`: west-pointing arrow
14. `text`: I-beam
15. `wait`: stop watch, spinning beach ball, hourglass or other icon indicating the passage of time
16. `help`: question mark

The following rule uses the `cursor` property to say that the hand cursor should be used when the pointer is over a `VERSE` element.

```
VERSE { cursor: hand }
```

You can also use a custom cursor that's loaded from an image file by giving a URL for the image. Generally you'll provide cursors in several formats in a comma-separated list, the last of which is the name of a generic cursor. For example:

```
VERSE { cursor: url("poetry.cur"), url("poetry.gif"), text }
```

## Color-Related Properties

CSS2 identifies colors as RGB values in the Standard Default Color Space for the Internet (sRGB). The way these colors are represented varies from browser to browser, but this specification provides an unambiguous and objectively measurable definition of a color's appearance. Web browsers that conform to the standard perform a gamma correction on the colors identified by the CSS2 specification. sRGB identifies a display gamma of 2.2 under most viewing conditions. This means that for most computer hardware, the colors given through CSS2 properties will have to be adjusted for an effective display gamma of 2.2.

**Note**

Only colors identified in CSS2 rules are affected. Colors used in images are expected to carry their own color correction information.

## Color Property

The `color` property specifies the foreground color for the text content of an element. It may be given as a literal color name like `red` or an RGB value like `#CC0000`. Color names (or values) include `aqua`, `black`, `blue`, `fuchsia`, `gray`, `green`, `lime`, `maroon`, `navy`, `olive`, `purple`, `red`, `silver`, `teal`, `white`, and `yellow`.

The following style rules apply `color` to three elements, using all three methods of identifying color. It specifies the RGB hex value `#FF0000` for `AUTHOR` elements, all `TITLE` elements to appear in red, and all `VERSE` elements to appear in `rgb(255,0,0)`. These values are all red:

```
AUTHOR { color: #FF0000}
TITLE  { color: red}
VERSE  { color: rgb(255,0,0) }
```

## Gamma Correction

At its most basic, gamma correction controls the brightness of images so they are displayed accurately on computer screens. Images that have not been corrected can appear bleached-out or too dark. In order to make gamma correction easier to understand, let's look at the images displayed on your computer screen.

Practically every computer monitor has a gamma of 2.5. This means that its intensity to voltage curve is roughly a function of the power 2.5. If you send your monitor a message for a specific pixel to have an intensity of  $x$ , that pixel will automatically have an intensity of  $x^{2.5}$  applied to it. Because the range of voltage is between 0 and 1, this means that your pixel's intensity is lower than you wish. To correct this, the voltage to the monitor has to be "gamma corrected."

The easiest way to correct this problem is to increase the voltage before it gets to the monitor. Since the relationship between the voltage and the brightness is known, the signal can be adjusted to remove the effect of the monitor's gamma. When this is done properly, the computer display should accurately reflect the image input. Of course, when you are gamma correcting an image, the light in your computer room, the brightness and contrast settings on your monitor, and your personal taste will also play a role.

When attempting to do gamma correction for the Web, platform idiosyncrasies come into play. Some UNIX workstations automatically correct for gamma variance on their video card, as does the Macintosh, but most PCs do not. This means that an image that looks good on a PC will be too light on a Mac; and when something looks good on a Mac, it will be too dark on a PC. If you are placing colored images or text on the Internet, you can't please all of the people all of the time. Currently, none of the graphic formats used on the Web can encode gamma correction information.

## System Colors

CSS2 enables you to specify colors by copying them from the user's native GUI. These system colors can be used with all color-related properties. Style rules based on system colors take into account user preferences, and therefore offer some advantages, including:

1. Pages that fit the user's preferred look and feel.
2. Pages that are potentially more accessible for users with settings that might be related to a disability.

Table 13-2 lists CSS2-system color keywords and their descriptions. Any of the color properties can take on these values.

Table 13-2 Additional System Colors to Be Used with All Color-related Properties	
<i>System Color-keywords</i>	<i>Description</i>
ActiveBorder	Active window border.
ActiveCaption	Active window caption.
AppWorkspace	Background color of multiple document interface.
Background	Desktop background.
ButtonFace	Face color for three-dimensional display elements.
ButtonHighlight	Dark shadow for three-dimensional display elements (for edges facing away from the light source).
ButtonShadow	Shadow color for three-dimensional display elements.
ButtonText	Text on push buttons.
CaptionText	Text in caption, size box, and scroll-bar arrow box.
GrayText	Grayed (disabled) text. This color is set to #000 if the current display driver does not support a solid gray color.
Highlight	Items selected in a control.
HighlightText	Text of items selected in a control.
InactiveBorder	Inactive window border.

*Continued*

Table 13-2 (continued)

<i>System Color-keywords</i>	<i>Description</i>
InactiveCaption	Inactive window caption.
InactiveCaptionText	Color of text in an inactive caption.
InfoBackground	Background color for tooltip controls.
InfoText	Text color for tooltip controls.
Menu	Menu background.
MenuText	Text in menus.
Scrollbar	Scroll bar gray area.
ThreeDDarkShadow	Dark shadow for three-dimensional display elements.
ThreeDFace	Face color for three-dimensional display elements.
ThreeDHighlight	Highlight color for three-dimensional display elements.
ThreeDLightShadow	Light color for three-dimensional display elements (for edges facing the light source).
ThreeDShadow	Dark shadow for three-dimensional display elements.
Window	Window background.
WindowFrame	Window frame.
WindowText	Text in windows.

For example, the following style rule sets the foreground and background colors of a VERSE to the same colors used for the foreground and background of the browser's window.

```
VERSE { color: WindowText; background-color: Window}
```

## Font Properties

Font properties in CSS1 are fairly complete. CSS2 doesn't add a lot to them. Changes include:

- ♦ The addition of the `font-size-adjust` property
- ♦ The scaling factor between the different keyword font sizes (`xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, `xx-large`) is **1.2**, not 1.5
- ♦ The `font-stretch` property can adjust the kerning

### font-size-adjust Property

The legibility of a font is generally less dependent upon the size of the font, than on the value of its *x*-height. The aspect value of a font is the font-size divided by the *x*-height. The higher this number, the more likely it is that a font will be legible when the font is a small size. The lower the aspect value, the more likely it is that the font will become illegible as it is shrunk. When browsers perform straightforward font substitutions that rely solely on the font size, the likelihood that the resulting font will be illegible is greatly increased. The `font-size-adjust` property controls the aspect value of elements that preserve the *x*-height of the first choice font in the substitute font when using the `font-family` property.

The Verdana and Times New Roman fonts provide a good example of this legibility issue. Verdana has an aspect value of .58, while Times New Roman has an aspect value of .46. Therefore, Verdana will remain legible at a smaller size than Times New Roman, but may appear too large if substituted directly for Times New Roman at the same font size.

If the value of the `font-size-adjust` property is none, the font's *x*-height is not preserved. If a number is specified, the value identifies the aspect value of the first-choice font, and directs the software to scale the substitution font accordingly. This system helps you force legibility across all platforms, and all supporting applications. The following rules use the `font-size-adjust` property to maintain legibility of fonts while implementing a range of sizes.

```
VERSE { font-size-adjust: ".58"; }
        font-family: "Verdana, Times New Roman,
                    Helvetica, Arial " ; }
AUTHOR { font-size-adjust: ".46" }
         font-family: "Times New Roman, Goudy Old Style,
                    serif, fantasy"; }
```

### font stretch Property

The `font-stretch` property controls the kerning of a font; that is, the amount of space found between two characters in the font. There are 12 legal keyword values for this property:

1. normal
2. ultra-condensed
3. extra-condensed
4. condensed
5. semi-condensed
6. semi-expanded

7. expanded
8. extra-expanded
9. ultra-expanded
10. wider
11. narrower
12. inherit

The default is `normal`. The values `ultra-condensed` through `ultra-expanded` are organized from most condensed to least condensed. Each is a small change in the horizontal spacing of the text. The values `wider` and `narrower` increase or decrease the kerning, without increasing or decreasing it more past the `ultra-expanded` or `ultra-condensed` level.

The following style sheet rules use a variety of kernings.

```
TITLE           { font-stretch: "ultra-expanded" }
AUTHOR          { font-stretch: "expanded" }
STANZA          { font-stretch: "ultra-condensed" }
VERSE           { font-stretch: "wider" }
REFRAIN VERSE   { font-stretch: "narrower" }
```

## The font Shorthand Property and System Fonts

In CSS1, the `font` property is a shorthand property that enables you to select font style, variant, weight, size, and family with one rule. In CSS2, the `font` property may also have one of these six keyword values that match all of a font's properties to the properties of particular elements of the browser user interface or the user's system:

1. `caption`: the font used for captioned controls like buttons
2. `icon`: the font that labels icons
3. `menu`: the font used in menus
4. `message-box`: the font used for display text in dialog boxes
5. `small-caption`: the font used for labels on small controls
6. `status-bar`: the font used in the browser status bar

For example this rule says that a `SONNET` element will be formatted with the same font family, size, weight, and style as the font the browser uses in its status bar:

```
SONNET { font: status-bar }
```

## Text Shadow Property

The `text-shadow` property applies shadows to text. The value is a comma-separated list of shadow effects to control the order, color, and dimensions of the shadows that are overlaid on the text. Shadows do not extend the size of the block containing the text, but may extend over the boundaries of the block. The stacking level of the shadows is the same as the element itself.

The value of the `text-shadow` includes a signed length for the offset of the shadow. It may also include a blur radius and a shadow color. The shadow offset is specified with two signed lengths that specify how far out from the text the shadow will extend. The first length specifies the horizontal distance from the text; the second length specifies vertical depth of the shadow. If you apply a negative value to the shadow offsets, the shadow will appear to the left and above the text, rather than below and to the right. An optional third signed length specifies the boundary of the blur effect. A fourth optional value specifies the color of the shadow. For example,

```
TITLE    { text-shadow: red -5pt -5pt -2pt }
AUTHOR   { text-shadow: 5pt 4pt 3pt green }
VERSE    { text-shadow: none }
```

## Vertical Align Property

The `vertical-align` property controls the vertical alignment of text within an inline box that is found within a block element. It's most commonly used with table cells. The eight possible alignment keyword values are:

1. **baseline**: aligns the baseline of the inline box with the baseline of the block box
2. **sub**: aligns the baseline of the inline box to the position for subscripts inside the parent block box
3. **super**: raises the baseline of the inline box to the position for superscripts in the parent's box
4. **top**: aligns the top of the inline box with the top of the line
5. **middle**: aligns the midpoint of the inline box with the baseline of the block box, plus half of the *x*-height of the block box
6. **bottom**: aligns the bottom of the inline box with the bottom of the line
7. **text-top**: aligns the top of the inline box with the top of the parent element's font
8. **text-bottom**: aligns the bottom of the inline box with the bottom of the parent element's font

You can also set the `vertical-align` property to a percentage that raises (positive value) or lowers (negative value) the box by the percentage of the line height. A value of 0% is the same as the `baseline` keyword. Finally, you can set `vertical-align` to a signed length that will raise or lower the box by the specified distance. A value of 0cm is the same as the `baseline` keyword.

## Boxes

When you are using CSS to format a document and its contents, you need to think in terms of boxes with borders and dimensions that hold the contents of an element. These boxes stack together and wrap around each other so that the contents of each element are aligned in an orderly fashion, based on the rules of the style sheets. CSS2 adds new outline properties for boxes, and enables boxes to be positioned in absolute positions on a page, in another box, or in a window.

## Outline Properties

CSS2 makes it possible to add outlines to objects. An outline is a lot like a border. However, an outline is drawn over the box. Its width does not add to the width of the box. Furthermore, if a CSS element is non-rectangular (unlikely), the outline around it will also be non-rectangular. Since outlines are not necessarily rectangular, you can not set the left, right, top, and bottom outline separately. You can only affect the entire outline at once.

### Outline Style Property

The `outline-style` property sets the style of the outline for the entire box. This functions just like the `border-style` property in CSS1, and has the same 11 possible values with the same meanings:

1. `none`: no line
2. `hidden`: an invisible line that still takes up space
3. `dotted`: a dotted line
4. `dashed`: a dashed line
5. `solid`: a solid line
6. `double`: a double solid line
7. `grooved`: a line that appears to be drawn into the page
8. `ridge`: a line that appears to be coming out of the page
9. `inset`: the entire object (not just the outline line) appears pushed into the document

10. `outset`: the entire object (not just the outline line) appears to be pushed out of the document
11. `inherit`: use the values of the parent

These three rules set the outline styles for the `TITLE`, `AUTHOR`, and `REFRAIN` elements:

```
TITLE    { outline-style: solid }
AUTHOR   { outline-style: outset }
REFRAIN  { outline-style: dashed }
```

### Outline Width Property

The `outline-width` property works like the `margin-width` and `border-width` properties discussed in Chapter 12. It sets the width of the outline of a box using either an unsigned length or one of these three keywords:

1. `thin`: about 0.5 to 0.75 points
2. `medium`: about 1 point
3. `thick`: about 1.5 to 2 points

For example, this rule outlines the `STANZA` with a thick outline and the `VERSE` with a thin one.

```
STANZA   { outline: thick }
VERSE    { outline: thin }
```

### Outline Color Property

The `outline-color` property sets the color of the outline of an element's box. Generally, this is set to either a color name like `red` or an RGB color like `#FF0000`. However, it may also have the keyword value `invert` which inverts the color of the pixels on the screen. (Black becomes white, and vice versa.) For example:

```
TITLE    { outline-color: #FFCCCC;
           outline-style: inset;
           outline-width: thick}
AUTHOR   { outline-color: #FF33CC}
VERSE    { outline-color: invert}
```

### Outline Shorthand Property

The `outline` property is a shorthand property that sets the outline width, color, and style for all four edges of a containing box. For example:

```
STANZA   { outline: thin dashed red }
VERSE    { outline: inset }
```

## Positioning Properties

CSS2 provides an astonishing amount of control over the position of each object in a document. You can put specific objects or specific types of objects in layers. Each layer can be moved independently of the other layers. The `position` property determines how objects are arranged and can have one of these four keyword values:

1. `static`: the default layout
2. `relative`: objects are offset from their static positions
3. `absolute`: objects are placed at a specific position relative to the box they're contained in
4. `fixed`: objects are placed at a specific point in the window or on the page

### Relative Positioning

As a document is laid out, the formatter chooses positions for items according to the normal flow of the objects and text. This is essentially the default static formatting of objects used by most document creators. After this has been completed, the objects may be shifted relative to their current position. This adjustment in an object's position is known as relative positioning. By using relative positioning, altering the position of an object has no effect on the objects following it. Thus boxes can overlap, since relatively positioned boxes retain all of their normal flow sizes and spacing.

You can generate a relatively positioned object by setting the `position` property to `relative`. Its offset will be controlled by the `left`, `right`, `top`, and `bottom` properties. By changing these properties with JavaScript you can even move objects and layers on your documents. You can make images or text move, appear and disappear, or change in mid-stream. For example, this rule moves the `TITLE` element 50 pixels up and 65 pixels to the left from where it would normally be.

```
TITLE { position: relative; top: 50px; left: 65px }
```

### Absolute Positioning

An absolutely positioned element is placed in reference to the block that contains it. It establishes a new containing block for boxes it contains. The contents of absolutely positioned elements do not flow around other boxes. This may cause them to obscure the contents of other boxes displayed in the document. Absolutely positioned elements have no impact on the flow of their following siblings, so elements that follow an absolutely positioned one, act as if it were not there. For example, this rule puts the top left corner of the `AUTHOR` element 60 pixels down and 140 pixels to the right of the top left corner of the box it's contained in.

```
AUTHOR { position: absolute; top: 60px; left: 140px }
```

## Fixed Positioning

Elements with fixed position are placed at coordinates relative to the window or page on which they're displayed. If you are viewing a document composed of continuous media, the fixed box will not move when the document is scrolled. If the fixed box is located on paged media, it will always appear at the end of each page. This enables you to place a footer or header on a document, or a signature at the end of a series of one-page letters. For example, this rule puts the top-left corner of the REFRAIN element 300 pixels down and 140 pixels to the right of the top-left corner of the window it's displayed in or the page it's printed on.

```
REFRAIN { position: fixed; top: 300px; left: 140px}
```

## Stacking Elements with the Z-Index Property

The `z-index` property controls the stacking order of positioned boxes. To change the default `z-index` value, you set `z-index` to an integer like 2. Objects with larger `z-index` values are placed on top of objects with smaller `z-index` values. Whether the objects on the bottom show through is a function of the background properties of the object on top of them. If the backgrounds are transparent, at least some of what's below will probably show through.

Listing 13-2 is a style sheet that uses absolute positioning with a `z-index` to create a multi-part overlay of the Shakespearean sonnet. The result is shown in Figure 13-3. It's certainly not as nice as the version that merely allows the browser to lay out the sonnet. Absolute positioning should be used with extreme care. I'd really only recommend it for print media where you'll be distributing the paper that comes out of your printer rather than the electronic files.

### Listing 13-2: Shakespeare's sonnet with a `z-index` stylesheet

```
#st1 { position: absolute;
      top: 160px;
      left: 200px;
      height: 100px;
      width: 200px;
      overflow: auto;
      z-index: 2}
#st2 { position: absolute;
      top: 210px;
      left: 50px;
      height: 100px;
      width: 200px;
      overflow: auto;
      z-index: 3}
#st3 { position: absolute;
      top: 210px;
      left: 250px;
```

*Continued*

## Listing 13-2 (continued)

```

        height: 100px;
        width:200px;
        overflow: auto;
        z-index: 4}
REFRAIN { position: absolute;
        top: 300px;
        left:200px;
        height: 100px;
        width:200px;
        overflow: auto;
        z-index: 5}

```

Using absolute positioning ordered by z-index, you can control the stacking order of text boxes.

## Counters and Automatic Numbering

CSS2 enables you to automatically generate some content. For instance, you can use the style sheet to create outlines that are properly indented with different numbering systems for each level of the outline.

The `counter-increment` property adds one to a counter. The `content` property inserts the current value of a named counter by using either the `counter(id)` or `counter(id, list-style-type)` functions as values. Finally, the `counter-reset` property sets a counter back to 0.

For example, let's suppose you want to number each VERSE in a poem starting from one, but reset the counting in each new STANZA. and the REFRAIN. You can do that with the following rules:

```

VERSE          {counter-increment: verse-num}
STANZA         {counter-reset: verse-num}
REFRAIN       {counter-reset: verse-num}
VERSE:before  {content: counter(verse-num) }

```

You can reset back to a number other than 0 by specifying the integer to reset to after the counter name in `counter-reset`. For example, to reset the counter to -10:

```

VERSE          {counter-reset: verse-num -10}

```

You can also increment by an integer different than 1 by specifying it in `counter-increment` after the counter name. For example,

```
VERSE          {counter-increment: verse-num -1}
```

Finally, the `content` property can have more than one counter, and additional content as well as counters. For instance, these rules number the verses in the form 1.1, 1.2, 1.3, ..., 2.1, 2.2, 2.3, ... where the first number indicates the stanza and the second the verse:

```
VERSE          {counter-increment: verse-num}
STANZA         {counter-reset: verse-num}
STANZA         {counter-increment: stanza-num}
REFRAIN        {counter-reset: verse-num}
REFRAIN        {counter-reset: stanza-num 0}
VERSE:before   {content:
                  counter(stanza-num) "." counter(verse-num) }
```

You're not limited to European numerals either. You can pass a second argument to the `counter()` function to specify a different number format. Available formats include `disc`, `circle`, `square`, `decimal`, `decimal-leading-zero`, `lower-roman`, `upper-roman`, `lower-greek`, `lower-alpha`, `lower-latin`, `upper-alpha`, `upper-latin`, `hebrew`, `armenian`, `georgian`, `CJK-ideographic`, `hiragana`, `katakana`, `hiragana-iroha`, and `katakana-iroha`. For example, to number the verses using Japanese numeral in hiragana, you might write:

```
VERSE:before {content: counter(stanza-num, hiragana)
                      "." counter(verse-num, hiragana) }
```

## Aural Style Sheets

Visually impaired users already use special software to read Web pages. In the future, such use is likely to expand to sighted people browsing the Web while talking on cell phones, driving their cars, washing the dishes, and performing other activities in which the eyes and hands have to be directed elsewhere. CSS2 supports new properties to describe how elements are read out loud as well as how they're printed or shown on a screen. The new properties are discussed in the sections that follow. Listing 13-3 is an aural style sheet that identifies specific ways to speak information found in common play-related XML elements.

### Listing 13-3: An Aural style sheet for a play or sonnet

```
TITLE, AUTHOR, ACT, SCENE {
    voice-family: narrator;
    stress: 20;
```

*Continued*

## Listing 13-3 (continued)

```

        richness: 90;
        cue-before: url("ping.au")
    }

.narrator { pause: 20ms;
            cue-before: url("pop.au");
            cue-after: url("pop.au");
            azimuth: 30deg;
            elevation: above }

ACT    { pause: 30ms 40ms } /* pause-before: 30ms;
                           pause-after: 40ms */

SCENE { pause-after: 10ms } /* pause-after: 10ms */

SCENE { cue-before: url("bell.aiff");
        cue-after: url("dong.wav") }

MOOD.sad      { play-during: url("violins.aiff") }
MOOD.funereal { play-during: url("harp.wav") mix }
MOOD.quiet    { play-during: none }

LINE.narrator    { azimuth: behind } /* 180deg */
LINE.part.romeo  { voice-family: romeo, male }
LINE.part.juliet { voice-family: juliet, female }
LINE.part.hercules { azimuth: center-left }
LINE.part.richard { azimuth: right }
LINE.part.carmen { volume: x-soft }
LINE.part.muse1  { elevation: 60deg }
LINE.part.muse2  { elevation: 30deg }
LINE.part.muse3  { elevation: level }

```

## Speak Property

The `speak` property determines whether text will be rendered aurally and if so, how. If `speak` has the value `normal`, text is spoken using the best available speech synthesis. If `speak` has the value `spell-out`, the text is spelled out letter-by-letter, which might be useful for unusual or foreign words a speech synthesizer probably can't handle. The default value is `none` (for example, just render the content visually and forget about speech synthesis).

## Volume Property

The `volume` property controls the average volume of the speaking voice of the speech synthesizer. This is the median value of the analog wave of the voice, but

it's only an average. A highly inflected voice at a volume of 50 might peak at 75. The minimum volume is 0. The maximum volume is 100. Percentage values can also be used, as can any of these six keywords:

1. `silent`: no sound
2. `x-soft`: 0, the minimum audible volume
3. `soft`: about 25
4. `medium`: about 50
5. `loud`: about 75
6. `x-loud`: 100, the maximum comfortable hearing level

## Pause Properties

Pauses are the aural equivalent of a comma. They can be used to provide drama, or just to help separate one speaker's voice from another's. They're set in CSS2 with the `pause`, `pause-before`, and `pause-after` properties.

The `pause-before` property specifies the length of time the speech synthesizer should pause before speaking an element's contents. The `pause-after` property specifies the length of time the speech synthesizer should pause after speaking an element's contents. These can be set as an absolute time or as a percentage of the `speech-rate` property. The `pause` property is shorthand for setting both `pause-before` and `pause-after`. When two values are supplied, the first is applied to `pause-before` and the second is applied to `pause-after`. When only one value is given, it applies to both properties. For example:

```
SCENE { pause-after: 10ms }

/* pause-before: 20ms; pause-after: 20ms */
.narrator { pause: 20ms }

/* pause-before: 30ms; pause-after: 40ms */
ACT { pause: 30ms 40ms }
```

## Cue Properties

Cues are audible clues that alert the listener to a specific event that is about to occur, or has just occurred. Each cue property specifies a URL for a sound file that will be played before or after an element is spoken. The `cue-before` property plays a sound before an element is read. The `cue-after` property plays a sound after an element is read.

The `cue` property is shorthand for setting both `cue-before` and `cue-after`. When two values are supplied, the first is applied to `cue-before` and the second is applied to `cue-after`. When only one value is given, it applies to both properties. For example:

```
ACT, SCENE { cue-before: url("ping.au") }
.narrator  { cue: url("pop.au") }
SCENE     { cue-before: url("bell.aiff");
           cue-after: url("dong.wav") }
```

## Play-During Property

The `play-during` property specifies a sound to be played in the background while an element's content is spoken. The value of the property is URL to the sound file. You can also add one or both of the keywords `mix` and `repeat` to the value. `Mix` tells the speech synthesizer to mix in the parent's `play-during` sound. The `repeat` value tells the speech synthesizer to loop the sound continuously until the entire element has been spoken. The default value is `none`.

## Spatial Properties

The spatial properties specify where the sound should appear to be coming from. For example, you can have a document read to you from 3 feet away in a ditch or 100 feet away on a cliff. This is of course limited by the capabilities of the speech synthesizer and audio hardware. Since you can not predetermine the number and location of speakers in use by the document reader, these properties simply identify the desired end result. As the document author, you can't really force the sound to appear to be coming from any particular direction, anymore than you can guarantee that a reader has a color monitor.

### Azimuth Property

The `azimuth` property controls the horizontal angle from which the sound appears to come. When you listen to audio through good stereo speakers, you seem to hear a lateral sound stage. The `azimuth` property can be used with this type of stereo system to create angles to the sound you hear. When you add a total surround-sound system using either a binaural headphone or a 5-speaker home theatre setup, the `azimuth` property becomes very noticeable.

The `azimuth` is specified as an angle between  $-360^\circ$  and  $360^\circ$ . A value of `0deg` means that the sound is directly in front of the listener (as are `-360deg` and `360deg`). A value of `180deg` means that the sound is directly behind the listener. (In CSS terminology `deg` replaces the more common  $^\circ$  degree symbol.) Angles are counted clockwise to the listener's right. You can also use one of these nine keywords to specify the azimuthal angle:

1. center: **0deg**
2. center-right: **20deg**
3. right: **40deg**
4. far-right: **60deg**
5. right-side: **90deg**
6. left-side: **270deg**
7. far-left: **300deg**
8. left: **320deg**
9. center-left: **340deg**

You can add the keyword `behind` behind to any of these values to set the position to 180deg minus the normal value. For example `left behind` is the same as `180deg - 320deg = -140deg` or `220deg`.

A value of `leftwards` moves the sound an additional 20 degrees to the left, relative to the current angle. This is most easily understood as turning the sound counter-clockwise. So even if the sound is already behind the listener, it will continue to move “left” around the circle. A value of `rightwards` moves the sound an additional 20 degrees to the right (clockwise) from to the current angle.

## Elevation Property

The `elevation` property controls the apparent height of the speaker above the listener’s position. The elevation is specified as an angle between -90° and 90°. It can also be given as one of these five keywords:

1. below **-90deg**
2. level **0deg**
3. above **90deg**
4. higher **10deg** above the current elevation (useful with inheritance)
5. lower **10deg** below the current elevation (useful with inheritance)

## Voice Characteristics Properties

The individual characteristics of the synthesizer’s voice can be controlled by adjusting the rate of speech, the voice-family used, the pitch, and the richness of the voice.

### Speech Rate Property

The `speech-rate` property specifies the speaking rate of the speech synthesizer as an approximate number of average sized words per minute. You can supply an integer or one of these five keywords:

1. `x-slow`: 80 words per minutes
2. `slow`: 120 words per minute
3. `medium`: 180 to 200 words per minute
4. `fast`: 300 words per minute
5. `x-fast`: 500 words per minute

You can also use the keyword `faster` to add 40 words per minute to the rate of the parent element or `slower` to subtract 40 words per minute from the rate of the parent element.

### Voice Family Property

The `voice-family` property is a comma-separated, prioritized list of voice-family names that chooses the voice used for reading the text of the document. It's like the `font-family` property discussed in Chapter 12, but is regarding voices instead of typefaces.

Generic voice values include `male`, `female`, and `child`. Specific names are as diverse as font names and include `Agnes`, `Bruce`, `Good News`, `Hysterical`, `Victoria`, `Whisper`, and many more. These names must be quoted if they do not conform to syntax rules for identifiers, or if they consist of more than one word. For example:

```
LINE.part.romeo { voice-family: Bruce, "Good News", male }
```

### Pitch Property

The `pitch` property specifies the frequency the speech synthesizer uses for a particular type of object. To some degree this controls whether a voice sounds male or female. However, it's better to use an appropriate voice-family instead. The value is given in hertz (cycles per second). Female voices are about 120Hz, while typical male voices are in the ballpark of 200Hz. You can also use these keywords to adjust the pitch:

1. `x-low`
2. `low`
3. `medium`
4. `high`
5. `x-high`

The exact frequencies of these keywords depend on the user's environment and selected voice. However, `x-low` is always lower than `low`, which is always lower than `medium`, and so forth.

### Pitch Range Property

The `pitch-range` property specifies the acceptable variations in the speaker's average pitch as a number between 0 and 100. This controls the inflection and variation of the voice used by the speech synthesizer. A value of 0 creates a flat, monotone voice, while 50 is a normal voice, and values above 50 create an exceptionally animated voice.

### Stress Property

The `stress` property specifies the level of assertiveness or emphasis that's used in the speaking voice. The default is 50. The value and effect of this attribute has a different effect in each language being spoken. When used with languages such as English that stress sentence position, you can select primary, secondary, and tertiary stress points to control the inflection that is applied to these areas of the sentence.

### Richness Property

The `richness` property specifies the “brightness” of the voice used by the speech synthesizer. The richer the voice, the better its carrying capacity. Smooth voices don't carry far because their wave forms are not as deeply pitched as rich voices. The value is a number between 1 and 100, with a default of 50. Higher values produce voices that carry better, while lower values produce softer voices that are easier to listen to.

## Speech Properties

These properties control how the speech synthesizer interprets punctuation and numbers. There are two such properties: `speak-punctuation` property and the `speak-numeral` property.

### Speak Punctuation Property

By default punctuation is spoken literally. A statement such as “The cat, Charm, ate all of his food.” is read as “The cat comma Charm comma ate all of his food period”. However, by setting the `speak-punctuation` property to `none`, none of the punctuation will be spoken. It will, however, have pauses, as would a natural speaking voice. For example, “The cat <pause> Charm <pause> ate all of his food <silence>”.

### Speak Numeral Property

By default numbers are spoken as a full string. For example, the number 102 would be read “one hundred and two”. If, however, you set the `speak-numeral` property

to digits, each number being will be spoken individually like “one zero two”. You can return to the default by setting `speak-numeral` property to `continuous`. If `speak-numeral` is set to `none`, numbers not will be spoken.

## Summary

This chapter covered CSS2’s features and how to implement them. In this chapter, you learned:

- ♦ CSS2 is mostly a superset of CSS1, though there are a few differences including a default display type of `inline` instead of `block`.
- ♦ Internet Explorer 5 and Mozilla 5 have only marginally implemented CSS2, so don’t expect a lot of its features to work flawlessly.
- ♦ CSS2 has expanded the various selectors that can apply specific properties to particular elements including a universal selector, child selectors, descendant selectors, and sibling selectors.
- ♦ New `@rules` have been developed to give document authors more control over their printed documents, including `@charset`, `@page`, and `@font-face`.
- ♦ CSS2 has seven new pseudo-classes, including `:first-child` and `:hover`, to select elements that have something in common, but do not necessarily have the same type.
- ♦ CSS2 has two new pseudo-elements that let you insert content into the document: `:after` and `:before`.
- ♦ CSS2 has increased the use of the `display` property, by incorporating values to display elements as all the parts of a table, not at all (`none`), and as `compact` or `run-in` objects.
- ♦ System colors and systems fonts enable you to create an interface on your XML applications that more closely matches the main system settings on each individual visitors computers.
- ♦ CSS2 adds aural properties for describing speech, volume, pausing, cues, voice characteristics, and the specification of a sound to be played and where it should be coming from, among other things.

As with CSS1, CSS2 still has many limitations, the most obvious of which is lack of full support from Web browsers, but this should change with time. XSL is still by far the most full-bodied style sheet language for use with XML documents. In the next chapter, you will explore XSL transformations, and see how much farther they can take you.

