

## Text Conventions

Throughout the book, we use a `constant-width` typeface to highlight any literal element of the HTML/XHTML standards, tags, and attributes. We always use lowercase letters for tags.<sup>[1]</sup> We use *italic* for filenames and URLs and to indicate new concepts when they are defined. Elements you need to supply when creating your own documents, such as tag attributes or user-defined strings, appear in `constant-width italic` in the code.

<sup>[1]</sup> HTML is case-insensitive with regard to tag and attribute names, but XHTML is case-sensitive. And some HTML items, such as source filenames, are case-sensitive, so be careful.

We discuss elements of the language throughout the book, but you'll find each one covered in depth (some might say in nauseating detail) in a shorthand, quick-reference definition box that looks like the one that follows. The first line of the box contains the element name, followed by a brief description of its function. Next, we list the various attributes, if any, of the element: those things that you may or must specify as part of the element.

## **<title>**

### *Function*

Defines the document title

### *Attributes*

`dirlang`

### *End tag*

`</title>;` never omitted

### *Contains*

*plain\_text*

### *Used in*

*head\_content*

We use the following symbols to identify tags and attributes that are not in the HTML 4.01 or XHTML 1.0 standards but are additions to the languages:

- ☐ Netscape extension to the standards
- ☐ Internet Explorer extension to the standards

The description also includes the ending tag, if any, for the element, along with a general indication of whether the end tag may be safely omitted in general use in HTML. For the few tags that require end tags in XHTML but do not have them in HTML, the language lets you indicate that by placing a forward slash (/) before the tag's closing bracket, as in `<br />`. In these cases, the tag may also contain attributes, indicated with an intervening ellipsis, such as `<br ... />`.

The "Contains" header names the rule in the HTML grammar that defines

the elements to be placed within this tag. Similarly, the "Used in" header lists those rules that allow this tag as part of their content. These rules are defined in [Appendix A](#).

Finally, HTML and XHTML are fairly intertwined languages. You will occasionally use elements in different ways depending on context, and many elements share identical attributes. Wherever possible, we place a cross-reference in the text that leads you to a related discussion elsewhere in the book. These cross-references, like the one at the end of this paragraph, serve as a crude paper model of hypertext documentation, one that would be replaced with a true hypertext link should this book be delivered in an electronic format. [[Section 3.3.1](#)]

We encourage you to follow these references whenever possible. Often, we cover an attribute briefly and expect you to jump to the cross-reference for a more detailed discussion. In other cases, following the link takes you to alternative uses of the element under discussion or to style and usage suggestions that relate to the current element.

## Appendix A. HTML Grammar

For the most part, the exact syntax of an HTML or XHTML document is not rigidly enforced by a browser. This gives authors wide latitude in creating documents and gives rise to documents that work on most browsers but actually are incompatible with the HTML and XHTML standards. Stick to the standards, unless your documents are fly-by-night affairs.

The standards explicitly define the ordering and nesting of tags and document elements. This syntax is embedded within the appropriate Document Type Definition and is not readily understood by those not versed in SGML (for the HTML 4.01 DTD, see [Appendix D](#)) or XML (for the XHTML 1.0 DTD, see [Appendix E](#)). Accordingly, we provide an alternate definition of the allowable HTML and XHTML syntax, using a fairly common tool called a "grammar."

Grammar, whether it defines English sentences or HTML documents, is just a set of rules that indicates the order of language elements. These language elements can be divided into two sets: *terminal* (the actual words of the language) and *nonterminal* (all other grammatical rules). In HTML and XHTML, the words correspond to the embedded markup tags and text in a document.

To use the grammar to create a valid document, follow the order of the rules to see where the tags and text may be placed to create a valid document.

## 3.3 Tags and Attributes

For the most part, tags—the markup elements of HTML and XHTML—are simple to understand and use, since they are made up of common words, abbreviations, and notations. For instance, the `<i>` and `</i>` tags respectively tell the browser to start and stop italicizing the text characters that come between them. Accordingly, the syllable "simp" in our barebones example above would appear italicized on a browser display.

The HTML and XHTML standards and their various extensions define how and where you place tags within a document. Let's take a closer look at that syntactic sugar that holds together all documents.

### 3.3.1 The Syntax of a Tag

Every tag consists of a tag *name*, sometimes followed by an optional list of tag *attributes*, all placed between opening and closing brackets (`<` and `>`). The simplest tag is nothing more than a name appropriately enclosed in brackets, such as `<head>` and `<i>`. More complicated tags contain one or more *attributes*, which specify or modify the behavior of the tag.

According to the HTML standard, tag and attribute names are not case-sensitive. There's no difference in effect between `<head>`, `<Head>`, `<HEAD>`, or even `<HeaD>`; they are all equivalent. With XHTML, case *is* important: all current standard tag and attribute names are in lowercase.

For both HTML and XHTML, the values that you assign to a particular attribute may be case-sensitive, depending on your browser and server. In particular, file location and name references—or uniform resource locators (URLs)—are case-sensitive. [[Section 6.2](#)]

Tag attributes, if any, belong after the tag name, each separated by one or more tab, space, or return characters. Their order of appearance is not important.

A tag attribute's value, if any, follows an equals sign (=) after the attribute name. You may include spaces around the equals sign, so that `width=6`, `width = 6`, `width =6`, and `width= 6` all mean the same. For readability, however, we prefer not to include spaces. That way, it's easier to pick out an attribute/value pair from a crowd of pairs in a lengthy tag.

With HTML, if an attribute's value is a single word or number (no spaces), you may simply add it after the equals sign. All other values should be enclosed in single or double quotation marks, especially those values that contain several words separated by spaces. With XHTML, all attribute values must be enclosed in double quotes. The length of the value is limited to 1,024 characters.

Most browsers are tolerant of how tags are punctuated and broken across lines. Nonetheless, avoid breaking tags across lines in your source document whenever possible. This rule promotes readability and reduces potential errors in your HTML documents.

### 3.3.2 Sample Tags

Here are some tags with attributes:

```
<a href="http://www.oreilly.com/catalog.html">
```

```
<ul compact>
```

```
<ul compact="compact">
```

```
<input type=text name=filename size=24 maxlength=80>
```

```
<link title="Table of Contents">
```

The first example is the `<a>` tag for a hyperlink to O'Reilly & Associates's web-based catalog of products. It has a single attribute, `href`, followed by the catalog's address in cyberspace its URL.

The second example shows an HTML tag that formats text into an

unordered list of items. Its single attribute `compact`, which limits the space between list items does not require a value.

The third example demonstrates how the second example must be written in XHTML. Notice the `compact` attribute now has a value, albeit a redundant one, and that its value is enclosed in double quotes.

The fourth example shows an HTML tag with multiple attributes, each with a value that does not require enclosing quotation marks. Of course, with XHTML, each attribute value must be enclosed in double quotes.

The last example shows proper use of enclosing quotation marks when the attribute value is more than one word long.

What is not immediately evident in these examples is that while HTML attribute names are not case-sensitive (`href` works the same as `HREF` and `HreF` in HTML), most attribute values are case-sensitive. The value `filename` for the `name` attribute in the `<input>` tag example is not the same as the value `Filename`, for instance.

### 3.3.3 Starting and Ending Tags

We alluded earlier to the fact that most tags have a beginning and an end and affect the portion of content between them. That enclosed segment may be large or small, from a single text character, syllable, or word such as the italicized "simp" syllable in our barebones example to the `<html>` tag that bounds the entire document. The starting component of any tag is the tag name and its attributes, if any. The corresponding ending tag is the tag name alone, preceded by a slash (/). Ending tags have no attributes.

### 3.3.4 Proper and Improper Nesting

Tags can be put inside the affected segment of another tag (nested) for multiple tag effects on a single segment of the document. For example, a portion of the following text is both bold and included as part of an anchor

defined by the `<a>` tag:

```
<body>
```

```
This is some text in the body, with a
```

```
<a href="another_doc.html">link, a portion of which  
is <b>set in bold</b></a>
```

```
</body>
```

According to the HTML and XHTML standards, you must end nested tags by starting with the most recent one and working your way back out first in, last out. For instance, in this example, we end the bold tag (`</b>`) before ending the link tag (`</a>`), since we started in the reverse order: `<a>` tag first, then `<b>` tag. It's a good idea to follow that standard, even though most browsers don't absolutely insist you do so. You may get away with violating this nesting rule for one browser, and sometimes even with all current browsers. But eventually a new browser version won't allow the violation, and you'll be hard pressed to straighten out your source HTML document. Also, be aware that the XHTML standard explicitly forbids improper nesting.

### 3.3.5 Tags Without Ends

According to the HTML standard, a few tags do not have ending tags. In fact, the standard forbids use of an end tag for these special ones, although most browsers are lenient and ignore the errant end tag. For example, the `<br>` tag causes a line break; it has no effect otherwise on the subsequent portion of the document and, hence, does not need an ending tag.

The HTML tags that do not have corresponding end tags are:

<code>&lt;area&gt;</code>	<code>&lt;base&gt;</code>	<code>&lt;basefont&gt;</code>
---------------------------	---------------------------	-------------------------------



<code>&lt;br&gt;</code>	<code>&lt;col&gt;</code>	<code>&lt;frame&gt;</code>
<code>&lt;hr&gt;</code>	<code>&lt;img&gt;</code>	<code>&lt;input&gt;</code>
<code>&lt;isindex&gt;</code>	<code>&lt;link&gt;</code>	<code>&lt;meta&gt;</code>
<code>&lt;param&gt;</code>		

XHTML always requires end tags. [[Section 16.3.3](#)]

### 3.3.6 Omitting Tags

You often see documents in which the author seemingly has forgotten to include an ending tag, in apparent violation of the HTML standard. Sometimes you even see a missing `<body>` tag. But your browser doesn't complain, and the document displays just fine. What gives? The HTML standard lets you omit certain tags or their endings for clarity and ease of preparation. The HTML standard writers didn't intend the language to be tedious.

For example, the `<p>` tag that defines the start of a paragraph has a corresponding end tag, `</p>`, but the end tag rarely is used. In fact, many HTML authors don't even know it exists. [[Section 4.1.2](#)]

The HTML standard lets you omit a starting tag or ending tag whenever it can be unambiguously inferred by the surrounding context. Many browsers make good guesses when confronted with missing tags, leading the document author to assume that a valid omission was made.

We recommend that you almost always add the ending tag. It'll make life easier for yourself as you transition to XHTML as well as for the browser and anyone who might need to modify your document in the future.

### 3.3.7 Ignored or Redundant Tags

HTML browsers sometimes ignore tags. This usually happens with redundant tags whose effects merely cancel or substitute for themselves. The best example is a series of `<p>` tags, one after the other, with no intervening content. Unlike how the similar series of repeating return characters is handled by a text-processing tool, most browsers skip to a new line only once. The extra `<p>` tags are redundant and usually ignored by the browser.

In addition, most HTML browsers ignore any tag that they don't understand or that was incorrectly specified by the document author. Browsers habitually forge ahead and make some sense of a document, no matter how badly formed and error-ridden it may be. This isn't just a tactic to overcome errors; it's also an important strategy for extensibility. Imagine how much harder it would be to add new features to the language if the existing base of browsers choked on them.

The thing to watch out for with nonstandard tags that aren't supported by most browsers is their enclosed contents, if any. Browsers that recognize the new tag may process those contents differently than those that don't support the new tag. For example, Internet Explorer and Netscape Navigator now both support the `<style>` tag, whose contents serve to set the various display characteristics of your document. However, previous versions of the popular browsers, many of which are still in use by many people today, don't support styles. Hence, older browsers ignore the `<style>` tag and render its contents on the user's screen, effectively defeating the tag's purpose in addition to ruining the document's appearance. [[Section 8.1.2](#)]

## Appendix D. The HTML 4.01 DTD

The HTML 4.01 standard is formally defined as three SGML Document Type Definitions (DTDs): the Strict DTD, the Transitional DTD, and the Frameset DTD. The Strict DTD defines only those elements that are not deprecated in the 4.0 standard. Ideally, everyone would create HTML documents that conform to the Strict DTD. The Transitional DTD includes all those deprecated elements and more accurately reflects the HTML in use today, with many older elements still in common use. The Frameset DTD is identical to the Transitional DTD, with the exception that the document `<body>` is replaced by the `<frameset>` tag.

Since the Transitional DTD provides the broadest coverage of all HTML elements currently in use, it is the DTD upon which this book is based and the one we reproduce here. Note that we have reprinted this DTD verbatim and have not attempted to add extensions to it. Where our description and the DTD deviate, assume the DTD is correct.

`<!--`

```
This is the HTML 4.01 Transitional DTD, which includes
presentation attributes and elements that W3C expects
as support for style sheets matures. Authors should
use the Strict DTD when possible, but may use the Transitional DTD
for presentation attributes and elements is required.
HTML 4 includes mechanisms for style sheets, script,
embedding objects, improved support for right to left
direction text, and enhancements to forms for improved
```

accessibility for people with disabilities.

Draft: \$Date: 2002/10/07 21:48:24 \$

Authors:

Dave Raggett <dsr@w3.org>

Arnaud Le Hors <lehors@w3.org>

Ian Jacobs <ij@w3.org>

Further information about HTML 4.01 is available at:

<http://www.w3.org/TR/1999/REC-html401-19991224>

The HTML 4.01 specification includes additional syntactic constraints that cannot be expressed with the DTDs.

-->

<!ENTITY % HTML.Version "-//W3C//DTD HTML 4.01 Transitional

-- Typical usage:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional
    "http://www.w3.org/TR/html4/loose.dtd">
```

```
<html>
```

```
<head>
```

...

```
</head>
```

```
<body>
```

```
...
```

```
</body>
```

```
</html>
```

The URI used as a system identifier with the public

the user agent to download the DTD and entity sets

The FPI for the Strict HTML 4.01 DTD is:

```
"-//W3C//DTD HTML 4.01//EN"
```

This version of the strict DTD is:

```
http://www.w3.org/TR/1999/REC-html401-19991224
```

Authors should use the Strict DTD unless they need

presentation control for user agents that don't (a

support style sheets.

If you are writing a document that includes frames

the following FPI:

```
"-//W3C//DTD HTML 4.01 Frameset//EN"
```

This version of the frameset DTD is:

<http://www.w3.org/TR/1999/REC-html401-19991224>

Use the following (relative) URIs to refer to  
the DTDs and entity definitions of this specificat.

"strict.dtd"

"loose.dtd"

"frameset.dtd"

"HTMLlat1.ent"

"HTMLsymbol.ent"

"HTMLspecial.ent"

-->

<!--===== Imported Names =====>

<!-- Feature Switch for frameset documents -->

<!ENTITY % HTML.Frameset "IGNORE">

<!ENTITY % ContentType "CDATA"

-- media type, as per [RFC2045]

-->

<!ENTITY % ContentTypes "CDATA"

-- comma-separated list of media types, as per [RFC

-->

```
<!ENTITY % Charset "CDATA"
```

```
    -- a character encoding, as per [RFC2045]
```

```
-->
```

```
<!ENTITY % Charsets "CDATA"
```

```
    -- a space-separated list of character encodings, ,
```

```
-->
```

```
<!ENTITY % LanguageCode "NAME"
```

```
    -- a language code, as per [RFC1766]
```

```
-->
```

```
<!ENTITY % Character "CDATA"
```

```
    -- a single character from [ISO10646]
```

```
-->
```

```
<!ENTITY % LinkTypes "CDATA"
```

```
    -- space-separated list of link types
```

```
-->
```

```
<!ENTITY % MediaDesc "CDATA"
```

```
    -- single or comma-separated list of media descrip
```

```
-->
```

```

<!ENTITY % URI "CDATA"

    -- a Uniform Resource Identifier,

    see [URI]

-->

<!ENTITY % Datetime "CDATA" -- date and time informati
<!ENTITY % Script "CDATA" -- script expression -->
<!ENTITY % StyleSheet "CDATA" -- style sheet data -->
<!ENTITY % FrameTarget "CDATA" -- render in this frame
<!ENTITY % Text "CDATA">

<!-- Parameter Entities -->

<!ENTITY % head.misc "SCRIPT|STYLE|META|LINK|OBJECT" -
<!ENTITY % heading "H1|H2|H3|H4|H5|H6">
<!ENTITY % list "UL | OL | DIR | MENU">
<!ENTITY % preformatted "PRE">

<!ENTITY % Color "CDATA" -- a color using sRGB: #RRGGBB
<!-- There are also 16 widely known color names with t

    Black   = #000000      Green   = #008000

    Silver  = #C0C0C0      Lime    = #00FF00

    Gray    = #808080      Olive   = #808000

```



```

        White   = #FFFFFF      Yellow = #FFFF00

        Maroon  = #800000      Navy    = #000080

        Red     = #FF0000      Blue     = #0000FF

        Purple  = #800080      Teal     = #008080

        Fuchsia= #FF00FF      Aqua     = #00FFFF

-->

<!ENTITY % bodycolors "

    bgcolor      %Color;      #IMPLIED  -- document backgroun
    text         %Color;      #IMPLIED  -- document text color
    link         %Color;      #IMPLIED  -- color of link
    vlink        %Color;      #IMPLIED  -- color of visited link
    alink        %Color;      #IMPLIED  -- color of selected link

">

<!--===== Character mnemonic entities =====>

<!ENTITY % HTMLlat1 PUBLIC

    "-//W3C//ENTITIES Latin1//EN//HTML"

    "HTMLlat1.ent">

%HTMLlat1;

```

```

<!ENTITY % HTMLsymbol PUBLIC

    "-//W3C//ENTITIES Symbols//EN//HTML"

    "HTMLsymbol.ent">

%HTMLsymbol;

<!ENTITY % HTMLspecial PUBLIC

    "-//W3C//ENTITIES Special//EN//HTML"

    "HTMLspecial.ent">

%HTMLspecial;

<!--===== Generic Attributes =====>

<!ENTITY % coreattrs

    "id            ID            #IMPLIED  -- document-wide
    class         CDATA         #IMPLIED  -- space-separated
    style         %StyleSheet;   #IMPLIED  -- associated s
    title         %Text;         #IMPLIED  -- advisory tit
    >

<!ENTITY % i18n

    "lang         %LanguageCode; #IMPLIED  -- language cod
    dir           (ltr|rtl)      #IMPLIED  -- direction fo
    >

```

```
<!ENTITY % events
```

```
"onclick      %Script;          #IMPLIED  -- a pointer button was clicked
ondblclick    %Script;          #IMPLIED  -- a pointer button was double-clicked
onmousedown   %Script;          #IMPLIED  -- a pointer button was pushed down
onmouseup     %Script;          #IMPLIED  -- a pointer button was released
onmouseover   %Script;          #IMPLIED  -- a pointer was moved over an element
onmousemove   %Script;          #IMPLIED  -- a pointer was moved across an element
onmouseout    %Script;          #IMPLIED  -- a pointer was moved away from an element
onkeypress    %Script;          #IMPLIED  -- a key was pressed
onkeydown     %Script;          #IMPLIED  -- a key was pressed down
onkeyup       %Script;          #IMPLIED  -- a key was released

>
```

```
<!-- Reserved Feature Switch -->
```

```
<!ENTITY % HTML.Reserved "IGNORE">
```

```
<!-- The following attributes are reserved for possible future use -->
```

```
<![ %HTML.Reserved; [
```

```
<!ENTITY % reserved
```

```
"datasrc      %URI;              #IMPLIED  -- a single or multiple data sources
```

```

datafld      CDATA      #IMPLIED  -- the property
dataformatas (plaintext|html) plaintext -- text or h
>
]]>

<!ENTITY % reserved "">

<!ENTITY % attrs "%coreattrs; %i18n; %events;">

<!ENTITY % align "align (left|center|right|justify) #
                -- default is left for ltr paragrap
>

<!--===== Text Markup =====>

<!ENTITY % fontstyle
"TT | I | B | U | S | STRIKE | BIG | SMALL">

<!ENTITY % phrase "EM | STRONG | DFN | CODE |
                SAMP | KBD | VAR | CITE | ABBR | AC
<!ENTITY % special
"A | IMG | APPLET | OBJECT | FONT | BASEFONT | BR |
MAP | Q | SUB | SUP | SPAN | BDO | IFRAME">

<!ENTITY % formctrl "INPUT | SELECT | TEXTAREA | LABEL
<!-- %inline; covers inline or "text-level" elements -

```

```

<!ENTITY % inline "#PCDATA | %fontstyle; | %phrase; |
<!ELEMENT (%fontstyle;|%phrase;) - - (%inline;)*>

<!ATTLIST (%fontstyle;|%phrase;)

    %attrs;                -- %coreattrs,
    >

<!ELEMENT (SUB|SUP) - - (%inline;)*        -- subscript, s
<!ATTLIST (SUB|SUP)

    %attrs;                -- %coreattrs,
    >

<!ELEMENT SPAN - - (%inline;)*              -- generic lang
<!ATTLIST SPAN

    %attrs;                -- %coreattrs,
    %reserved;            -- reserved for
    >

<!ELEMENT BDO - - (%inline;)*              -- I18N BiDi ov
<!ATTLIST BDO

    %coreattrs;            -- id, class, s
    lang                    %LanguageCode; #IMPLIED -- language cod

```

```

    dir            (ltr|rtl)            #REQUIRED -- directionali
>

<!ELEMENT BASEFONT - O EMPTY           -- base font si
<!ATTLIST BASEFONT
    id            ID                    #IMPLIED -- document-wid
    size          CDATA                 #REQUIRED -- base font si
    color         %Color;              #IMPLIED -- text color --
    face          CDATA                 #IMPLIED -- comma-separa
>

<!ELEMENT FONT - - (%inline;)*         -- local change
<!ATTLIST FONT
    %coreattrs;          -- id, class, s
    %i18n;              -- lang, dir --
    size              CDATA             #IMPLIED -- [+|-]nn e.g.
    color            %Color;           #IMPLIED -- text color --
    face             CDATA             #IMPLIED -- comma-separa
>

<!ELEMENT BR - O EMPTY                 -- forced line b
<!ATTLIST BR

```

```

%coreattrs;                                -- id, class, style
clear          (left|all|right|none) none -- control of
>

<!--===== HTML content models =====>

<!--
    HTML has two basic content models:

        %inline;      character level elements and text
        %block;       block-like elements e.g. paragraphs

-->

<!ENTITY % block
    "P | %heading; | %list; | %preformatted; | DL | D
    NOSCRIPT | NOFRAMES | BLOCKQUOTE | FORM | ISINDEX
    TABLE | FIELDSET | ADDRESS">

<!ENTITY % flow "%block; | %inline;">

<!--===== Document Body =====>

<!ELEMENT BODY O O (%flow;)* +(INS|DEL) -- document body

<!ATTLIST BODY

    %attrs;                                -- %coreattrs,

```

```

    onload          %Script;      #IMPLIED    -- the document
    onunload        %Script;      #IMPLIED    -- the document
    background      %URI;         #IMPLIED    -- texture tile
    %bodycolors;                                -- bgcolor, text
>

<!ELEMENT ADDRESS - - ((%inline;)|P)*    -- information
<!ATTLIST ADDRESS
    %attrs;                                -- %coreattrs,
>

<!ELEMENT DIV - - (%flow;)*              -- generic language
<!ATTLIST DIV
    %attrs;                                -- %coreattrs,
    %align;                                -- align, text
    %reserved;                             -- reserved for
>

<!ELEMENT CENTER - - (%flow;)*           -- shorthand for
<!ATTLIST CENTER
    %attrs;                                -- %coreattrs,
>

```



```

<!--===== The Anchor Element =====>

<!ENTITY % Shape "(rect|circle|poly|default)">

<!ENTITY % Coords "CDATA" -- comma-separated list of 1

<!ELEMENT A - - (%inline;)* -(A)          -- anchor -->

<!ATTLIST A

    %attrs;                                -- %coreattrs,

    charset      %Charset;      #IMPLIED -- char encoding

    type         %ContentType;  #IMPLIED -- advisory con

    name         CDATA          #IMPLIED -- named link e

    href         %URI;          #IMPLIED -- URI for link

    hreflang     %LanguageCode; #IMPLIED -- language cod

    target       %FrameTarget;  #IMPLIED -- render in th

    rel          %LinkTypes;    #IMPLIED -- forward link

    rev          %LinkTypes;    #IMPLIED -- reverse link

    accesskey    %Character;    #IMPLIED -- accessibilit

    shape        %Shape;        rect      -- for use with

    coords       %Coords;       #IMPLIED -- for use with

    tabindex     NUMBER         #IMPLIED -- position in

```

```

onfocus      %Script;          #IMPLIED  -- the element
onblur        %Script;          #IMPLIED  -- the element
>

<!--===== Client-side image maps =====>
<!-- These can be placed in the same document or group
       separate document although this isn't yet widely
<!--=====
<!ELEMENT MAP - - ((%block;) | AREA)+ -- client-side image
<!ATTLIST MAP
    %attrs;                      -- %coreattrs,
    name          CDATA          #REQUIRED -- for reference
>

<!ELEMENT AREA - O EMPTY          -- client-side image
<!ATTLIST AREA
    %attrs;                      -- %coreattrs,
    shape         %Shape;        rect      -- controls inter
    coords        %Coords;       #IMPLIED  -- comma-separated
    href          %URI;          #IMPLIED  -- URI for link
    target        %FrameTarget;  #IMPLIED  -- render in th
    nohref        (nohref)       #IMPLIED  -- this region

```

```

alt          %Text;          #REQUIRED -- short description
tabindex     NUMBER          #IMPLIED  -- position in tab order
accesskey    %Character;     #IMPLIED  -- accessibility key
onfocus     %Script;        #IMPLIED  -- the element has focus
onblur       %Script;        #IMPLIED  -- the element loses focus

```

>

<!--===== The LINK Element =====>

<!--

Relationship values can be used in principle:

- a) for document specific toolbars/menus when used with the LINK element in document head e.g.  
start, contents, previous, next, index, end, help
- b) to link to a separate style sheet (rel=stylesheet)
- c) to make a link to a script (rel=script)
- d) by stylesheets to control how collections of html nodes are rendered into printed documents
- e) to make a link to a printable version of this document e.g. a postscript or pdf version (rel=alternate)

```

-->

<!ELEMENT LINK - O EMPTY                                -- a media-index

<!ATTLIST LINK

    %attrs;                                              -- %coreattrs,

    charset      %Charset;      #IMPLIED  -- char encoding

    href         %URI;          #IMPLIED  -- URI for link

    hreflang     %LanguageCode; #IMPLIED  -- language code

    type         %ContentType;  #IMPLIED  -- advisory content

    rel          %LinkTypes;    #IMPLIED  -- forward link

    rev          %LinkTypes;    #IMPLIED  -- reverse link

    media        %MediaDesc;    #IMPLIED  -- for rendering

    target       %FrameTarget;  #IMPLIED  -- render in the

    >

<!--===== Images =====>

<!-- Length defined in strict DTD for cellpadding/cellpadding

<!ENTITY % Length "CDATA" -- nn for pixels or nn% for percentage

<!ENTITY % MultiLength "CDATA" -- pixel, percentage, or percentage

<![ %HTML.Frameset; [

<!ENTITY % MultiLengths "CDATA" -- comma-separated list

```

```
]]>
```

```
<!ENTITY % Pixels "CDATA" -- integer representing length
```

```
<!ENTITY % IAlign "(top|middle|bottom|left|right)" --
```

```
<!-- To avoid problems with text-only UAs as well as
```

```
to make image content understandable and navigable
```

```
to users of non-visual UAs, you need to provide
```

```
a description with ALT, and avoid server-side image
```

```
<!ELEMENT IMG - O EMPTY -- Embedded image
```

```
<!ATTLIST IMG
```

```
  %attrs; -- %coreattrs,
```

```
  src          %URI;          #REQUIRED -- URI of image
```

```
  alt          %Text;         #REQUIRED -- short description
```

```
  longdesc     %URI;          #IMPLIED  -- link to long
```

```
                                (complements
```

```
  name         CDATA          #IMPLIED  -- name of image
```

```
  height       %Length;       #IMPLIED  -- override height
```

```
  width        %Length;       #IMPLIED  -- override width
```

```
  usemap       %URI;          #IMPLIED  -- use client-side
```

```

        ismap          (ismap)          #IMPLIED  -- use server-side
        align          %IAAlign;        #IMPLIED  -- vertical or horizontal
        border         %Pixels;         #IMPLIED  -- link border width
        hspace         %Pixels;         #IMPLIED  -- horizontal gutter
        vspace         %Pixels;         #IMPLIED  -- vertical gutter

    >

<!-- USEMAP points to a MAP element which may be in the same document
      or an external document, although the latter is not recommended.
-->
<!--===== OBJECT =====>
<!--
    OBJECT is used to embed objects as part of HTML page.
    PARAM elements should precede other content. SGML mini-model
    technicality precludes specifying this formally.
-->

<!ELEMENT OBJECT - - (PARAM | %flow;)*

    -- generic embedded object -->

<!ATTLIST OBJECT

    %attrs;                                -- %coreattrs, %classes, %id, %style

    declare          (declare)          #IMPLIED  -- declare but not use

```

|            |               |          |                 |
|------------|---------------|----------|-----------------|
| classid    | %URI;         | #IMPLIED | -- identifies a |
| codebase   | %URI;         | #IMPLIED | -- base URI for |
| data       | %URI;         | #IMPLIED | -- reference to |
| type       | %ContentType; | #IMPLIED | -- content type |
| codetype   | %ContentType; | #IMPLIED | -- content type |
| archive    | CDATA         | #IMPLIED | -- space-separa |
| standby    | %Text;        | #IMPLIED | -- message to s |
| height     | %Length;      | #IMPLIED | -- override hei |
| width      | %Length;      | #IMPLIED | -- override wid |
| usemap     | %URI;         | #IMPLIED | -- use client-s |
| name       | CDATA         | #IMPLIED | -- submit as pa |
| tabindex   | NUMBER        | #IMPLIED | -- position in  |
| align      | %IAAlign;     | #IMPLIED | -- vertical or  |
| border     | %Pixels;      | #IMPLIED | -- link border  |
| hspace     | %Pixels;      | #IMPLIED | -- horizontal g |
| vspace     | %Pixels;      | #IMPLIED | -- vertical gut |
| %reserved; |               |          | -- reserved for |

>

```

<!ELEMENT PARAM - O EMPTY                                -- named property
<!--
  id          ID          #IMPLIED  -- document-wide
  name        CDATA       #REQUIRED -- property name
  value       CDATA       #IMPLIED  -- property value
  valuetype   (DATA|REF|OBJECT) DATA -- How to interpret
  type        %ContentType; #IMPLIED -- content type
                                           when valuetype is REF or OBJECT
-->

<!--===== Java APPLET =====>
<!--
  One of code or object attributes must be present.

  Place PARAM elements before other content.
-->

<!ELEMENT APPLET - - (PARAM | %flow;)* -- Java applet
<!--
  %coreattrs; -- id, class, style
  codebase    %URI;          #IMPLIED  -- optional base URL
  archive     CDATA          #IMPLIED  -- comma-separated list of

```



```

code          CDATA          #IMPLIED  -- applet class
object        CDATA          #IMPLIED  -- serialized applet
alt           %Text;         #IMPLIED  -- short description
name          CDATA          #IMPLIED  -- allows applet
width         %Length;       #REQUIRED -- initial width
height        %Length;       #REQUIRED -- initial height
align         %IAAlign;      #IMPLIED  -- vertical or horizontal
hspace        %Pixels;       #IMPLIED  -- horizontal gutter
vspace        %Pixels;       #IMPLIED  -- vertical gutter
>

<!--===== Horizontal Rule =====>

<!ELEMENT HR - O EMPTY -- horizontal rule -->

<!ATTLIST HR
    %attrs;                  -- %coreattrs, %classes, %style
    align                    (left|center|right) #IMPLIED
    noshade                  (noshade)          #IMPLIED
    size                     %Pixels;           #IMPLIED
    width                    %Length;           #IMPLIED

```

```

>

<!--===== Paragraphs =====>

<!ELEMENT P - O (%inline;)*           -- paragraph -->

<!ATTLIST P

    %attrs;                -- %coreattrs, %class, %style
    %align;                 -- align, text-align

>

<!--===== Headings =====>

<!--
    There are six levels of headings from H1 (the most important)
    to H6 (the least important).

-->

<!ELEMENT (%heading;) - - (%inline;)* -- heading -->

<!ATTLIST (%heading;)

    %attrs;                -- %coreattrs, %class, %style
    %align;                 -- align, text-align

>

<!--===== Preformatted Text =====>

<!-- excludes markup for images and changes in font size

```

```
<!ENTITY % pre.exclusion "IMG|OBJECT|APPLET|BIG|SMALL|
<!--===== Inline Quotes =====
<!ELEMENT Q - - (%inline;)* -- short inline
<!ATTLIST Q
%attrs; -- %coreattrs,
cite %URI; #IMPLIED -- URI for source
>
<!--===== Block-like Quotes =====
<!ELEMENT BLOCKQUOTE - - (%flow;)* -- long quotation
<!ATTLIST BLOCKQUOTE
%attrs; -- %coreattrs,
cite %URI; #IMPLIED -- URI for source
>
```

```
<!--===== Inserted/Deleted Text =====>

<!-- INS/DEL are handled by inclusion on BODY -->

<!ELEMENT (INS|DEL) - - (%flow;)*          -- inserted text

<!ATTLIST (INS|DEL)

    %attrs;                                -- %coreattrs,

    cite          %URI;          #IMPLIED    -- info on reason

    datetime      %Datetime;      #IMPLIED    -- date and time

    >

<!--===== Lists =====>

<!-- definition lists - DT for term, DD for its definition

<!ELEMENT DL - - (DT|DD)+                  -- definition list

<!ATTLIST DL

    %attrs;                                -- %coreattrs,

    compact       (compact)         #IMPLIED    -- reduced interline spacing

    >

<!ELEMENT DT - O (%inline;)*                -- definition term

<!ELEMENT DD - O (%flow;)*                  -- definition description

<!ATTLIST (DT|DD)

    %attrs;                                -- %coreattrs,
```

>

```
<!-- Ordered lists (OL) numbering style
```

```
1 arabic numbers      1, 2, 3, ...
```

```
a lower alpha        a, b, c, ...
```

```
A upper alpha        A, B, C, ...
```

```
i lower roman        i, ii, iii, ...
```

```
I upper roman        I, II, III, ...
```

The style is applied to the sequence number which is

is reset to 1 for the first list item in an ordered

This can't be expressed directly in SGML due to ca

```
-->
```

```
<!ENTITY % OLStyle "CDATA" -- constrained
```

```
<!ELEMENT OL - - (LI)+ -- ordered list
```

```
<!ATTLIST OL
```

```
  %attrs; -- %coreattrs, %listattrs
```

```
  type          %OLStyle;      #IMPLIED -- numbering style
```

```
  compact       (compact)      #IMPLIED -- reduced interline
```

```
  start         NUMBER          #IMPLIED -- starting sequence
```

```

>

<!-- Unordered Lists (UL) bullet styles -->

<!ENTITY % ULStyle "(disc|square|circle)">

<!ELEMENT UL - - (LI)+ -- unordered li

<!ATTLIST UL

    %attrs; -- %coreattrs,

    type          %ULStyle;          #IMPLIED -- bullet style

    compact       (compact)          #IMPLIED -- reduced inte

>

<!ELEMENT (DIR|MENU) - - (LI)+ -(%block;) -- directory

<!ATTLIST DIR

    %attrs; -- %coreattrs,

    compact       (compact)          #IMPLIED -- reduced inte

>

<!ATTLIST MENU

    %attrs; -- %coreattrs,

    compact       (compact)          #IMPLIED -- reduced inte

>

<!ENTITY % LIStyle "CDATA" -- constrained

```

```

<!ELEMENT LI - O (%flow;)* -- list item --
<!--
<!ATTLIST LI
    %attrs; -- %coreattrs,
    type %LIStyle; #IMPLIED -- list item style
    value NUMBER #IMPLIED -- reset sequence
>

<!--===== Forms =====>
<!ELEMENT FORM - - (%flow;)* -(FORM) -- interactive
<!--
<!ATTLIST FORM
    %attrs; -- %coreattrs,
    action %URI; #REQUIRED -- server-side
    method (GET|POST) GET -- HTTP method
    enctype %ContentType; "application/x-www-form-urlencoded"
    accept %ContentTypes; #IMPLIED -- list of MIME
    name CDATA #IMPLIED -- name of form
    onsubmit %Script; #IMPLIED -- the form was
    onreset %Script; #IMPLIED -- the form was
    target %FrameTarget; #IMPLIED -- render in th

```

```

    accept-charset %Charsets;    #IMPLIED    -- list of supported charsets
  >

<!-- Each label must not contain more than ONE field -->
<!ELEMENT LABEL - - (%inline;)* -(LABEL) -- form field
<!ATTLIST LABEL
    id             ID              #REQUIRED
    %attrs;        -- %coreattrs, %classes, %types, %events
    for            IDREF          #IMPLIED    -- matches fieldset
    accesskey      %Character;    #IMPLIED    -- accessibility
    onfocus       %Script;        #IMPLIED    -- the element has focus
    onblur         %Script;        #IMPLIED    -- the element loses focus
  >

<!ENTITY % InputType
    "(TEXT | PASSWORD | CHECKBOX |
     RADIO | SUBMIT | RESET |
     FILE | HIDDEN | IMAGE | BUTTON)"
  >

<!-- attribute name required for all but submit and reset -->
<!ELEMENT INPUT - O EMPTY          -- form control
<!ATTLIST INPUT

```



%attrs;			-- %coreattrs,
type	%InputType;	TEXT	-- what kind of
name	CDATA	#IMPLIED	-- submit as pa
value	CDATA	#IMPLIED	-- specify for
checked	(checked)	#IMPLIED	-- for radio bu
disabled	(disabled)	#IMPLIED	-- unavailable
readonly	(readonly)	#IMPLIED	-- for text and
size	CDATA	#IMPLIED	-- specific to
maxlength	NUMBER	#IMPLIED	-- max chars fo
src	%URI;	#IMPLIED	-- for fields w
alt	CDATA	#IMPLIED	-- short descri
usemap	%URI;	#IMPLIED	-- use client-s
ismap	(ismap)	#IMPLIED	-- use server-s
tabindex	NUMBER	#IMPLIED	-- position in
accesskey	%Character;	#IMPLIED	-- accessibilit
onfocus	%Script;	#IMPLIED	-- the element
onblur	%Script;	#IMPLIED	-- the element
onselect	%Script;	#IMPLIED	-- some text wa

```

    onchange      %Script;          #IMPLIED  -- the element
accept           %ContentTypes; #IMPLIED  -- list of MIME
align            %IAlign;          #IMPLIED  -- vertical or
%reserved;              -- reserved for
>

<!ELEMENT SELECT - - (OPTGROUP|OPTION)+ -- option sele
<!ATTLIST SELECT
    %attrs;              -- %coreattrs,
name                    CDATA          #IMPLIED  -- field name -
size                    NUMBER         #IMPLIED  -- rows visible
multiple                (multiple)    #IMPLIED  -- default is s
disabled                (disabled)    #IMPLIED  -- unavailable
tabindex               NUMBER         #IMPLIED  -- position in
onfocus                %Script;       #IMPLIED  -- the element
onblur                 %Script;       #IMPLIED  -- the element
onchange                %Script;       #IMPLIED  -- the element
%reserved;              -- reserved for
>

<!ELEMENT OPTGROUP - - (OPTION)+ -- option group -->

```

```

<!ATTLIST OPTGROUP

    %attrs;                                -- %coreattrs,
    disabled      (disabled)      #IMPLIED -- unavailable
    label         %Text;          #REQUIRED -- for use in h
>

<!ELEMENT OPTION - O (#PCDATA)           -- selectable c

<!ATTLIST OPTION

    %attrs;                                -- %coreattrs,
    selected      (selected)      #IMPLIED
    disabled      (disabled)      #IMPLIED -- unavailable
    label         %Text;          #IMPLIED -- for use in h
    value         CDATA           #IMPLIED -- defaults to
>

<!ELEMENT TEXTAREA - - (#PCDATA)         -- multi-line t

<!ATTLIST TEXTAREA

    %attrs;                                -- %coreattrs,
    name          CDATA           #IMPLIED
    rows          NUMBER          #REQUIRED

```

```

cols            NUMBER            #REQUIRED

disabled        (disabled)        #IMPLIED    -- unavailable

readonly        (readonly)        #IMPLIED

tabindex        NUMBER            #IMPLIED    -- position in

accesskey        %Character;       #IMPLIED    -- accessibility

onfocus         %Script;          #IMPLIED    -- the element

onblur           %Script;          #IMPLIED    -- the element

onselect         %Script;          #IMPLIED    -- some text was

onchange         %Script;          #IMPLIED    -- the element

%reserved;                                -- reserved for

>

<!--

#PCDATA is to solve the mixed content problem,

per specification only whitespace is allowed there!

-->

<!ELEMENT FIELDSET - - (#PCDATA,LEGEND, (%flow;)* ) -- f

<!ATTLIST FIELDSET

%attrs;                                -- %coreattrs,

>

```

```

<!ELEMENT LEGEND - - (%inline;)*          -- fieldset legend
<!ENTITY % LAlign "(top|bottom|left|right)">

<!ATTLIST LEGEND

    %attrs;                                -- %coreattrs, %formattrs
    accesskey    %Character;    #IMPLIED  -- accessibility
    align        %LAlign;       #IMPLIED  -- relative to parent
    >

<!ELEMENT BUTTON - -

    (%flow;)* - (A|%formctrl;|FORM|ISINDEX|FIELDSET|IFRAME)
    -- push button -->

<!ATTLIST BUTTON

    %attrs;                                -- %coreattrs, %formattrs
    name          CDATA              #IMPLIED
    value         CDATA              #IMPLIED  -- sent to server
    type          (button|submit|reset) submit -- for use with FORM
    disabled      (disabled)         #IMPLIED  -- unavailable
    tabindex      NUMBER              #IMPLIED  -- position in tab order
    accesskey     %Character;        #IMPLIED  -- accessibility

```

```

onfocus      %Script;          #IMPLIED  -- the element
onblur        %Script;          #IMPLIED  -- the element
%reserved;                                -- reserved for
>

<!--===== Tables =====>

<!-- IETF HTML table standard, see [RFC1942] -->

<!--

The BORDER attribute sets the thickness of the frame
table. The default units are screen pixels.

The FRAME attribute specifies which parts of the frame
the table should be rendered. The values are not the
CALCS to avoid a name clash with the VALIGN attribute.
The value "border" is included for backwards compatib
<TABLE BORDER> which yields frame=border and border=i
For <TABLE BORDER=1> you get border=1 and frame=impli
case, it is appropriate to treat this as frame=border
compatibility with deployed browsers.

-->

<!ENTITY % TFrame "(void|above|below|hsides|lhs|rhs|vs

```

```
<!--
```

```
The RULES attribute defines which rules to draw between
```

```
If RULES is absent then assume:
```

```
    "none" if BORDER is absent or BORDER=0 otherwise
```

```
-->
```

```
<!ENTITY % TRules "(none | groups | rows | cols | all)"
```

```
<!-- horizontal placement of table relative to document
```

```
<!ENTITY % TAlign "(left|center|right)">
```

```
<!-- horizontal alignment attributes for cell contents
```

```
<!ENTITY % cellhalign
```

```
    "align          (left|center|right|justify|char) #IMPLIED
```

```
    char           %Character;          #IMPLIED -- alignment character
```

```
    charoff        %Length;             #IMPLIED -- offset for alignment
```

```
>
```

```
<!-- vertical alignment attributes for cell contents --
```

```
<!ENTITY % cellvalign
```

```
    "valign        (top|middle|bottom|baseline) #IMPLIED"
```

>

<!ELEMENT TABLE - -

(CAPTION?, (COL\*|COLGROUP\*), THEAD?, TFOOT?, TBODY\*)

<!ELEMENT CAPTION - - (%inline;)\* -- table caption

<!ELEMENT THEAD - O (TR)+ -- table header

<!ELEMENT TFOOT - O (TR)+ -- table footer

<!ELEMENT TBODY O O (TR)+ -- table body -

<!ELEMENT COLGROUP - O (COL)\* -- table column

<!ELEMENT COL - O EMPTY -- table column

<!ELEMENT TR - O (TH|TD)+ -- table row --

<!ELEMENT (TH|TD) - O (%flow;)\* -- table header

<!ATTLIST TABLE -- table element

%attrs; -- %coreattrs, %table

summary %Text; #IMPLIED -- purpose/structure

width %Length; #IMPLIED -- table width

border %Pixels; #IMPLIED -- controls frame

frame %TFrame; #IMPLIED -- which parts of

rules %TRules; #IMPLIED -- rulings between

cellspacing %Length; #IMPLIED -- spacing between



```

        cellpadding %Length;          #IMPLIED -- spacing with
align          %TAlign;              #IMPLIED -- table position
bgcolor        %Color;              #IMPLIED -- background color
%reserved;                                -- reserved for
datapagesize CDATA                  #IMPLIED -- reserved for
>

<!ENTITY % CAlign "(top|bottom|left|right)">

<!ATTLIST CAPTION

    %attrs;                            -- %coreattrs,
align          %CAlign;                #IMPLIED -- relative to
>

<!--

COLGROUP groups a set of COL elements. It allows you to
several semantically related columns together.

-->

<!ATTLIST COLGROUP

    %attrs;                            -- %coreattrs,
span          NUMBER                  1      -- default number

```

```

width          %MultiLength;  #IMPLIED  -- default width
%cellhalign;                                -- horizontal a
%cellvalign;                                -- vertical ali
>
<!--
COL elements define the alignment properties for cell
one or more columns.

The WIDTH attribute specifies the width of the column

width=64          width in screen pixels

width=0.5*        relative width of 0.5

The SPAN attribute causes the attributes of one
COL element to apply to more than one column.

-->
<!--
-- column group
%attrs;                                -- %coreattrs,
span          NUMBER          1          -- COL attribute
width          %MultiLength;  #IMPLIED  -- column width
%cellhalign;                                -- horizontal a
%cellvalign;                                -- vertical ali

```

>

<!--

Use THEAD to duplicate headers when breaking table across page boundaries, or for static headers when TBODY sections are rendered in scrolling panel.

Use TFOOT to duplicate footers when breaking table across page boundaries, or for static footers when TBODY sections are rendered in scrolling panel.

Use multiple TBODY sections when rules are needed between groups of table rows.

-->

```
<!--ATTLIST (THEAD|TBODY|TFOOT)          -- table section
%attrs;                                   -- %coreattrs, %
%cellhalign;                             -- horizontal alignment
%cellvalign;                             -- vertical alignment
-->
```

```
<!--ATTLIST TR                            -- table row --
%attrs;                                   -- %coreattrs, %
```

```

%cellhalign;                                -- horizontal a
%cellvalign;                                -- vertical ali
bgcolor      %Color;                        #IMPLIED  -- background c
>

<!-- Scope is simpler than headers attribute for commo
<!ENTITY % Scope "(row|col|rowgroup|colgroup)">

<!-- TH is for headers, TD for data, but for cells act
<!ATTLIST (TH|TD)                            -- header or da
%attrs;                                       -- %coreattrs,
abbr        %Text;                          #IMPLIED  -- abbreviation
axis        CDATA                          #IMPLIED  -- comma-separa
headers     IDREFS                         #IMPLIED  -- list of id's
scope       %Scope;                        #IMPLIED  -- scope covere
rowspan     NUMBER                         1          -- number of ro
colspan     NUMBER                         1          -- number of co
%cellhalign;                                -- horizontal a
%cellvalign;                                -- vertical ali
nowrap      (nowrap)                       #IMPLIED  -- suppress wor
bgcolor     %Color;                        #IMPLIED  -- cell backgro

```

```

width          %Length;          #IMPLIED  -- width for ce
height         %Length;          #IMPLIED  -- height for c
>

<!--===== Document Frames =====>

<!--

The content model for HTML documents depends on whether
the document is followed by a FRAMESET or BODY element. The widespread
use of the BODY start tag makes it impractical to define the
content model without the use of a marked section.

-->

<![ %HTML.Frameset; [

<!ELEMENT FRAMESET - - ((FRAMESET|FRAME)+ & NOFRAMES?)

<!ATTLIST FRAMESET

    %coreattrs;                -- id, class, s

    rows          %MultiLengths; #IMPLIED  -- list of leng
    cols          %MultiLengths; #IMPLIED  -- list of leng
    onload        %Script;       #IMPLIED  -- all the fram
    onunload      %Script;       #IMPLIED  -- all the fram

```

```

>

]]>

<![ %HTML.Frameset; [

<!-- reserved frame names start with "_" otherwise sta

<!ELEMENT FRAME - O EMPTY          -- subwindow --

<!ATTLIST FRAME

    %coreattrs;                    -- id, class, s

    longdesc      %URI;            #IMPLIED -- link to long
                                     (complements

    name          CDATA            #IMPLIED -- name of fram

    src           %URI;            #IMPLIED -- source of fr

    frameborder  (1|0)            1        -- request fram

    marginwidth  %Pixels;          #IMPLIED -- margin width

    marginheight %Pixels;          #IMPLIED -- margin heigh

    noresize     (noresize)        #IMPLIED -- allow users

    scrolling     (yes|no|auto)    auto      -- scrollbar or

>

]]>

<!ELEMENT IFRAME - - (%flow;)*      -- inline subwi

```

```
<!ATTLIST IFRAME
```

```
    %coreattrs;                -- id, class, style, ...

    longdesc    %URI;          #IMPLIED -- link to long
                                   (complements name)

    name        CDATA          #IMPLIED -- name of frame

    src         %URI;          #IMPLIED -- source of frame

    frameborder (1|0)         1      -- request frame border

    marginwidth %Pixels;        #IMPLIED -- margin width

    marginheight %Pixels;       #IMPLIED -- margin height

    scrolling    (yes|no|auto)   auto   -- scrollbar or no

    align       %IAAlign;       #IMPLIED -- vertical or horizontal

    height      %Length;        #IMPLIED -- frame height

    width       %Length;        #IMPLIED -- frame width

>
```

```
<![ %HTML.Frameset; [
```

```
<!ENTITY % noframes.content "(BODY) -(NOFRAMES)">
```

```
]]>
```

```
<!ENTITY % noframes.content "(%flow;)*">
```

```

<!ELEMENT NOFRAMES - - %noframes.content;

    -- alternate content container for non frame-based re:

<!ATTLIST NOFRAMES

    %attrs;                                -- %coreattrs,

    >

<!--===== Document Head =====>

<!-- %head.misc; defined earlier on as "SCRIPT|STYLE|M

<!ENTITY % head.content "TITLE & ISINDEX? & BASE?">

<!ELEMENT HEAD O O (%head.content;) +(%head.misc;) --

<!ATTLIST HEAD

    %i18n;                                -- lang, dir --

    profile      %URI;                    #IMPLIED -- named dictio:

    >

<!-- The TITLE element is not considered part of the f

    It should be displayed, for example as the page

    window title. Exactly one title is required per

    -->

<!ELEMENT TITLE - - (#PCDATA) -(%head.misc;) -- docume:

<!ATTLIST TITLE %i18n>

```



```

<!ELEMENT ISINDEX - O EMPTY                                -- single line
<!--
<!ATTLIST ISINDEX
    %coreattrs;                                             -- id, class, s
    %i18n;                                                  -- lang, dir --
    prompt          %Text;          #IMPLIED              -- prompt messa
<!ELEMENT BASE - O EMPTY                                    -- document bas
<!--
<!ATTLIST BASE
    href            %URI;          #IMPLIED              -- URI that act
    target          %FrameTarget;  #IMPLIED              -- render in th
    >
<!ELEMENT META - O EMPTY                                    -- generic meta
<!--
<!ATTLIST META
    %i18n;                                                  -- lang, dir, f
    http-equiv      NAME          #IMPLIED              -- HTTP respons
    name            NAME          #IMPLIED              -- metainformat
    content         CDATA         #REQUIRED              -- associated i
    scheme          CDATA         #IMPLIED              -- select form
    >

```

```

<!ELEMENT STYLE - - %StyleSheet          -- style info -
<!--
<!ATTLIST STYLE

    %i18n;                                -- lang, dir, f
    type          %ContentType;    #REQUIRED -- content type
    media          %MediaDesc;      #IMPLIED  -- designed for
    title          %Text;           #IMPLIED  -- advisory tit
    >

<!ELEMENT SCRIPT - - %Script;            -- script state
<!--
<!ATTLIST SCRIPT

    charset        %Charset;        #IMPLIED  -- char encodin
    type           %ContentType;    #REQUIRED -- content type
    language        CDATA            #IMPLIED  -- predefined s
    src            %URI;             #IMPLIED  -- URI for an e
    defer           (defer)          #IMPLIED  -- UA may defer
    event           CDATA            #IMPLIED  -- reserved for
    for            %URI;             #IMPLIED  -- reserved for
    >

<!ELEMENT NOSCRIPT - - (%flow;)*

    -- alternate content container for non script-based

```

```

<!ATTLIST NOSCRIPT

    %attrs;                                -- %coreattrs,

    >

<!--===== Document Structure =====>

<!ENTITY % version "version CDATA #FIXED '%HTML.Versio:

<![ %HTML.Frameset; [

<!ENTITY % html.content "HEAD, FRAMESET">

]]>

<!ENTITY % html.content "HEAD, BODY">

<!ELEMENT HTML O O (%html.content;)      -- document root

<!ATTLIST HTML

    %i18n;                                -- lang, dir --

    %version;

    >

```

## Appendix E. The XHTML 1.0 DTD

The XHTML 1.0 standard is formally defined as three XML Document Type Definitions (DTDs): the Strict DTD, the Transitional DTD, and the Frameset DTD. These DTDs correspond to the respective HTML 4.01 DTDs, defining the same elements and attributes using XML instead of SGML as the DTD authoring language.

The Strict DTD defines only those elements that are not deprecated in the HTML 4.01 standard. Ideally, everyone would create XHTML documents that conform to the Strict DTD. The Transitional DTD includes all those deprecated elements and more accurately reflects the HTML in use today, with many older elements still in common use. The Frameset DTD is identical to the Transitional DTD, with the exception that the document `<body>` is replaced by the `<frameset>` element.

Since the HTML Transitional DTD is the one upon which this book is based, it is only appropriate that we include the corresponding XHTML DTD. Note that we have reprinted this DTD verbatim and have not attempted to add extensions to it. Where our description and the DTD deviate, assume the DTD is correct.

```
<!--
```

```
Extensible HTML version 1.0 Transitional DTD
```

```
This is the same as HTML 4.0 Transitional except for  
changes due to the differences between XML and SGML
```

```
Namespace = http://www.w3.org/1999/xhtml
```

```
For further information, see: http://www.w3.org/TR/
```

```
Copyright (c) 1998-2000 W3C (MIT, INRIA, Keio),
```

All Rights Reserved.

This DTD module is identified by the PUBLIC and SYS

PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"

SYSTEM "http://www.w3.org/TR/xhtml1/DTD/xhtml1-tran

\$Revision: 1.2 \$

\$Date: 2002/10/07 21:48:25 \$

-->

<!--===== Character mnemonic entities =====>

<!ENTITY % HTMLlat1 PUBLIC

"-//W3C//ENTITIES Latin 1 for XHTML//EN"

"xhtml-lat1.ent">

%HTMLlat1;

<!ENTITY % HTMLsymbol PUBLIC

"-//W3C//ENTITIES Symbols for XHTML//EN"

"xhtml-symbol.ent">

%HTMLsymbol;

<!ENTITY % HTMLspecial PUBLIC

"-//W3C//ENTITIES Special for XHTML//EN"

"xhtml-special.ent">

```
%HTMLspecial;

<!--===== Imported Names =====>

<!ENTITY % ContentType "CDATA">

    <!-- media type, as per [RFC2045] -->

<!ENTITY % ContentTypes "CDATA">

    <!-- comma-separated list of media types, as per [RFC2045] -->

<!ENTITY % Charset "CDATA">

    <!-- a character encoding, as per [RFC2045] -->

<!ENTITY % Charsets "CDATA">

    <!-- a space separated list of character encodings -->

<!ENTITY % LanguageCode "NMTOKEN">

    <!-- a language code, as per [RFC1766] -->

<!ENTITY % Character "CDATA">

    <!-- a single character from [ISO10646] -->

<!ENTITY % Number "CDATA">

    <!-- one or more digits -->

<!ENTITY % LinkTypes "CDATA">

    <!-- space-separated list of link types -->
```

<!ENTITY % MediaDesc "CDATA">

<!-- single or comma-separated list of media descr.

<!ENTITY % URI "CDATA">

<!-- a Uniform Resource Identifier, see [RFC2396]

<!ENTITY % UriList "CDATA">

<!-- a space separated list of Uniform Resource Id

<!ENTITY % Datetime "CDATA">

<!-- date and time information. ISO date format -->

<!ENTITY % Script "CDATA">

<!-- script expression -->

<!ENTITY % StyleSheet "CDATA">

<!-- style sheet data -->

<!ENTITY % Text "CDATA">

<!-- used for titles etc. -->

<!ENTITY % FrameTarget "NMTOKEN">

<!-- render in this frame -->

<!ENTITY % Length "CDATA">

<!-- nn for pixels or nn% for percentage length -->

<!ENTITY % MultiLength "CDATA">

```

        <!-- pixel, percentage, or relative -->

<!ENTITY % MultiLengths "CDATA">

        <!-- comma-separated list of MultiLength -->

<!ENTITY % Pixels "CDATA">

        <!-- integer representing length in pixels -->

<!-- these are used for image maps -->

<!ENTITY % Shape "(rect|circle|poly|default)">

<!ENTITY % Coords "CDATA">

        <!-- comma separated list of lengths -->

<!-- used for object, applet, img, input and iframe -->

<!ENTITY % ImgAlign "(top|middle|bottom|left|right)">

<!-- a color using sRGB: #RRGGBB as Hex values -->

<!ENTITY % Color "CDATA">

<!-- There are also 16 widely known color names with t

    Black   = #000000      Green   = #008000

    Silver  = #C0C0C0      Lime    = #00FF00

    Gray    = #808080      Olive   = #808000

    White   = #FFFFFF      Yellow  = #FFFF00

```



```

    Maroon = #800000      Navy    = #000080

    Red     = #FF0000     Blue    = #0000FF

    Purple  = #800080     Teal    = #008080

    Fuchsia= #FF00FF     Aqua    = #00FFFF

-->

<!--===== Generic Attributes =====>

<!-- core attributes common to most elements

    id          document-wide unique id

    class       space separated list of classes

    style       associated style info

    title       advisory title/amplification

-->

<!ENTITY % coreattrs

    "id          ID          #IMPLIED

    class       CDATA       #IMPLIED

    style       %StyleSheet; #IMPLIED

    title       %Text;      #IMPLIED"

>

<!-- internationalization attributes

```

lang            language code (backwards compatible)

xml:lang       language code (as per XML 1.0 spec)

dir            direction for weak/neutral text

-->

<!ENTITY % i18n

"lang            %LanguageCode; #IMPLIED

xml:lang        %LanguageCode; #IMPLIED

dir            (ltr|rtl)            #IMPLIED"

>

<!-- attributes for common UI events

onclick        a pointer button was clicked

ondblclick     a pointer button was double clicked

onmousedown    a pointer button was pressed down

onmouseup      a pointer button was released

onmousemove    a pointer was moved onto the element

onmouseout     a pointer was moved away from the element

onkeypress     a key was pressed and released

onkeydown      a key was pressed down

onkeyup        a key was released

-->

<!ENTITY % events

```
"onclick        %Script;        #IMPLIED
ondblclick     %Script;        #IMPLIED
onmousedown    %Script;        #IMPLIED
onmouseup      %Script;        #IMPLIED
onmouseover    %Script;        #IMPLIED
onmousemove    %Script;        #IMPLIED
onmouseout     %Script;        #IMPLIED
onkeypress     %Script;        #IMPLIED
onkeydown      %Script;        #IMPLIED
onkeyup        %Script;        #IMPLIED"
>
```

<!-- attributes for elements that can get the focus

accesskey      accessibility key character

tabindex       position in tabbing order

onfocus       the element got the focus

onblur         the element lost the focus

-->

<!ENTITY % focus

"accesskey %Character; #IMPLIED

tabindex %Number; #IMPLIED

onfocus %Script; #IMPLIED

onblur %Script; #IMPLIED"

>

<!ENTITY % attrs "%coreattrs; %i18n; %events;">

<!-- text alignment for p, div, h1-h6. The default is  
align="left" for ltr headings, "right" for rtl -->

<!ENTITY % TextAlign "align (left|center|right) #IMPLIED"

<!--===== Text Elements =====>

<!ENTITY % special

"br | span | bdo | object | applet | img | map | if.

<!ENTITY % fontstyle "tt | i | b | big | small | u

| s | strike | font | basefont">

<!ENTITY % phrase "em | strong | dfn | code | q | sub

samp | kbd | var | cite | abbr | ac.

```

<!ENTITY % inline.forms "input | select | textarea | li
<!-- these can occur at block or inline level -->

<!ENTITY % misc "ins | del | script | noscript">

<!ENTITY % inline "a | %special; | %fontstyle; | %phra
<!-- %Inline; covers inline or "text-level" elements -
<!ENTITY % Inline "(#PCDATA | %inline; | %misc;)*">

<!--===== Block level elements =====>

<!ENTITY % heading "h1|h2|h3|h4|h5|h6">

<!ENTITY % lists "ul | ol | dl | menu | dir">

<!ENTITY % blocktext "pre | hr | blockquote | address
<!ENTITY % block

    "p | %heading; | div | %lists; | %blocktext; | isi:
<!ENTITY % Block "(%block; | form | %misc;)*">

<!-- %Flow; mixes Block and Inline and is used for lis
<!ENTITY % Flow "(#PCDATA | %block; | form | %inline;
<!--===== Content models for exclusions =====>

<!-- a elements use %Inline; excluding a -->

<!ENTITY % a.content

    "(#PCDATA | %special; | %fontstyle; | %phrase; | %i:

```

```

<!-- pre uses %Inline excluding img, object, applet, b
      sub, sup, font, or basefont -->

<!ENTITY % pre.content
      "(#PCDATA | a | br | span | bdo | map | tt | i | b
      %phrase; | %inline.forms;)*">

<!-- form uses %Flow; excluding form -->

<!ENTITY % form.content "(#PCDATA | %block; | %inline;
<!-- button uses %Flow; but excludes a, form, form con
<!ENTITY % button.content
      "(#PCDATA | p | %heading; | div | %lists; | %blockt
      table | br | span | bdo | object | applet | img
      %fontstyle; | %phrase; | %misc;)*">

<!--===== Document Structure =====
<!-- the namespace URI designates the document profile
<!ELEMENT html (head, body)>
<!ATTLIST html
      %i18n;
      xmlns          %URI;          #FIXED 'http://www.w3.org

```

```

>

<!--===== Document Head =====>

<!ENTITY % head.misc "(script|style|meta|link|object|i

<!-- content model is %head.misc; combined with a sing

        title and an optional base element in any order -

<!ELEMENT head (%head.misc;,

        ((title, %head.misc;, (base, %head.misc;)? ) |

        (base, %head.misc;, (title, %head.misc;))))>

<!ATTLIST head

        %i18n;

        profile      %URI;          #IMPLIED

>

<!-- The title element is not considered part of the f

        It should be displayed, for example as the page

        window title. Exactly one title is required per

-->

<!ELEMENT title (#PCDATA)>

<!ATTLIST title %i18n;>

<!-- document base URI -->

```

```
<!ELEMENT base EMPTY>
```

```
<!ATTLIST base
```

```
    href          %URI;          #IMPLIED
```

```
    target        %FrameTarget;  #IMPLIED
```

```
>
```

```
<!-- generic metainformation -->
```

```
<!ELEMENT meta EMPTY>
```

```
<!ATTLIST meta
```

```
    %i18n;
```

```
    http-equiv    CDATA          #IMPLIED
```

```
    name          CDATA          #IMPLIED
```

```
    content       CDATA          #REQUIRED
```

```
    scheme        CDATA          #IMPLIED
```

```
>
```

```
<!--
```

Relationship values can be used in principle:

- a) for document specific toolbars/menus when used  
with the link element in document head e.g.



start, contents, previous, next, index, end, help

b) to link to a separate style sheet (rel="stylesheet")

c) to make a link to a script (rel="script")

d) by stylesheets to control how collections of  
html nodes are rendered into printed documents

e) to make a link to a printable version of this document  
e.g. a PostScript or PDF version (rel="alternate")

-->

<!ELEMENT link EMPTY>

<!ATTLIST link

%attrs;

|         |           |          |
|---------|-----------|----------|
| charset | %Charset; | #IMPLIED |
|---------|-----------|----------|

|      |       |          |
|------|-------|----------|
| href | %URI; | #IMPLIED |
|------|-------|----------|

|          |                |          |
|----------|----------------|----------|
| hreflang | %LanguageCode; | #IMPLIED |
|----------|----------------|----------|

|      |               |          |
|------|---------------|----------|
| type | %ContentType; | #IMPLIED |
|------|---------------|----------|

|     |             |          |
|-----|-------------|----------|
| rel | %LinkTypes; | #IMPLIED |
|-----|-------------|----------|

|     |             |          |
|-----|-------------|----------|
| rev | %LinkTypes; | #IMPLIED |
|-----|-------------|----------|

|       |             |          |
|-------|-------------|----------|
| media | %MediaDesc; | #IMPLIED |
|-------|-------------|----------|

|        |               |          |
|--------|---------------|----------|
| target | %FrameTarget; | #IMPLIED |
|--------|---------------|----------|

>

<!-- style info, which may include CDATA sections -->

<!ELEMENT style (#PCDATA)>

<!ATTLIST style

  %i18n;

  type           %ContentType;   #REQUIRED

  media          %MediaDesc;     #IMPLIED

  title          %Text;          #IMPLIED

  xml:space      (preserve)       #FIXED 'preserve'

>

<!-- script statements, which may include CDATA sections -->

<!ELEMENT script (#PCDATA)>

<!ATTLIST script

  charset        %Charset;       #IMPLIED

  type           %ContentType;   #REQUIRED

  language       CDATA            #IMPLIED

  src            %URI;            #IMPLIED

  defer          (defer)          #IMPLIED

```

    xml:space      (preserve)      #FIXED 'preserve'

>

<!-- alternate content container for non script-based

<!ELEMENT noscript %Flow;>

<!ATTLIST noscript

    %attrs;

>

<!--===== Frames =====>

<!-- inline subwindow -->

<!ELEMENT iframe %Flow;>

<!ATTLIST iframe

    %coreattrs;

    longdesc      %URI;          #IMPLIED

    name          NMTOKEN        #IMPLIED

    src           %URI;          #IMPLIED

    frameborder   (1|0)          "1"

    marginwidth   %Pixels;        #IMPLIED

    marginheight  %Pixels;        #IMPLIED

    scrolling      (yes|no|auto)   "auto"

```

```

    align          %ImgAlign;          #IMPLIED

    height         %Length;            #IMPLIED

    width          %Length;            #IMPLIED

>

<!-- alternate content container for non frame-based r

<!ELEMENT noframes %Flow;>

<!ATTLIST noframes

    %attrs;

>

<!--===== Document Body =====>

<!ELEMENT body %Flow;>

<!ATTLIST body

    %attrs;

    onload        %Script;            #IMPLIED

    onunload       %Script;            #IMPLIED

    background     %URI;               #IMPLIED

    bgcolor        %Color;             #IMPLIED

    text           %Color;             #IMPLIED

```

```
link          %Color;          #IMPLIED
```

```
vlink         %Color;          #IMPLIED
```

```
alink         %Color;          #IMPLIED
```

```
>
```

```
<!ELEMENT div %Flow;>  <!-- generic language/style con
```

```
<!ATTLIST div
```

```
  %attrs;
```

```
  %TextAlign;
```

```
>
```

```
<!--===== Paragraphs =====>
```

```
<!ELEMENT p %Inline;>
```

```
<!ATTLIST p
```

```
  %attrs;
```

```
  %TextAlign;
```

```
>
```

```
<!--===== Headings =====>
```

```
<!--
```

There are six levels of headings from h1 (the most important) to h6 (the least important).

-->

<!ELEMENT h1 %Inline;>

<!ATTLIST h1

%attrs;

%TextAlign;

>

<!ELEMENT h2 %Inline;>

<!ATTLIST h2

%attrs;

%TextAlign;

>

<!ELEMENT h3 %Inline;>

<!ATTLIST h3

%attrs;

%TextAlign;

>

<!ELEMENT h4 %Inline;>

<!ATTLIST h4

```

    %attrs;

    %TextAlign;

>

<!ELEMENT h5 %Inline;>

<!ATTLIST h5

    %attrs;

    %TextAlign;

>

<!ELEMENT h6 %Inline;>

<!ATTLIST h6

    %attrs;

    %TextAlign;

>

<!--===== Lists =====>

<!-- Unordered list bullet styles -->

<!ENTITY % ULStyle "(disc|square|circle)">

<!-- Unordered list -->

<!ELEMENT ul (li)+>

<!ATTLIST ul

```

```
%attrs;
```

```
type          %ULStyle;      #IMPLIED
```

```
compact       (compact)      #IMPLIED
```

```
>
```

```
<!-- Ordered list numbering style
```

```
1    arabic numbers          1, 2, 3, ...
```

```
a    lower alpha             a, b, c, ...
```

```
A    upper alpha             A, B, C, ...
```

```
i    lower roman             i, ii, iii, ...
```

```
I    upper roman             I, II, III, ...
```

```
The style is applied to the sequence number which is
```

```
is reset to 1 for the first list item in an ordered
```

```
-->
```

```
<!ENTITY % OLStyle "CDATA">
```

```
<!-- Ordered (numbered) list -->
```

```
<!ELEMENT ol (li)+>
```

```
<!ATTLIST ol
```

```
%attrs;
```



```

    type          %OLStyle;          #IMPLIED

    compact       (compact)          #IMPLIED

    start         %Number;           #IMPLIED

>

<!-- single column list (DEPRECATED) -->

<!ELEMENT menu (li)+>

<!ATTLIST menu

    %attrs;

    compact       (compact)          #IMPLIED

>

<!-- multiple column list (DEPRECATED) -->

<!ELEMENT dir (li)+>

<!ATTLIST dir

    %attrs;

    compact       (compact)          #IMPLIED

>

<!-- LIStyle is constrained to: "(%ULStyle;|%OLStyle;)"

<!ENTITY % LIStyle "CDATA">

<!-- list item -->

```

```
<!ELEMENT li %Flow;>
```

```
<!ATTLIST li
```

```
    %attrs;
```

```
    type          %LIStyle;          #IMPLIED
```

```
    value         %Number;           #IMPLIED
```

```
>
```

```
<!-- definition lists - dt for term, dd for its defini
```

```
<!ELEMENT dl (dt|dd)+>
```

```
<!ATTLIST dl
```

```
    %attrs;
```

```
    compact       (compact)          #IMPLIED
```

```
>
```

```
<!ELEMENT dt %Inline;>
```

```
<!ATTLIST dt
```

```
    %attrs;
```

```
>
```

```
<!ELEMENT dd %Flow;>
```

```
<!ATTLIST dd
```

```

    %attrs;

>

<!--===== Address =====>

<!-- information on author -->

<!ELEMENT address %Inline;>

<!ATTLIST address

    %attrs;

>

<!--===== Horizontal Rule =====>

<!ELEMENT hr EMPTY>

<!ATTLIST hr

    %attrs;

    align      (left|center|right) #IMPLIED

    noshade     (noshade)          #IMPLIED

    size        %Pixels;           #IMPLIED

    width       %Length;           #IMPLIED

>

<!--===== Preformatted Text =====>

<!-- content is %Inline; excluding

```

"img|object|applet|big|small|sub|sup|font|base

<!ELEMENT pre %pre.content;>

<!ATTLIST pre

%attrs;

width %Number; #IMPLIED

xml:space (preserve) #FIXED 'preserve'

>

<!--===== Block-like Quotes =====>

<!ELEMENT blockquote %Flow;>

<!ATTLIST blockquote

%attrs;

cite %URI; #IMPLIED

>

<!--===== Text alignment =====>

<!-- center content -->

<!ELEMENT center %Flow;>

<!ATTLIST center

%attrs;

>

<!--===== Inserted/Deleted Text =====>

<!--

ins/del are allowed in block and inline content, but inappropriate to include block content within an ins occurring in inline content.

-->

<!ELEMENT ins %Flow;>

<!ATTLIST ins

%attrs;

cite %URI; #IMPLIED

datetime %Datetime; #IMPLIED

>

<!ELEMENT del %Flow;>

<!ATTLIST del

%attrs;

cite %URI; #IMPLIED

datetime %Datetime; #IMPLIED

>

<!--===== The Anchor Element =====>

<!-- content is %Inline; except that anchors shouldn't

<!ELEMENT a %a.content;>

<!ATTLIST a

  %attrs;

  charset      %Charset;      #IMPLIED

  type         %ContentType;  #IMPLIED

  name         NMTOKEN        #IMPLIED

  href         %URI;          #IMPLIED

  hreflang     %LanguageCode; #IMPLIED

  rel          %LinkTypes;     #IMPLIED

  rev          %LinkTypes;     #IMPLIED

  accesskey    %Character;     #IMPLIED

  shape        %Shape;         "rect"

  coords       %Coords;        #IMPLIED

  tabindex     %Number;        #IMPLIED

  onfocus     %Script;        #IMPLIED

  onblur       %Script;        #IMPLIED

```

    target          %FrameTarget;  #IMPLIED

>

<!--===== Inline Elements =====>

<!ELEMENT span %Inline;> <!-- generic language/style class -->

<!ATTLIST span

    %attrs;

>

<!ELEMENT bdo %Inline;> <!-- I18N BiDi over-ride -->

<!ATTLIST bdo

    %coreattrs;

    %events;

    lang          %LanguageCode;  #IMPLIED

    xml:lang      %LanguageCode;  #IMPLIED

    dir           (ltr|rtl)       #REQUIRED

>

<!ELEMENT br EMPTY> <!-- forced line break -->

<!ATTLIST br

    %coreattrs;

    clear         (left|all|right|none) "none"

```

>

<!ELEMENT em %Inline;> <!-- emphasis -->

<!ATTLIST em %attrs;>

<!ELEMENT strong %Inline;> <!-- strong emphasis -->

<!ATTLIST strong %attrs;>

<!ELEMENT dfn %Inline;> <!-- definitional -->

<!ATTLIST dfn %attrs;>

<!ELEMENT code %Inline;> <!-- program code -->

<!ATTLIST code %attrs;>

<!ELEMENT samp %Inline;> <!-- sample -->

<!ATTLIST samp %attrs;>

<!ELEMENT kbd %Inline;> <!-- something user would type -->

<!ATTLIST kbd %attrs;>

<!ELEMENT var %Inline;> <!-- variable -->

<!ATTLIST var %attrs;>

<!ELEMENT cite %Inline;> <!-- citation -->

<!ATTLIST cite %attrs;>

<!ELEMENT abbr %Inline;> <!-- abbreviation -->



```
<!ATTLIST abbr %attrs;>

<!ELEMENT acronym %Inline;>    <!-- acronym -->

<!ATTLIST acronym %attrs;>

<!ELEMENT q %Inline;>    <!-- inlined quote -->

<!ATTLIST q

    %attrs;

    cite          %URI;          #IMPLIED

    >

<!ELEMENT sub %Inline;> <!-- subscript -->

<!ATTLIST sub %attrs;>

<!ELEMENT sup %Inline;> <!-- superscript -->

<!ATTLIST sup %attrs;>

<!ELEMENT tt %Inline;>    <!-- fixed pitch font -->

<!ATTLIST tt %attrs;>

<!ELEMENT i %Inline;>    <!-- italic font -->

<!ATTLIST i %attrs;>

<!ELEMENT b %Inline;>    <!-- bold font -->

<!ATTLIST b %attrs;>

<!ELEMENT big %Inline;>    <!-- bigger font -->
```

```

<!ATTLIST big %attrs;>

<!ELEMENT small %Inline;>    <!-- smaller font -->

<!ATTLIST small %attrs;>

<!ELEMENT u %Inline;>    <!-- underline -->

<!ATTLIST u %attrs;>

<!ELEMENT s %Inline;>    <!-- strike-through -->

<!ATTLIST s %attrs;>

<!ELEMENT strike %Inline;>    <!-- strike-through -->

<!ATTLIST strike %attrs;>

<!ELEMENT basefont EMPTY>    <!-- base font size -->

<!ATTLIST basefont

    id            ID            #IMPLIED

    size          CDATA         #REQUIRED

    color         %Color;       #IMPLIED

    face          CDATA         #IMPLIED

    >

<!ELEMENT font %Inline;> <!-- local change to font -->

<!ATTLIST font

```

```

%coreattrs;

%il8n;

size          CDATA          #IMPLIED

color         %Color;        #IMPLIED

face          CDATA          #IMPLIED

>

<!--===== Object =====>

<!--

    object is used to embed objects as part of HTML page
    param elements should precede other content. Paramet
    can also be expressed as attribute/value pairs on th
    object element itself when brevity is desired.

-->

<!ELEMENT object (#PCDATA | param | %block; | form | %

<!ATTLIST object

    %attrs;

    declare    (declare)      #IMPLIED

    classid    %URI;          #IMPLIED

    codebase    %URI;          #IMPLIED

```

data	%URI;	#IMPLIED
type	%ContentType;	#IMPLIED
codetype	%ContentType;	#IMPLIED
archive	%UriList;	#IMPLIED
standby	%Text;	#IMPLIED
height	%Length;	#IMPLIED
width	%Length;	#IMPLIED
usemap	%URI;	#IMPLIED
name	NMTOKEN	#IMPLIED
tabindex	%Number;	#IMPLIED
align	%ImgAlign;	#IMPLIED
border	%Pixels;	#IMPLIED
hspace	%Pixels;	#IMPLIED
vspace	%Pixels;	#IMPLIED

>

<!--

param is used to supply a named property value.

In XML it would seem natural to follow RDF and suppo

abbreviated syntax where the param elements are replaced by attribute value pairs on the object start tag.

-->

<!ELEMENT param EMPTY>

<!ATTLIST param

|           |                   |           |
|-----------|-------------------|-----------|
| id        | ID                | #IMPLIED  |
| name      | CDATA             | #REQUIRED |
| value     | CDATA             | #IMPLIED  |
| valuetype | (data ref object) | "data"    |
| type      | %ContentType;     | #IMPLIED  |
| >         |                   |           |

<!--===== Java applet =====>

<!--

One of code or object attributes must be present.

Place param elements before other content.

-->

<!ELEMENT applet (#PCDATA | param | %block; | form | %

<!ATTLIST applet

%coreattrs;

|          |            |           |
|----------|------------|-----------|
| codebase | %URI;      | #IMPLIED  |
| archive  | CDATA      | #IMPLIED  |
| code     | CDATA      | #IMPLIED  |
| object   | CDATA      | #IMPLIED  |
| alt      | %Text;     | #IMPLIED  |
| name     | NMTOKEN    | #IMPLIED  |
| width    | %Length;   | #REQUIRED |
| height   | %Length;   | #REQUIRED |
| align    | %ImgAlign; | #IMPLIED  |
| hspace   | %Pixels;   | #IMPLIED  |
| vspace   | %Pixels;   | #IMPLIED  |

>

<!--===== Images =====>

<!--

To avoid accessibility problems for people who aren't able to see the image, you should provide a text description using the alt and longdesc attributes.

In addition, avoid the use of server-side image map

-->

<!ELEMENT img EMPTY>

<!ATTLIST img

  %attrs;

  src          %URI;          #REQUIRED

  alt          %Text;         #REQUIRED

  name         NMTOKEN        #IMPLIED

  longdesc     %URI;          #IMPLIED

  height       %Length;        #IMPLIED

  width         %Length;        #IMPLIED

  usemap        %URI;          #IMPLIED

  ismap         (ismap)        #IMPLIED

  align         %ImgAlign;      #IMPLIED

  border        %Length;        #IMPLIED

  hspace        %Pixels;        #IMPLIED

  vspace        %Pixels;        #IMPLIED

>

<!-- usemap points to a map element which may be in the same document or an external document, although the latter is not recommended -->

```
<!--===== Client-side image maps =====>
```

```
<!-- These can be placed in the same document or group
```

```
       separate document although this isn't yet widely
```

```
<!ELEMENT map ((%block; | form | %misc;)+ | area+)>
```

```
<!ATTLIST map
```

```
    %i18n;
```

```
    %events;
```

```
    id          ID          #REQUIRED
```

```
    class       CDATA       #IMPLIED
```

```
    style       %StyleSheet; #IMPLIED
```

```
    title       %Text;      #IMPLIED
```

```
    name        CDATA       #IMPLIED
```

```
>
```

```
<!ELEMENT area EMPTY>
```

```
<!ATTLIST area
```

```
    %attrs;
```

```
    shape       %Shape;     "rect"
```

```
    coords      %Coords;    #IMPLIED
```



href	%URI;	#IMPLIED
nohref	(nohref)	#IMPLIED
alt	%Text;	#REQUIRED
tabindex	%Number;	#IMPLIED
accesskey	%Character;	#IMPLIED
onfocus	%Script;	#IMPLIED
onblur	%Script;	#IMPLIED
target	%FrameTarget;	#IMPLIED

>

<!--===== Forms =====>

<!ELEMENT form %form.content;>    <!-- forms shouldn't !

<!ATTLIST form

|          |               |                            |
|----------|---------------|----------------------------|
| %attrs;  |               |                            |
| action   | %URI;         | #REQUIRED                  |
| method   | (get post)    | "get"                      |
| name     | NMTOKEN       | #IMPLIED                   |
| enctype  | %ContentType; | "application/x-www-form-u. |
| onsubmit | %Script;      | #IMPLIED                   |
| onreset  | %Script;      | #IMPLIED                   |

```

    accept          %ContentTypes; #IMPLIED

    accept-charset  %Charsets;    #IMPLIED

    target          %FrameTarget; #IMPLIED

>

<!--

    Each label must not contain more than ONE field

    Label elements shouldn't be nested.

-->

<!ELEMENT label %Inline;>

<!ATTLIST label

    %attrs;

    for              IDREF          #IMPLIED

    accesskey        %Character;    #IMPLIED

    onfocus         %Script;       #IMPLIED

    onblur           %Script;       #IMPLIED

>

<!ENTITY % InputType

    "(text | password | checkbox |

```

```

        radio | submit | reset |

        file | hidden | image | button)"

    >

<!-- the name attribute is required for all but submit

<!ELEMENT input EMPTY>          <!-- form control -->

<!ATTLIST input

    %attrs;

    type          %InputType;      "text"

    name          CDATA            #IMPLIED

    value         CDATA            #IMPLIED

    checked       (checked)        #IMPLIED

    disabled      (disabled)       #IMPLIED

    readonly      (readonly)       #IMPLIED

    size          CDATA            #IMPLIED

    maxlength     %Number;         #IMPLIED

    src           %URI;            #IMPLIED

    alt           CDATA            #IMPLIED

    usemap        %URI;            #IMPLIED

    tabindex      %Number;         #IMPLIED

```

|           |             |          |
|-----------|-------------|----------|
| accesskey | %Character; | #IMPLIED |
|-----------|-------------|----------|

|         |          |          |
|---------|----------|----------|
| onfocus | %Script; | #IMPLIED |
|---------|----------|----------|

|        |          |          |
|--------|----------|----------|
| onblur | %Script; | #IMPLIED |
|--------|----------|----------|

|          |          |          |
|----------|----------|----------|
| onselect | %Script; | #IMPLIED |
|----------|----------|----------|

|          |          |          |
|----------|----------|----------|
| onchange | %Script; | #IMPLIED |
|----------|----------|----------|

|        |                |          |
|--------|----------------|----------|
| accept | %ContentTypes; | #IMPLIED |
|--------|----------------|----------|

|       |            |          |
|-------|------------|----------|
| align | %ImgAlign; | #IMPLIED |
|-------|------------|----------|

>

<!ELEMENT select (optgroup|option)+> <!-- option sele

<!ATTLIST select

  %attrs;

|      |       |          |
|------|-------|----------|
| name | CDATA | #IMPLIED |
|------|-------|----------|

|      |          |          |
|------|----------|----------|
| size | %Number; | #IMPLIED |
|------|----------|----------|

|          |            |          |
|----------|------------|----------|
| multiple | (multiple) | #IMPLIED |
|----------|------------|----------|

|          |            |          |
|----------|------------|----------|
| disabled | (disabled) | #IMPLIED |
|----------|------------|----------|

|          |          |          |
|----------|----------|----------|
| tabindex | %Number; | #IMPLIED |
|----------|----------|----------|

|         |          |          |
|---------|----------|----------|
| onfocus | %Script; | #IMPLIED |
|---------|----------|----------|

|        |          |          |
|--------|----------|----------|
| onblur | %Script; | #IMPLIED |
|--------|----------|----------|

```

    onchange      %Script;          #IMPLIED

>

<!-- ELEMENT optgroup (option)+>      <!-- option group -->

<!-- ATTLIST optgroup

    %attrs;

    disabled      (disabled)        #IMPLIED

    label         %Text;            #REQUIRED

>

<!-- ELEMENT option (#PCDATA)>         <!-- selectable choice

<!-- ATTLIST option

    %attrs;

    selected      (selected)        #IMPLIED

    disabled      (disabled)        #IMPLIED

    label         %Text;            #IMPLIED

    value         CDATA              #IMPLIED

>

<!-- ELEMENT textarea (#PCDATA)>       <!-- multi-line text

<!-- ATTLIST textarea

    %attrs;

```

|           |             |           |
|-----------|-------------|-----------|
| name      | CDATA       | #IMPLIED  |
| rows      | %Number;    | #REQUIRED |
| cols      | %Number;    | #REQUIRED |
| disabled  | (disabled)  | #IMPLIED  |
| readonly  | (readonly)  | #IMPLIED  |
| tabindex  | %Number;    | #IMPLIED  |
| accesskey | %Character; | #IMPLIED  |
| onfocus   | %Script;    | #IMPLIED  |
| onblur    | %Script;    | #IMPLIED  |
| onselect  | %Script;    | #IMPLIED  |
| onchange  | %Script;    | #IMPLIED  |

>

<!--

The fieldset element is used to group form fields.

Only one legend element should occur in the content

and if present should only be preceded by whitespace

-->

<!ELEMENT fieldset (#PCDATA | legend | %block; | form

```

<!ATTLIST fieldset

    %attrs;

>

<!ENTITY % LAlign "(top|bottom|left|right)">

<!ELEMENT legend %Inline;>      <!-- fieldset label -->

<!ATTLIST legend

    %attrs;

    accesskey    %Character;      #IMPLIED

    align        %LAlign;         #IMPLIED

>

<!--

    Content is %Flow; excluding a, form, form controls, i

-->

<!ELEMENT button %button.content;>  <!-- push button -->

<!ATTLIST button

    %attrs;

    name          CDATA            #IMPLIED

    value         CDATA            #IMPLIED

    type          (button|submit|reset) "submit"

```

|          |            |          |
|----------|------------|----------|
| disabled | (disabled) | #IMPLIED |
|----------|------------|----------|

|          |          |          |
|----------|----------|----------|
| tabindex | %Number; | #IMPLIED |
|----------|----------|----------|

|           |             |          |
|-----------|-------------|----------|
| accesskey | %Character; | #IMPLIED |
|-----------|-------------|----------|

|         |          |          |
|---------|----------|----------|
| onfocus | %Script; | #IMPLIED |
|---------|----------|----------|

|        |          |          |
|--------|----------|----------|
| onblur | %Script; | #IMPLIED |
|--------|----------|----------|

>

<!-- single-line text input control (DEPRECATED) -->

<!ELEMENT isindex EMPTY>

<!ATTLIST isindex

    %coreattrs;

    %i18n;

|        |        |          |
|--------|--------|----------|
| prompt | %Text; | #IMPLIED |
|--------|--------|----------|

>

<!--===== Tables =====>

<!-- Derived from IETF HTML table standard, see [RFC19

<!--

The border attribute sets the thickness of the frame of the table. The default units are screen pixels.



The frame attribute specifies which parts of the frame the table should be rendered. The values are not the CALS to avoid a name clash with the valign attribute.

-->

<!ENTITY % TFrame "(void|above|below|hsides|lhs|rhs|vsides)"

<!--

The rules attribute defines which rules to draw between cells. If rules is absent then assume:

"none" if border is absent or border="0" otherwise

-->

<!ENTITY % TRules "(none | groups | rows | cols | all)"

<!-- horizontal placement of table relative to document

<!ENTITY % TAlign "(left|center|right)">

<!-- horizontal alignment attributes for cell contents

char alignment char, e.g. char=':'

charoff offset for alignment char

-->

<!ENTITY % cellhalign

```

"align      (left|center|right|justify|char) #IMPLIED
char        %Character;      #IMPLIED
charoff     %Length;         #IMPLIED"
>

<!-- vertical alignment attributes for cell contents -->
<!ENTITY % cellvalign
"valign     (top|middle|bottom|baseline) #IMPLIED"
>

<!ELEMENT table
      (caption?, (col*|colgroup*), thead?, tfoot?, (tbody|tfoot)*)
>
<!ELEMENT caption %Inline;>
<!ELEMENT thead   (tr)+>
<!ELEMENT tfoot   (tr)+>
<!ELEMENT tbody   (tr)+>
<!ELEMENT colgroup (col)*>
<!ELEMENT col     EMPTY>
<!ELEMENT tr      (th|td)+>
<!ELEMENT th      %Flow;>

```

```
<!ELEMENT td %Flow;>
```

```
<!ATTLIST table
```

```
  %attrs;
```

```
  summary %Text; #IMPLIED
```

```
  width %Length; #IMPLIED
```

```
  border %Pixels; #IMPLIED
```

```
  frame %TFrame; #IMPLIED
```

```
  rules %TRules; #IMPLIED
```

```
  cellspacing %Length; #IMPLIED
```

```
  cellpadding %Length; #IMPLIED
```

```
  align %TAlign; #IMPLIED
```

```
  bgcolor %Color; #IMPLIED
```

```
>
```

```
<!ENTITY % CAlign "(top|bottom|left|right)">
```

```
<!ATTLIST caption
```

```
  %attrs;
```

```
  align %CAlign; #IMPLIED
```

```
>
```

```
<!--
```

colgroup groups a set of col elements. It allows you to group several semantically related columns together.

-->

<!ATTLIST colgroup

  %attrs;

  span          %Number;          "1"

  width         %MultiLength;  #IMPLIED

  %cellhalign;

  %cellvalign;

>

<!--

col elements define the alignment properties for cell. You can use colgroup to group one or more columns.

The width attribute specifies the width of the column.

width=64         width in screen pixels

width=0.5\*        relative width of 0.5

The span attribute causes the attributes of one col element to apply to more than one column.

-->

<!ATTLIST col

  %attrs;

  span          %Number;          "1"

  width         %MultiLength;  #IMPLIED

  %cellhalign;

  %cellvalign;

>

<!--

  Use thead to duplicate headers when breaking table  
  across page boundaries, or for static headers when  
  tbody sections are rendered in scrolling panel.

  Use tfoot to duplicate footers when breaking table  
  across page boundaries, or for static footers when  
  tbody sections are rendered in scrolling panel.

  Use multiple tbody sections when rules are needed  
  between groups of table rows.

-->

<!ATTLIST thead

%attrs;

%cellhalign;

%cellvalign;

>

<!ATTLIST tfoot

%attrs;

%cellhalign;

%cellvalign;

>

<!ATTLIST tbody

%attrs;

%cellhalign;

%cellvalign;

>

<!ATTLIST tr

%attrs;

%cellhalign;

%cellvalign;

```

    bgcolor      %Color;          #IMPLIED
  >

<!-- Scope is simpler than headers attribute for commo:

<!ENTITY % Scope "(row|col|rowgroup|colgroup)">

<!-- th is for headers, td for data and for cells acti:

<!ATTLIST th

    %attrs;

    abbr         %Text;          #IMPLIED

    axis         CDATA           #IMPLIED

    headers      IDREFS          #IMPLIED

    scope        %Scope;         #IMPLIED

    rowspan      %Number;        "1"

    colspan      %Number;        "1"

    %cellhalign;

    %cellvalign;

    nowrap       (nowrap)        #IMPLIED

    bgcolor      %Color;          #IMPLIED

    width        %Pixels;        #IMPLIED

    height       %Pixels;        #IMPLIED

```

>

<!ATTLIST td

%attrs;

abbr           %Text;           #IMPLIED

axis           CDATA           #IMPLIED

headers       IDREFS           #IMPLIED

scope          %Scope;        #IMPLIED

rowspan       %Number;        "1"

colspan       %Number;        "1"

%cellhalign;

%cellvalign;

nowrap        (nowrap)        #IMPLIED

bgcolor       %Color;        #IMPLIED

width         %Pixels;        #IMPLIED

height        %Pixels;        #IMPLIED

>



## 6.2 Referencing Documents: The URL

Every document on the Web has a unique address. (Imagine the chaos if they didn't.) The document's address is known as its *uniform resource locator* (URL).<sup>[2]</sup>

[2] "URL" usually is pronounced "you are ell," not "earl."

Several HTML/XHTML tags include a URL attribute value, including hyperlinks, inline images, and forms. All use the same URL syntax to specify the location of a web resource, regardless of the type or content of that resource. That's why it's known as a *uniform* resource locator.

Since they can be used to represent almost any resource on the Internet, URLs come in a variety of flavors. All URLs, however, have the same top-level syntax:

*scheme:scheme\_specific\_part*

The *scheme* describes the kind of object the URL references; the *scheme\_specific\_part* is, well, the part that is peculiar to the specific scheme. The important thing to note is that the *scheme* is always separated from the *scheme\_specific\_part* by a colon, with no intervening spaces.

### 6.2.1 Writing a URL

Write URLs using the displayable characters in the US-ASCII character set. For example, surely you have heard what has become annoyingly common on the radio for an announced business web site: "h, t, t, p, colon, slash, slash, w, w, w, dot, blah-blah, dot, com." That's a simple URL, written:

`http://www.blah-blah.com`

If you need to use a character in a URL that is not part of this character

set, you must encode the character using a special notation. The encoding notation replaces the desired character with three characters: a percent sign and two hexadecimal digits whose values correspond to the position of the character in the ASCII character set.

This is easier than it sounds. One of the most common special characters is the space (owners of older Macintoshes, take special notice), whose position in the character set is 20 hexadecimal. You can't type a space in a URL (well, you can, but it won't work). Rather, replace spaces in the URL with `%20`:

```
http://www.kumquat.com/new%20pricing.html
```

This URL actually retrieves a document named *new pricing.html* from the *www.kumquat.com* server.

### 6.2.1.1 Handling reserved and unsafe characters

In addition to the nonprinting characters, you'll need to encode reserved and unsafe characters in your URLs as well.

Reserved characters are those that have a specific meaning within the URL itself. For example, the slash character separates elements of a pathname within a URL. If you need to include in a URL a slash that is not intended to be an element separator, you'll need to encode it as `%2F`:  
[\[3\]](#)

[3] Hexadecimal numbering is based on 16 characters: 0 through 9 followed by A through F, which in decimal are equivalent to values 0 through 15. Also, letter case for these extended values is not significant; "a" (10 decimal) is the same as "A," for example.

```
http://www.calculator.com/compute?3%2f4
```

This URL actually references the resource named *compute* on the *www.calculator.com* server and passes the string *3/4* to it, as delineated by the question mark (`?`). Presumably, the resource is a server-side program that performs some arithmetic function on the passed value and

returns a result.

Unsafe characters are those that have no special meaning within the URL but may have a special meaning in the context in which the URL is written. For example, double quotes ("" ) delimit URL attribute values in tags. If you were to include a double quotation mark directly in a URL, you would probably confuse the browser. Instead, you should encode the double quotation mark as %22 to avoid any possible conflict.

Other reserved and unsafe characters that should always be encoded are shown in [Table 6-1](#).

**Table 6-1. Reserved and unsafe characters and their URL encodings**

| Character | Description       | Usage    | Encoding |
|-----------|-------------------|----------|----------|
| ;         | Semicolon         | Reserved | %3B      |
| /         | Slash             | Reserved | %2F      |
| ?         | Question mark     | Reserved | %3F      |
| :         | Colon             | Reserved | %3A      |
| @         | At sign           | Reserved | %40      |
| =         | Equals sign       | Reserved | %3D      |
| &         | Ampersand         | Reserved | %26      |
| <         | Less-than sign    | Unsafe   | %3C      |
| >         | Greater-than sign | Unsafe   | %3E      |

|   |                            |        |     |
|---|----------------------------|--------|-----|
| " | Double quotation mark      | Unsafe | %22 |
| # | Hash symbol                | Unsafe | %23 |
| % | Percent                    | Unsafe | %25 |
| { | Left curly brace           | Unsafe | %7B |
| } | Right curly brace          | Unsafe | %7D |
|   | Vertical bar               | Unsafe | %7C |
| \ | Backslash                  | Unsafe | %5C |
| ^ | Caret                      | Unsafe | %5E |
| ~ | Tilde                      | Unsafe | %7E |
| [ | Left square bracket        | Unsafe | %5B |
| ] | Right square bracket       | Unsafe | %5D |
| ` | Back single quotation mark | Unsafe | %60 |

In general, you should always encode a character if there is some doubt as to whether it can be placed as-is in a URL. As a rule of thumb, any character other than a letter, number, or any of the characters \$ – \_ . + ! \* ' ( ) should be encoded.

It is never an error to encode a character, unless that character has a

specific meaning in the URL. For example, encoding the slashes in an http URL causes them to be used as regular characters, not as pathname delimiters, breaking the URL.

## 6.2.2 Absolute and Relative URLs

You may address a URL in one of two ways: absolute or relative. An absolute URL is the complete address of a resource and has everything your system needs to find a document and its server on the Web. At the very least, an absolute URL contains the scheme and all required elements of the *scheme\_specific\_part* of the URL. It may also contain any of the optional portions of the *scheme\_specific\_part*.

With a relative URL, you provide an abbreviated document address that, when automatically combined with a "base address" by the system, becomes a complete address for the document. Within the relative URL, any component of the URL may be omitted. The browser automatically fills in the missing pieces of the relative URL using corresponding elements of a base URL. This base URL is usually the URL of the document containing the relative URL, but it may be another document specified with the `<base>` tag. [`<base>`]

### 6.2.2.1 Relative schemes and servers

A common form of a relative URL is missing the scheme and server name. Since many related documents are on the same server, it makes sense to omit the scheme and server name from the relative URL. For instance, assume the base document was last retrieved from the server *www.kumquat.com*. This relative URL:

```
another-doc.html
```

is equivalent to the absolute URL:

```
http://www.kumquat.com/another-doc.html
```

Table 6-2 shows how the base and relative URLs in this example are combined to form an absolute URL.

**Table 6-2. Forming an absolute URL**

|                          | Protocol                 | Server                   | Directory                | File                     |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| Base URL                 | http                     | <i>www.kumquat.com</i>   | /                        |                          |
| Relative URL             | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <i>another-doc.html</i>  |
| <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> | <input type="checkbox"/> |
| Absolute URL             | http                     | <i>www.kumquat.com</i>   | /                        | <i>another-doc.html</i>  |

#### 6.2.2.2 Relative document directories

Another common form of a relative URL omits the leading slash and one or more directory names from the beginning of the document pathname. The directory of the base URL is automatically assumed to replace these missing components. It's the most common abbreviation, because most authors place their collections of documents and subdirectories of support resources in the same directory path as the home page. For example, you might have a *special* subdirectory containing FTP files referenced in your document. Let's say that the absolute URL for that document is:

```
http://www.kumquat.com/planting/guide.html
```

A relative URL for the file *README.txt* in the *special* subdirectory looks like this:

```
ftp:special/README.txt
```

You'll actually be retrieving:

`ftp://www.kumquat.com/planting/special/README.txt`

Visually, the operation looks like that in [Table 6-3](#).

**Table 6-3. Forming an absolute FTP URL**

|                      | Protocol             | Server                 | Directory                | File                 |
|----------------------|----------------------|------------------------|--------------------------|----------------------|
| Base URL             | http                 | <i>www.kumquat.com</i> | <i>/planting</i>         | <i>guide.html</i>    |
| Relative URL         | ftp                  | <input type="text"/>   | <i>special</i>           | <i>README.txt</i>    |
| <input type="text"/> | <input type="text"/> | <input type="text"/>   | <input type="text"/>     | <input type="text"/> |
| Absolute URL         | ftp                  | <i>www.kumquat.com</i> | <i>/planting/special</i> | <i>README.txt</i>    |

### 6.2.2.3 Using relative URLs

Relative URLs are more than just a typing convenience. Because they are relative to the current server and directory, you can move an entire set of documents to another directory or even another server and never have to change a single relative link. Imagine the difficulties if you had to go into every source document and change the URL for every link every time you moved it. You'd loathe using hyperlinks! Use relative URLs wherever possible.

### 6.2.3 The http URL

The http URL is by far the most common. It is used to access documents from a web server, and it has two formats:

`http://server:port/path#fragment`

`http://server:port/path?search`

Some of the parts are optional. In fact, the most common form of the http URL is simply:

`http://server/path`

which designates the unique server and the directory path and name of a document.

### 6.2.3.1 The http server

The *server* is the unique Internet name or Internet protocol (IP) numerical address of the computer system that stores the web resource. We suspect you'll mostly use more easily remembered Internet names for the servers in your URLs.<sup>[4]</sup> The name consists of several parts, including the server's actual name and the successive names of its network domain, each part separated by a period. Typical Internet names look like *www.oreilly.com* or *hoohoo.ncsa.uiuc.edu*.<sup>[5]</sup>

[4] Each Internet-connected computer has a unique address—a numeric (IP) address, of course, because computers deal only in numbers. Humans prefer names, so the Internet folks provide us with a collection of special servers and software (the Domain Name System, or DNS) that automatically resolve Internet names into IP addresses.

[5] The three-letter suffix of the domain name identifies the type of organization or business that operates that portion of the Internet. For instance, "com" is a commercial enterprise, "edu" is an academic institution, and "gov" identifies a government-based domain. Outside the United States, a less-descriptive suffix is often assigned—typically a two-letter abbreviation of the country name, such as "jp" for Japan and "de" for Deutschland. Many organizations around the world now use the generic three-letter suffixes in place of the more conventional two-letter national suffixes.

It has become something of a convention that webmasters name their servers *www* for quick and easy identification on the Web. For instance, O'Reilly & Associates's web server's name is *www*, which, along with the publisher's domain name, becomes the very easily remembered web site *www.oreilly.com*. Similarly, ActivMedia Robotics's web server is named *www.activmedia.com*. Being a nonprofit organization, the American Kennel Club's main server has a different domain suffix: *www.akc.org*. The naming convention has very obvious benefits, which you, too, should



take advantage of if you are called upon to create a web server for your organization.

You may also specify the address of a server using its numerical IP address. The address is a sequence of four numbers, 0 to 255, separated by periods. Valid IP addresses look like 137.237.1.87 or 192.249.1.33.

It'd be a dull diversion to tell you now what the numbers mean or how to derive an IP address from a domain name, particularly since you'll rarely, if ever, use one in a URL. Rather, this is a good place to hyperlink: pick up any good Internet networking treatise for rigorous detail on IP addressing, such as Ed Krol's *The Whole Internet User's Guide and Catalog* (O'Reilly).

### 6.2.3.2 The http port

The *port* is the number of the communication port by which the client browser connects to the server. It's a networking thing servers perform many functions besides serving up web documents and resources to client browsers: electronic mail, FTP document fetches, filesystem sharing, and so on. Although all that network activity may come into the server on a single wire, it's typically divided into software-managed "ports" for service-specific communications something analogous to boxes at your local post office.

The default URL port for web servers is 80. Special secure web servers Secure HTTP (SHTTP) or Secure Sockets Layer (SSL) run on port 443. Most web servers today use port 80; you need to include a port number along with an immediately preceding colon in your URL if the target server does *not* use port 80 for web communication.

When the Web was in its infancy, pioneer webmasters ran their Wild Wild Web connections on all sorts of port numbers. For technical and security reasons, system-administrator privileges are required to install a server on port 80. Lacking such privileges, these webmasters chose other, more easily accessible, port numbers.

Now that web servers have become acceptable and are under the care and feeding of responsible administrators, documents being served on some port other than 80 or 443 should make you wonder if that server is really on the up and up. Most likely, the maverick server is being run by a clever user unbeknownst to the server's bona fide system administrators.

### 6.2.3.3 The http path

The document *path* is the Unix-style hierarchical location of the file in the server's storage system. The pathname consists of one or more names separated by slashes. All but the last name represent directories leading down to the document; the last name is usually that of the document itself.

It has become a convention that for easy identification, HTML document names end with the suffix *.html* (otherwise they're plain ASCII text files, remember?). Although recent versions of Windows allow longer suffixes, their users often stick to the three-letter *.htm* name suffix for HTML documents.

Although the server name in a URL is not case-sensitive, the document pathname may be. Since most web servers are run on Unix-based systems, and Unix filenames are case-sensitive, those document pathname will be case-sensitive, too. Web servers running on Windows machines are not case-sensitive, so those document pathnames are not. Since it is impossible to know the operating system of the server you are accessing, always assume that the server has case-sensitive pathnames and take care to get the case correct when typing your URLs.

Certain conventions regarding the document pathname have arisen. If the last element of the document path is a directory, not a single document, the server usually will send back either a listing of the directory contents or the HTML index document in that directory. You should end the document name for a directory with a trailing slash character, but in practice, most servers will honor the request even if this character is omitted.

If the directory name is just a slash alone, or nothing at all, the server decides what to serve to your browser typically, a so-called *home page* in the root directory stored as a file named `index.html`. Every well-designed web server should have an attractive, well-designed home page; it's a shorthand way for users to access your web collection, since they don't need to remember the document's actual filename, just your server's name. That's why, for example, you can type `http://www.oreilly.com` into Netscape's "Open" dialog box and get O'Reilly's home page.

Another twist: if the first component of the document path starts with the tilde character (`~`), it means that the rest of the pathname begins from the personal directory in the home directory of the specified user on the server machine. For instance, the URL `http://www.kumquat.com/~chuck /` would retrieve the top-level page from Chuck's document collection.

Different servers have different ways of locating documents within a user's home directory. Many search for the documents in a directory named `public_html`. Unix-based servers are fond of the name `index.html` for home pages. When all else fails, servers tend to cough up a directory listing or the first text document in the home page directory.

#### 6.2.3.4 The http document fragment

The *fragment* is an identifier that points to a specific section of a document. In URL specifications, it follows the server and pathname and is separated by the pound sign (`#`). A fragment identifier indicates to the browser that it should begin displaying the target document at the indicated fragment name. As we describe in more detail later in this chapter, you insert fragment names into a document either with the universal `id` tag attribute or with the `name` attribute for the `<a>` tag. Like a pathname, a fragment name may be any sequence of characters.

The fragment name and the preceding hash symbol are optional; omit them when referencing a document without defined fragments.

Formally, the fragment element applies only to HTML or XHTML

documents. If the target of the URL is some other document type, the fragment name may be misinterpreted by the browser.

Fragments are useful for long documents. By identifying key sections of your document with a fragment name, you make it easy for readers to link directly to that portion of the document, avoiding the tedium of scrolling or searching through the document to get to the section that interests them.

As a rule of thumb, we recommend that every section header in your documents be accompanied by an equivalent fragment name. By consistently following this rule, you'll make it possible for readers to jump to any section in any of your documents. Fragments also make it easier to build tables of contents for your document families.

#### **6.2.3.5 The `http search` parameter**

The *search* component of the http URL, along with its preceding question mark, is optional. It indicates that the path is a searchable or executable resource on the server. The content of the search component is passed to the server as parameters that control the search or execution function.

The actual encoding of parameters in the search component is dependent upon the server and the resource being referenced. The parameters for searchable resources are covered later in this chapter, when we discuss searchable documents. Parameters for executable resources are discussed in [Chapter 9](#).

Although our initial presentation of http URLs indicated that a URL can have either a fragment identifier or a search component, some browsers let you use both in a single URL. If you so desire, you can follow the search parameter with a fragment identifier, telling the browser to begin displaying the results of the search at the indicated fragment. Netscape, for example, supports this usage.

We don't recommend this kind of URL, though. First and foremost, it doesn't work on a lot of browsers. Just as important, using a fragment

implies that you are sure that the results of the search will have a fragment of that name defined within the document. For large document collections, this is hardly likely. You are better off omitting the fragment, showing the search results from the beginning of the document, and avoiding potential confusion among your readers.

### 6.2.3.6 Sample http URLs

Here are some sample http URLs:

```
http://www.oreilly.com/catalog.html
```

```
http://www.oreilly.com/
```

```
http://www.kumquat.com:8080/
```

```
http://www.kumquat.com/planting/guide.html#soil_prep
```

```
http://www.kumquat.com/find_a_quat?state=Florida
```

The first example is an explicit reference to a bona fide HTML document named *catalog.html* that is stored in the root directory of the *www.oreilly.com* server. The second references the top-level home page on that same server. That home page may or may not be *catalog.html*. Sample three also assumes that there is a home page in the root directory of the *www.kumquat.com* server and that the web connection is to the nonstandard port 8080.

The fourth example is the URL for retrieving the web document named *guide.html* from the *planting* directory on the *www.kumquat.com* server. Once retrieved, the browser should display the document beginning at the fragment named *soil\_prep*.

The last example invokes an executable resource named *find\_a\_quat* with the parameter named *state* set to the value *Florida*. Presumably, this resource generates an HTML or XHTML response, often a new document, that is subsequently displayed by the browser.

## 6.2.4 The file URL

The file URL is perhaps the second most common one used, but it is not readily recognized by web users and particularly web authors. It points to a file stored on a computer without indicating the protocol used to retrieve the file. As such, it has limited use in a networked environment. That's a good thing. The file URL lets you load and display a locally stored document and is particularly useful for referencing personal HTML/XHTML document collections, such as those "under construction" and not yet ready for general distribution, or document collections on CD-ROM. The file URL has the following format:

```
file://server/path
```

### 6.2.4.1 The file server

The file *server* can be, like the http one, an Internet domain name or IP address of the computer containing the file to be retrieved. Unlike http, however, which requires TCP/IP networking, the file server may also be the unqualified but unique name of a computer on a personal network, or a storage device on the same computer, such as a CD-ROM, or mapped from another networked computer. No assumptions are made as to how the browser might contact the machine to obtain the file; presumably the browser can make some connection, perhaps via a Network File System or FTP, to obtain the file.

If you omit the server name by including an extra slash (/) in the URL, or if you use the special name *localhost*, the browser retrieves the file from the machine on which the browser is running. In this case, the browser simply accesses the file using the normal facilities of the local operating system. In fact, this is the most common usage of the file URL. By creating document families on a diskette or CD-ROM and referencing your hyperlinks using the *file:///* URL, you create a distributable, standalone document collection that does not require a network connection to use.

#### 6.2.4.2 The file path

This is the path of the file to be retrieved on the desired server. The syntax of the path may differ based upon the operating system of the server; be sure to encode any potentially dangerous characters in the path.

#### 6.2.4.3 Sample file URLs

The file URL is easy:

```
file://localhost/home/chuck/document.html
```

```
file:///home/chuck/document.html
```

```
file://marketing.kumquat.com/monthly_sales.html
```

```
file://D:/monthly_sales.html
```

The first URL retrieves */home/chuck/document.html* from the user's local machine off the current storage device, typically C:\ on a Windows PC. The second is identical to the first, except we've omitted the *localhost* reference to the server; the server name defaults to the local drive.

The third example uses some protocol to retrieve *monthly\_sales.html* from the *marketing.kumquat.com* server, while the fourth example uses the local PC's operating system to retrieve the same file from the *D:\* drive or device.

#### 6.2.5 The mailto URL

The mailto URL is very common in HTML/XHTML documents. It has the browser send an electronic mail message to a named recipient. It has the format:

```
mailto:address
```

The *address* is any valid email address, usually of the form:

```
user@server
```

Thus, a typical mailto URL might look like:

```
mailto:cmusciano@aol.com
```

You may include multiple recipients in the mailto URL, separated by commas. For example, this URL addresses the message to all three recipients.

```
mailto:cmusciano@aol.com,bkennedy@activmedia.com,bookt
```

There should be no spaces before or after the commas in the URL.

#### **6.2.5.1 Defining mail header fields**

The popular browsers open an email helper or plug-in application when the user selects a mailto URL. It may be the default email program for their system, or Outlook Express with Internet Explorer, or Netscape's built-in Communicator. With some browsers, users can designate their own email programs for handling mailto URLs by altering a specification in their browsers' Internet Options or Preferences.

Like http search parameters that you attach at the end of the URL, separated by question marks (?), you include email-related parameters with the mailto URL in the HTML document. Typically, additional parameters may include the message's header fields, such as the subject, cc (carbon copy), and bcc (blind carbon copy) recipients. How these additional fields are handled depends on the email program.

A few examples are in order:

```
mailto:cmusciano@aol.com?subject=Loved your book!
```

```
mailto:cmusciano@aol.com?cc=booktech@oreilly.com
```



```
mailto:cmusciano@aol.com?bcc=archive@myserver.com
```

As you can probably guess, the first URL sets the subject of the message. Note that some email programs allow spaces in the parameter value while others do not. Annoyingly, you can't replace spaces with their hexadecimal equivalent, %20, because many email programs won't make the proper substitution. It's best to use spaces, since the email programs that don't honor the spaces simply truncate the parameter to the first word.

The second URL places the address *booktech@oreilly.com* in the cc field of the message. Similarly, the last example sets the bcc field. You may also set several fields in one URL by separating the field definitions with ampersands. For example, this URL sets the subject and carbon-copy addresses:

```
mailto:cmusciano@aol.com?subject=Loved your book!&cc=b
```

Not all email programs accept or recognize the bcc and cc extensions in the mailto URL some either ignore them or append them to a preceding subject. Thus, when forming a mailto URL, it's best to order the extra fields as subject first, followed by cc and bcc. And don't depend on the cc and bcc recipients being included in the email.

## 6.2.6 The ftp URL

The ftp URL is used to retrieve documents from an FTP (File Transfer Protocol) server.<sup>[6]</sup> It has the format:

[6] FTP is an ancient Internet protocol that dates back to the Dark Ages, around 1975. It was designed as a simple way to move files between machines and is popular and useful to this day. Many HTML/XHTML authors use FTP to place files on their web servers.

```
ftp://user:password@server:port/path;type=typecode
```

### 6.2.6.1 The ftp user and password

FTP is an authenticated service, meaning that you must have a valid username and password in order to retrieve documents from a server. However, most FTP servers also support restricted, nonauthenticated access known as *anonymous FTP*. In this mode, anyone can supply the username "anonymous" and be granted access to a limited portion of the server's documents. Most FTP servers also assume (but may not grant) anonymous access if the username and password are omitted.

If you are using an ftp URL to access a site that requires a username and password, include the *user* and *password* components in the URL, along with the colon (:) and at sign (@). More commonly, you'll be accessing an anonymous FTP server, and the user and password components can be omitted.

If you keep the user component and at sign but omit the password and the preceding colon, most browsers prompt you for a password after connecting to the FTP server. This is the recommended way of accessing authenticated resources on an FTP server; it prevents others from seeing your password.

We recommend you *never* place an ftp URL with a username and password in any HTML/XHTML document. The reasoning is simple: anyone can retrieve the simple text document, extract the username and password from the URL, log into the FTP server, and tamper with its documents.

#### **6.2.6.2 The ftp server and port**

The *ftp server* and *port* operate by the same rules as the server and port in an http URL. The server must be a valid Internet domain name or IP address, and the optional port specifies the port on which the server is listening for requests. If omitted, the default port number is 21.

#### **6.2.6.3 The ftp path and typecode**

The *path* component of an ftp URL represents a series of directories,

separated by slashes, leading to the file to be retrieved. By default, the file is retrieved as a binary file; this can be changed by adding the *typecode* (and the preceding *;type=*) to the URL.

If the typecode is set to *d*, the path is assumed to be a directory. The browser requests a listing of the directory contents from the server and displays this listing to the user. If the typecode is any other letter, it is used as a parameter to the FTP *type* command before retrieving the file referenced by the path. While some FTP servers may implement other codes, most servers accept *i* to initiate a binary transfer and *a* to treat the file as a stream of ASCII text.

#### 6.2.6.4 Sample ftp URLs

Here are some sample ftp URLs:

```
ftp://www.kumquat.com/sales/pricing
```

```
ftp://bob@bobs-box.com/results;type=d
```

```
ftp://bob:secret@bobs-box.com/listing;type=a
```

The first example retrieves the file named *pricing* from the *sales* directory on the anonymous FTP server at *www.kumquat.com*. The second logs into the FTP server on *bobs-box.com* as user *bob*, prompting for a password before retrieving the contents of the directory named *results* and displaying them to the user. The last example logs into *bobs-box.com* as *bob* with the password *secret* and retrieves the file named *listing*, treating its contents as ASCII characters.

#### 6.2.7 The javascript URL

The javascript URL actually is a pseudoprotocol, not usually included in discussions of URLs. Yet, with advanced browsers like Netscape and Internet Explorer, the javascript URL can be associated with a hyperlink and used to execute JavaScript commands when the user selects the

link. [[Section 12.3.4](#)]

### 6.2.7.1 The javascript URL arguments

What follows the javascript pseudoprotocol is one or more semicolon-separated JavaScript expressions and methods, including references to multi-expression JavaScript functions that you embed within the `<script>` tag in your documents (see [Chapter 12](#) for details). For example:

```
javascript:window.alert('Hello, world!')
```

```
javascript:doFlash('red', 'blue'); window.alert('Do not
```

are valid URLs that you may include as the value for a link reference (see [Section 6.3.1.2](#) and [Section 6.5.4.3](#)). The first example contains a single JavaScript method that activates an alert dialog with the simple message "Hello, world!"

The second javascript URL example contains two arguments: the first calls a JavaScript function, `doFlash`, which presumably you have located elsewhere in the document within the `<script>` tag and which perhaps flashes the background color of the document window between red and blue. The second expression is the same alert method as in the first example, with a slightly different message.

The javascript URL may appear in a hyperlink sans arguments, too. In that case, the Netscape browser alone not Internet Explorer opens a special JavaScript editor wherein the user may type in and test the various expressions and methods.

### 6.2.8 The news URL

Although rarely used anymore, the news URL accesses either a single message or an entire newsgroup within the Usenet news system. It has two forms:

`news:newsgroup`

`news:message_id`

An unfortunate limitation in news URLs is that they don't allow you to specify a news server. Rather, users specify news servers in their browser preferences. At one time, not long ago, Internet newsgroups were nearly universally distributed; all news servers carried all the same newsgroups and their respective articles, so one news server was as good as any. Today, the sheer bulk of disk space needed to store the daily volume of newsgroup activity is often prohibitive for any single news server, and there's also local censorship of newsgroups. Hence, you cannot expect that all newsgroups, and certainly not all articles for a particular newsgroup, will be available on the user's news server.

Many users' browsers may not be correctly configured to read news. We recommend that you avoid placing news URLs in your documents except in rare cases.

### **6.2.8.1 Accessing entire newsgroups**

There are several thousand newsgroups devoted to nearly every conceivable topic under the sun and beyond. Each group has a unique name, composed of hierarchical elements separated by periods. For example, the World Wide Web announcements newsgroup is:

`comp.infosys.www.announce`

To access this group, use the URL:

`news:comp.infosys.www.announce`

### **6.2.8.2 Accessing single messages**

Every message on a news server has a unique message identifier (ID) associated with it. This ID has the form:

*unique\_string@server*

The *unique\_string* is a sequence of ASCII characters; the server is usually the name of the machine from which the message originated. The *unique\_string* must be unique among all the messages that originated from the server. A sample URL to access a single message might be:

*news:12A7789B@news.kumquat.com*

In general, message IDs are cryptic sequences of characters not readily understood by humans. Moreover, the life span of a message on a server is usually measured in days, after which the message is deleted and the message ID is no longer valid. The bottom line: single message news URLs are difficult to create, become invalid quickly, and generally are not used.

## 6.2.9 The nntp URL

The nntp URL goes beyond the news URL to provide a complete mechanism for accessing articles in the Usenet news system. It has the form:

*nntp://server:port/newsgroup/article*

### 6.2.9.1 The nntp server and port

The *nntp server* and *port* are defined similarly to the *http server* and *port*, described earlier. The server must be the Internet domain name or IP address of an nntp server; the port is the port on which that server is listening for requests.

If the port and its preceding colon are omitted, the default port of 119 is used.

### 6.2.9.2 The nntp newsgroup and article

The *newsgroup* is the name of the group from which an article is to be retrieved, as defined in [Section 6.2.8](#). The *article* is the numeric id of the desired article within that newsgroup. Although the article number is easier to determine than a message ID, it falls prey to the same limitations of single message references using the news URL, described in [Section 6.2.8](#). Specifically, articles do not last long on most nntp servers, and nntp URLs quickly become invalid as a result.

### 6.2.9.3 Sample nntp URLs

A sample nntp URL might be:

```
nntp://news.kumquat.com/alt.fan.kumquats/417
```

This URL retrieves article 417 from the *alt.fan.kumquats* newsgroup on *news.kumquat.com*. Keep in mind that the article will be served only to machines that are allowed to retrieve articles from this server. In general, most nntp servers restrict access to those machines on the same local area network.

## 6.2.10 The telnet URL

The telnet URL opens an interactive session with a desired server, allowing the user to log in and use the machine. Often, the connection to the machine automatically starts a specific service for the user; in other cases, the user must know the commands to type to use the system. The telnet URL has the form:

```
telnet://user:password@server:port/
```

### 6.2.10.1 The telnet user and password

The telnet *user* and *password* are used exactly like the user and password components of the ftp URL, described previously. In particular, the same caveats apply regarding protecting your password and never

placing it within a URL.

Just like the ftp URL, if you omit the password from the URL, the browser should prompt you for a password just before contacting the telnet server.

If you omit both the user and password, the telnet occurs without supplying a username. For some servers, telnet automatically connects to a default service when no username is supplied. For others, the browser may prompt for a username and password when making the connection to the telnet server.

### **6.2.10.2 The telnet server and port**

The telnet server and port are defined similarly to the http server and port, described earlier. The server must be the Internet domain name or IP address of a telnet server; the port is the port on which that server is listening for requests. If the port and its preceding colon are omitted, the default port of 23 is used.

## **6.2.11 The gopher URL**

Gopher is a web-like document-retrieval system that achieved some popularity on the Internet just before the Web took off, making gopher obsolete. Some gopher servers still exist, though, and the gopher URL lets you access gopher documents.

The gopher URL has the form:

*gopher://server:port/path*

### **6.2.11.1 The gopher server and port**

The gopher server and port are defined similarly to the http server and port, described previously. The server must be the Internet domain name or IP address of a gopher server; the port is the port on which that server



is listening for requests.

If the port and its preceding colon are omitted, the default port of 70 is used.

### 6.2.11.2 The gopher path

The gopher path can take one of three forms:

*type/selector*

*type/selector%09search*

*type/selector%09search%09gopherplus*

The *type* is a single character value denoting the type of the gopher resource. If the entire path is omitted from the gopher URL, the type defaults to 1.

The *selector* corresponds to the path of a resource on the gopher server. It may be omitted, in which case the top-level index of the gopher server is retrieved.

If the gopher resource is actually a gopher search engine, the *search* component provides the string for which to search. The search string must be preceded by an encoded horizontal tab (%09).

If the gopher server supports gopher+ resources, the *gopherplus* component supplies the necessary information to locate that resource. The exact content of this component varies based upon the resources on the gopher server. This component is preceded by an encoded horizontal tab (%09). If you want to include the *gopherplus* component but omit the search component, you must still supply both encoded tabs within the URL.

## 16.3 HTML Versus XHTML

The majority of HTML is completely compatible with XHTML, and this book is devoted to that majority. In this chapter, however, we talk about the minority: where the HTML 4.01 standard and the XHTML DTD differ. If you truly desire to create documents that are both HTML- and XHTML-compliant, you must heed the various warnings and caveats we outline in the following sections.

The biggest difference that's Difference with a capital D and that spells difficult is that writing XHTML documents requires much more discipline and attention to detail than even the most fastidious HTML author ever dreamed necessary. In W3C parlance, that means your documents must be impeccably well formed. Throughout the history of HTML and in this book authors have been encouraged to create well-formed documents, but you have to break rank with the HTML standards for your documents to be considered well formed by XML standards.

Nonetheless, your efforts to master XHTML will be rewarded with documents that are well formed and a sense of satisfaction from playing by the new rules. You will truly benefit in the future, too: through XML, your documents will be able to appear in places you never dreamed would exist (mostly good places, we hope).

### 16.3.1 Correctly Nested Elements

One requirement of a well-formed XHTML document is that its elements are nested correctly. This isn't any different from in the HTML standards: simply close the markup elements in the order in which you opened them. If one element is within another, the end tag of the inner element must appear before the end tag of the outer element.

Hence, in the following well-formed XHTML segment, we end the *italics* tag before we end the **bold** one, because we started italicizing after we started bolding the content:

`<b>Close the italics tag <i>first</i></b>.`

On the other hand, the following:

`<b>Well formed, this is <i>not!</b></i>`

is not well formed.

XHTML strictly enforces other nesting restrictions that have always been part of HTML but have not always been enforced. These restrictions are not formally part of the XHTML DTD; they are instead defined as part of the XHTML standard that is based on the DTD.<sup>[6]</sup>

[6] This is hair-splitting within the XHTML standard. The XML standard has no mechanism to define which tags may not be placed within another tag. SGML, upon which XML is based, does have such a feature, but it was removed from XML to make the language easier to use and implement. As a result, these restrictions are simply listed in an appendix of the XHTML standard instead of being explicitly defined in the XHTML DTD.

Nesting restrictions include:

- The `<a>` tag cannot contain another `<a>` tag.
- The `<pre>` tag cannot contain `<img>`, `<object>`, `<big>`, `<small>`, `<sub>`, or `<sup>` tags.
- The `<button>` tag cannot contain `<input>`, `<select>`, `<textarea>`, `<label>`, `<button>`, `<form>`, `<fieldset>`, `<iframe>`, or `<isindex>` tags.
- The `<label>` tag cannot contain other `<label>` tags.
- The `<form>` tag cannot contain other `<form>` tags.

These restrictions apply to nesting at any level. For example, an `<a>` tag cannot contain any other `<a>` tags, or any tag that in turn contains an `<a>` tag.

## 16.3.2 End Tags

As we've documented throughout this book, any HTML tag that contains other tags or content has a corresponding end tag. However, one of the hallmarks of HTML (codified in the 4.01 standard) is that you may leave out the end tags if their presence can be inferred by the processing agent. This is why most of us HTML authors commonly leave out the `</p>` end tag between adjacent paragraphs. Also, lists and tables can be complicated to wade through, and not having to visually stumble over all the `</li>`, `</td>`, `</th>`, and `</tr>` end tags certainly makes HTML a little clearer and easier to read.

This is not so for XHTML. Every tag that contains other tags or content must have a corresponding end tag present, correctly nested within the XHTML document. A missing end tag is an error and renders the document noncompliant.

## 16.3.3 Handling Empty Elements

In XML, and thus XHTML, every tag must have a corresponding end tag even those that aren't allowed to contain other tags or content. Accordingly, XHTML expects the line break to appear as `<br></br>` in your document. Ugh.

Fortunately, there is an acceptable alternative: include a slash before the closing bracket of the tag to indicate its ending (e.g., `<br />`). If the tag has attributes, the slash comes after all the attributes, so that an image could be defined as:

```

```

While this notation may seem foreign and annoying to an HTML author, it actually serves a useful purpose. Any XHTML element that has no content can be written this way. Thus, an empty paragraph can be written as `<p />`, and an empty table cell can be written as `<td />`. This is a handy way to mark empty table cells.

Clever as it may seem, writing empty tags in this abbreviated way may confuse HTML browsers. To avoid compatibility problems, you can fool the HTML browsers by placing a space before the forward slash in an empty element using the XHTML version of its end tag. For example, use `<br />`, with a space between the `br` and `/`, instead of the XHTML equivalents `<br/>` or `<br></br>`. [Table 16-1](#) contains all of the empty HTML tags, expressed in their acceptable XHTML (transitional DTD) forms.

**Table 16-1. HTML empty tags in XHTML format**

|                                |                             |                                 |
|--------------------------------|-----------------------------|---------------------------------|
| <code>&lt;area /&gt;</code>    | <code>&lt;base /&gt;</code> | <code>&lt;basefont /&gt;</code> |
| <code>&lt;br /&gt;</code>      | <code>&lt;col /&gt;</code>  | <code>&lt;frame /&gt;</code>    |
| <code>&lt;hr /&gt;</code>      | <code>&lt;img /&gt;</code>  | <code>&lt;input /&gt;</code>    |
| <code>&lt;isindex /&gt;</code> | <code>&lt;link /&gt;</code> | <code>&lt;meta /&gt;</code>     |
| <code>&lt;param /&gt;</code>   |                             |                                 |

### 16.3.4 Case-Sensitivity

If you thought getting all those end tags in the right place and cleaning up the occasional nesting error would make writing XHTML documents difficult, hold on to your hat. XHTML is case-sensitive for *all* tag and attribute names. In an XHTML document, `<a>` and `<A>` are different tags; `src` and `SRC` are different attributes, and so are `sRc` and `SrC` ! How forgiving HTML seems now.

The XHTML DTD defines all former HTML tags and attributes using lowercase letters. Uppercase tag or attribute names are not valid XHTML tags or attributes.

This can be a difficult situation for any author wishing to convert existing HTML documents into XHTML-compliant ones. Lots of web pages use uppercase tag and attribute names, to make them stand out from the surrounding lowercase content.

To become compliant, all those names must be converted to lowercase even the ones you used in your CSS style-sheet definitions. Fortunately, this kind of change is easily accomplished with various editing tools, and XHTML authoring systems should perform the conversion for you.

### 16.3.5 Quoted Attribute Values

As if all those case-sensitive attribute names weren't aggravating enough, XHTML requires that every attribute value even the numeric ones be enclosed in double quotes. In HTML, you could quote anything your heart desired, but quotes were required only if the attribute value included whitespace or other special characters. To be XHTML-compliant, every attribute must be enclosed in quotes.

For example:

```
<table rows=3>
```

is wrong in XHTML. It is correctly written:

```
<table rows="3">
```

### 16.3.6 Explicit Attribute Values

Within HTML, there are a small number of attributes that have no value. Instead, their mere presence within a tag causes that tag to behave differently. In general, these attributes represent a sort of on/off switch for the tag, like the `compact` attribute for the various list tags or the `ismap` attribute for the `<img>` tag.

In XHTML, every attribute must have a value. Those without values must use their own names. Thus, `compact` in XHTML is correctly specified as

`compact="compact"`, and `checked` becomes `checked="checked"`. Each must contain the required attribute value enclosed in quotes. [Table 16-2](#) contains a list of attributes with the required XHTML values.

**Table 16-2. XHTML values for valueless HTML attributes**

|                                  |                                  |                                  |
|----------------------------------|----------------------------------|----------------------------------|
| <code>checked="checked"</code>   | <code>compact="compact"</code>   | <code>declare="declare"</code>   |
| <code>defer="defer"</code>       | <code>disabled="disabled"</code> | <code>ismap="ismap"</code>       |
| <code>multiple="multiple"</code> | <code>noresize="noresize"</code> | <code>noshade="noshade"</code>   |
| <code>nowrap="nowrap"</code>     | <code>readonly="readonly"</code> | <code>selected="selected"</code> |

Be aware that this attribute value requirement may cause some old HTML browsers to ignore the attribute altogether. HTML 4.0-compliant browsers don't have that problem, so the majority of users won't notice any difference. There is no good solution to this problem, other than distributing HTML 4.0-compliant browsers to the needy.

### 16.3.7 Handling Special Characters

XHTML is more sensitive than HTML to the use of the `<` and `&` characters in JavaScript and CSS declarations within your documents. In HTML, you can avoid potential conflicts by enclosing your scripts and style sheets in comments (`<!--` and `-->`). XML browsers, however, may simply remove all the contents of comments from your document, thereby deleting your hidden scripts and style sheets.

To properly shield your special characters from XML browsers, enclose your styles or scripts in a CDATA section. This tells the XML browser that any characters contained within are plain old characters, without special meanings. For example:

```
<script language="JavaScript">
```

```
<![CDATA[  
    ...JavaScript here...  
]]>  
  
</script>
```

This doesn't solve the problem, though. HTML browsers ignore the contents of the `CDATA` XML tag but honor the contents of comment-enclosed scripts and style sheets, whereas XML browsers do just the opposite. We recommend that you put your scripts and styles in external files and reference them in your document with appropriate external links.

Special characters in attribute values are problematic in XHTML, too. In particular, an ampersand within an attribute value should always be written using `&amp;` and not simply an `&` character. Similarly, play it safe and encode less-than and greater-than signs using their `&lt;` and `&gt;` entities. For example, while:

```
<img src=seasonings.gif alt="Salt & pepper">
```

is perfectly valid HTML, it must be written as:

```

```

to be compliant XHTML.

### 16.3.8 The `id` and `name` Attributes

Early versions of HTML used the `name` attribute with the `<a>` tag to create a fragment identifier in the document. This fragment could then be used in a URL to refer to a particular spot within a document. The `name` attribute was later added to other tags, like `<frame>` and `<img>`, allowing those elements to be referenced by name from other spots in the document.



With HTML 4.0, the W3C added the `id` attribute to almost every tag. Like `name`, `id` lets you associate an identifier with nearly any element in a document for later reference and use, perhaps by a hyperlink or a script.

XHTML has a strong preference for the `id` attribute as the anchor of choice within a document. The `name` attribute is defined but formally deprecated for those elements that have historically used it. With widespread support of HTML 4.0 now in place, you should begin to avoid the `name` attribute where possible and instead use the `id` attribute to bind names to elements in your documents. If you must use the `name` attribute on certain tags, include an identical `id` attribute to ensure that the tag will behave similarly when processed by an XHTML browser.

## 4.1 Divisions and Paragraphs

Like most text processors, a browser wraps the words it finds to fit the horizontal width of its viewing window. Widen the browser's window, and words automatically flow up to fill the wider lines. Squeeze the window, and words wrap downward.

Unlike most text processors, however, HTML and XHTML use explicit division (`<div>`), paragraph (`<p>`), and line-break (`<br>`) tags to control the alignment and flow of text. Return characters, although quite useful for readability of the source document, typically are ignored by the browser authors must use the `<br>` tag to explicitly force a common text line break. The `<p>` tag, while also causing a line break, carries with it meaning and effects beyond a simple return.

The `<div>` tag is a little different. Originally codified in the HTML 3.2 standard, `<div>` was included in the language to be a simple organizational tool to divide the document into discrete sections whose somewhat obtuse meaning meant few authors used it. But recent innovations (alignment, styles, and the `id` attribute for document referencing and automation) now let you more distinctly label and thereby define individual sections of your documents, as well as control the alignment and appearance of those sections. These features breathe real life and meaning into the `<div>` tag.

By associating an `id` and a `class` name with the various sections of your document, each delimited by a `<div id=name class=name>` tag and attributes (you can do the same with other tags, like `<p>`, too), you not only label those divisions for later reference by a hyperlink and for automated processing and management (collecting all the bibliography divisions, for instance), but you may also define different, distinct display styles for those portions of your document. For instance, you might define one divisional class for your document's abstract (`<div class=abstract>`, for example), another for the body, a third for the conclusion, and a fourth divisional class for the bibliography (`<div`

`class=biblio>`, for example).

Each class, then, might be given a different display definition in a document-level or externally related style sheet: for example, the abstract indented and in an italic typeface (such as `div.abstract {left-margin: +0.5in; font-style: italic}`);, the body in a left-justified roman typeface, the conclusion similar to the abstract, and the bibliography automatically numbered and formatted appropriately.

We provide a detailed description of style sheets, classes, and their applications in [Chapter 8](#).

### 4.1.1 The `<div>` Tag

As defined in the HTML 4.01 and XHTML 1.0 standards, the `<div>` tag divides your document into separate, distinct sections. It may be used strictly as an organizational tool, without any sort of formatting associated with it, but it becomes more effective if you add the `id` and `class` attributes to label the divisions. The `<div>` tag may also be combined with the `align` attribute to control the alignment of whole sections of your document's content in the display and with the many programmatic "on" attributes for user interaction.

## <div>

### *Function*

Defines a block of text

### *Attributes*

`align, class, dir, id, lang, nowrap, onClick, onDbClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, style, title`

### *End tag*

`</div>`; usually omitted in HTML

### *Contains*

*body\_content*

### *Used in*

*block*

#### 4.1.1.1 The align attribute

The `align` attribute for `<div>` positions the enclosed content to either the `left` (default), `center`, or `right` of the display. In addition, you can specify `justify` to align both the left and right margins of the text. The `<div>` tag may be nested, and the alignment of the nested `<div>` tag takes precedence over the containing `<div>` tag. Further, other nested alignment tags, such as `<center>`, aligned paragraphs (see `<p>` in [Section 4.1.2](#)), or specially aligned table rows and cells override the effect of `<div>`. Like the `align` attribute for other tags, it is deprecated in the HTML and XHTML standards in deference to style sheet-based layout controls.

#### 4.1.1.2 The nowrap attribute

Supported only by Internet Explorer, the `nowrap` attribute suppresses automatic word wrapping of the text within the division. Line breaks will occur only where you have placed carriage returns in your source document.

While the `nowrap` attribute probably doesn't make much sense for large sections of text that would otherwise be flowed together on the page, it can make things a bit easier when creating blocks of text with many explicit line breaks: poetry, for example, or addresses. You don't have to insert all those explicit `<br>` tags in a text flow within a `<div nowrap>` tag. On the other hand, all other browsers ignore the `nowrap` attribute and merrily flow your text together anyway. If you are targeting only Internet Explorer with your documents, consider using `nowrap` where needed, but otherwise, we can't recommend this attribute for general use.

#### 4.1.1.3 The dir and lang attributes

The `dir` attribute lets you advise the browser which direction the text should be displayed in, and the `lang` attribute lets you specify the language used within the division. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)]

#### 4.1.1.4 The id attribute

Use the `id` attribute to label the document division specially for later reference by a hyperlink, style sheet, applet, or other automated process. An acceptable `id` value is any quote-enclosed string that uniquely identifies the division and that later can be used to reference that document section unambiguously. Although we're introducing it within the context of the `<div>` tag, this attribute can be used with almost any tag.

When used as an element label, the value of the `id` attribute can be

added to a URL to address the labelled element uniquely within the document. You can label both large portions of content (via a tag like `<div>`) and small snippets of text (using a tag like `<i>` or `<span>`). For example, you might label the abstract of a technical report using `<div id="abstract">`. A URL could jump right to that abstract by referencing `report.html#abstract`. When used in this manner, the value of the `id` attribute must be unique with respect to all other `id` attributes within the document and all the names defined by any `<a>` tags with the `name` attribute. [[Section 6.3.3](#)]

When used as a style-sheet selector, the value of the `id` attribute is the name of a style rule that can be associated with the current tag. This provides a second set of definable style rules, similar to the various style classes you can create. A tag can use both the `class` and `id` attributes to apply two different rules to a single tag. In this usage, the name associated with the `id` attribute must be unique with respect to all other style IDs within the current document. A more complete description of style classes and IDs can be found in [Chapter 8](#).

#### 4.1.1.5 The title attribute

Use the optional `title` attribute and quote-enclosed string value to associate a descriptive phrase with the division. Like the `id` attribute, the `title` attribute can be used with almost any tag and behaves similarly for all tags.

There is no defined usage for the value of the `title` attribute, and many browsers simply ignore it. Internet Explorer, however, will display the title associated with any element when the mouse pauses over that element. Used correctly, the `title` attribute could be used in this manner to provide spot help for the various elements within your document.

#### 4.1.1.6 The class and style attributes

Use the `style` attribute with the `<div>` tag to create an inline style for the content enclosed by the tag. The `class` attribute lets you apply the style of a predefined class of the `<div>` tag to the contents of this division. The value of the `class` attribute is the name of a style defined in some document-level or externally defined style sheet. In addition, class-identified divisions lend themselves well for computer processing of your documents; for example, extracting all divisions with the class name "biblio," for the automated assembly of a master bibliography. [[Section 8.1.1](#)] [[Section 8.3](#)]

#### 4.1.1.7 Event attributes

The many user-related events that may happen in and around a division, such as when a user clicks or double-clicks the mouse within its display space, are recognized by the browser if it conforms to the current HTML or XHTML standard. With the respective "on" attribute and value, you may react to those events by displaying a user dialog box or activating some multimedia event. [[Section 12.3.3](#)]

### 4.1.2 The `<p>` Tag

The `<p>` tag signals the start of a paragraph. That's not well known even by some veteran webmasters, because it runs counterintuitive to what we've come to expect from experience. Most word processors we're familiar with use just one special character, typically the return character, to signal the *end* of a paragraph. In HTML and XHTML, each paragraph should start with `<p>` and end with the corresponding `</p>` tag. And while a sequence of newline characters in a text processor-displayed document creates an empty paragraph for each one, browsers typically ignore all but the first paragraph tag.

In practice, with HTML you can ignore the starting `<p>` tag at the beginning of the first paragraph and the `</p>` tags at the ends of each paragraph: they can be implied from other tags that occur in the document and hence safely omitted.<sup>[1]</sup> For example:

[1] XHTML, on the other hand, requires explicit starting and ending tags.

```
<body>
```

```
This is the first paragraph, at the very beginning of  
this document.
```

```
<p>
```

```
The tag above signals the start of this second paragraph.  
by a browser, it will begin slightly below the end of  
with a bit of extra whitespace between the two paragraphs.
```

```
<p>
```

```
This is the last paragraph in the example.
```

```
</body>
```

**Notice that we haven't included the paragraph start tag (`<p>`) for the first paragraph or any end paragraph tags; they can be unambiguously inferred by the browser and are therefore unnecessary.**



**<p>**

*Function*

Defines a paragraph of text

*Attributes*

`align, class, dir, id, lang, onDb1Click, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, style, title`

*End tag*

`</p>`; often omitted in HTML

*Contains*

*text*

*Used in*

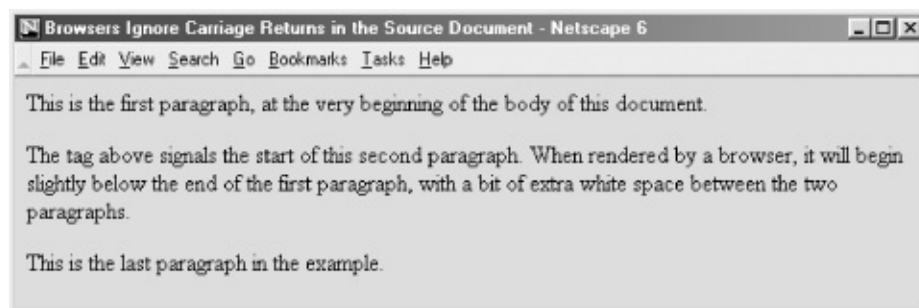
*block*

In general, you'll find that human document authors tend to omit postulated tags whenever possible, while automatic document generators tend to insert them. That may be because the software designers didn't want to run the risk of having their products chided by competitors as not adhering to the HTML standard, even though we're splitting letter-of-the-law hairs here. Go ahead and be defiant: omit that first paragraph's `<p>` tag and don't give a second thought to paragraph-ending `</p>` tags provided, of course, that your document's structure and clarity are not compromised (that is, as long as you are aware that XHTML frowns severely on such laxity).

#### 4.1.2.1 Paragraph rendering

When encountering a new paragraph (`<p>`) tag, the browser typically inserts one blank line plus some extra vertical space into the display before starting the new paragraph. The browser then collects all the words and, if present, inline images into the new paragraph, ignoring leading and trailing spaces (not spaces between words, of course) and return characters in the source text. The browser software then flows the resulting sequence of words and images into a paragraph that fits within the margins of its display window, automatically generating line breaks as needed to wrap the text within the window. For example, compare how a browser arranges the text into lines and paragraphs ([Figure 4-1](#)) to how the preceding example is printed on the page. The browser may also automatically hyphenate long words, and the paragraph may be full-justified to stretch the line of words out toward both margins.

**Figure 4-1. Browsers ignore common return characters in the source HTML/XHTML document**



The net result is that you do not have to worry about line length, word wrap, and line breaks when composing your documents. The browser will take any arbitrary sequence of words and images and display a nicely formatted paragraph.

If you want to control line length and breaks explicitly, consider using a preformatted text block with the `<pre>` tag. If you need to force a line break, use the `<br>` tag. [[<pre>](#)] [[Section 4.6.1](#)]

#### **4.1.2.2 The align attribute**

Most browsers automatically left-justify a new paragraph. To change this behavior, HTML 4 and XHTML give you the `align` attribute for the `<p>` tag and provide four kinds of content justification: `left`, `right`, `center`, or `justify`.

**Figure 4-2** shows you the effect of various alignments as rendered from the following source:

```
<p align=right>
```

Right over here!

```
<br>
```

This is too.

```
<p align=left>
```

Slide back left.

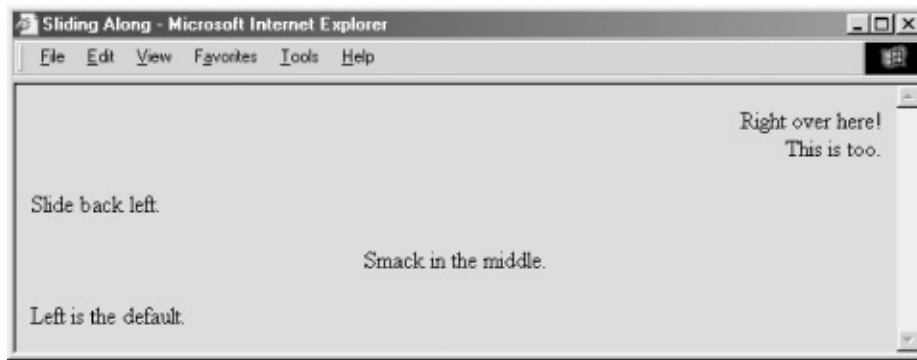
```
<p align=center>
```

Smack in the middle.

```
</p>
```

Left is the default.

**Figure 4-2. Effect of the align attribute on paragraph justification**



Notice in the HTML example that the paragraph alignment remains in effect until the browser encounters another `<p>` tag or an ending `</p>` tag. We deliberately left out a final `<p>` tag in the example to illustrate the effects of the `</p>` end tag on paragraph justification. Other body elements including forms, headers, tables, and most other body content-related tags may also disrupt the current paragraph alignment and cause subsequent paragraphs to revert to the default left alignment.

Note that the `align` attribute is deprecated in HTML 4 and XHTML, in deference to style sheet-based alignments.

#### 4.1.2.3 The `dir` and `lang` attributes

The `dir` attribute lets you advise the browser which direction the text within the paragraph should be displayed in, and the `lang` attribute lets you specify the language used within that paragraph. The `dir` and `lang` attributes are supported by the popular browsers, even though there are no behaviors defined for any specific language. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)]

#### 4.1.2.4 The `class`, `id`, `style`, and `title` attributes

Use the `id` attribute to create a label for the paragraph that can later be used to unambiguously reference that paragraph in a hyperlink target, for automated searches, as a style-sheet selector, and with a host of other applications. [[Section 4.1.1.4](#)]

Use the optional `title` attribute and quote-enclosed string value to provide a descriptive phrase for the paragraph. [[Section 4.1.1.4](#)]

Use the `style` attribute with the `<p>` tag to create an inline style for the paragraph's contents. The `class` attribute lets you label the paragraph with a name that refers to a predefined class of the `<p>` tag previously declared in some document-level or externally defined style sheet. Class-identified paragraphs lend themselves well to computer processing of your documents for example, extracting all paragraphs whose class name is "citation," for automated assembly of a master list of citations. [[Section 8.1.1](#)] [[Section 8.3](#)]

#### 4.1.2.5 Event attributes

As with divisions, there are many user-initiated events, such as when a user clicks or double-clicks within a tag's display space, that are recognized by the browser if it conforms to the current HTML or XHTML standard. With the respective "on" attribute and value, you may react to those events by displaying a user dialog box or activating some multimedia event. [[Section 12.3.3](#)]

#### 4.1.2.6 Allowed paragraph content

A paragraph may contain any element allowed in a text flow, including conventional words and punctuation, links (`<a>`), images (`<img>`), line breaks (`<br>`), font changes (`<b>`, `<i>`, `<tt>`, `<u>`, `<strike>`, `<big>`, `<small>`, `<sup>`, `<sub>`, and `<font>`), and content-based style changes (`<acronym>`, `<cite>`, `<code>`, `<dfn>`, `<em>`, `<kbd>`, `<samp>`, `<strong>`, and `<var>`). If any other element occurs within the paragraph, it implies that the paragraph has ended, and the browser assumes that the closing `</p>` tag was not specified.

#### 4.1.2.7 Allowed paragraph usage

You may specify a paragraph only within a *block*, along with other paragraphs, lists, forms, and preformatted text. In general, this means that paragraphs can appear where a flow of text is appropriate, such as in the body of a document, in an element in a list, and so on. Technically, paragraphs cannot appear within a header, anchor, or other element whose content is strictly text-only. In practice, most browsers ignore this restriction and format the paragraph as a part of the containing element.

## 8.1 The Elements of Styles

At the simplest level, a style is nothing more than a rule that tells the browser how to render a particular HTML or XHTML tag's contents.<sup>[2]</sup> Each tag has a number of style properties associated with it, whose values define how that tag is rendered by the browser. A rule defines a specific value for one or more properties of a tag. For example, most tags have a `color` property, the value of which defines the color in which Netscape or Internet Explorer should display the contents of the tag. Other properties include fonts, line spacing, margins, borders, sound volume, and voice, which we describe in detail later in this chapter.

[2] We explicitly avoided the term "display" here because it connotes visual presentation, whereas the CSS2 standard works hard to suggest many different ways of presenting the tagged contents of a document.

There are three ways to attach a style to a tag: inline, on the document level, or through the use of an external style sheet. You may use one or more style sheets for your documents. The browser either merges the style definitions from each style or redefines the style characteristic for a tag's contents. Styles from these various sources are applied to your document, combining and defining style properties that cascade from external style sheets through local document styles, ending with inline styles. This cascade of properties and style rules gives rise to the standard's name: Cascading Style Sheets.

We cover the syntactic basics of the three style-sheet techniques here. We delve more deeply into the appropriate use of inline, document-level, and external style sheets at the end of this chapter.

### 8.1.1 Inline Styles: The style Attribute

The inline style is the simplest way to attach a style to a tag just include a `style` attribute with the tag along with a list of properties and their values. The browser uses those style properties and values to render the

contents of that tag.

For instance, the following style tells the browser to display the level-1 header text, "I'm so bluuuuuoooo!", not only in the `<h1>` tag style, but also colored blue and italicized:

```
<h1 style="color: blue; font-style: italic">I'm so blu
```

Inline styles can be difficult to maintain, because they add more contents to their tags' definitions, making them harder to read. Also, because they have only a local effect, they must be sprinkled throughout your document. Use the inline `style` attribute sparingly and only in those rare circumstances when you cannot achieve the same effects otherwise.

## 8.1.2 Document-Level Style Sheets

The real power of style sheets becomes more evident when you place a list of presentation rules at the beginning of your HTML or XHTML document. Placed within the `<head>` and enclosed within their own `<style>` and `</style>` tags, document-level style sheets affect all the same tags within that document, except for tags that contain overriding inline `style` attributes.<sup>[3]</sup>

[3] XHTML-based document-level style sheets are specially enclosed in CDATA sections of your documents. See [Section 16.3.7](#) in [Chapter 16](#) for details.



## **<style>**

### *Function*

Defines a document-level style sheet

### *Attributes*

`dir, lang, media, title, type`

### *End tag*

`</style>`; rarely omitted in HTML

### *Contains*

*styles*

### *Used in*

*head\_content*

Everything between the `<style>` and `</style>` tags is considered part of the style rules that the browser is to apply when rendering the document. Actually, the contents of the `<style>` tag are not HTML or XHTML and are not bound by the normal rules for markup content. The `<style>` tag, in effect, lets you insert foreign content into your document that the browser uses to format your tags.

For example, a styles-conscious browser displays the contents of all `<h1>` tags as blue, italic text in an HTML document that has the following document-level style sheet definition in its head:

```
<head>
```

```
<title>All True Blue</title>
```

```
<style type="text/css">
```

```
<!--  
  
/* make all level-1 headers blue */  
  
h1 {color: blue; font-style: italic}  
  
-->  
  
</style>  
  
</head>  
  
<body>  
  
<h1>I'm so bluuuuuoooo!</h1>  
  
...  
  
<h1>I am ba-loooooo, tooooo!</h1>
```

### 8.1.2.1 The type attribute

There are other types of style sheets available for HTML/XHTML besides CSS. Like the JavaScript style sheets we describe in [Chapter 12](#), they are not well supported, if at all, by the popular browsers, so we don't spend a lot of time on them in this book. Nonetheless, the browser needs a way to distinguish which style sheet you use in your document. Use the `type` attribute within the `<style>` tag for that. Cascading style sheets are all `type text/css`; JavaScript style sheets use the `type text/javascript`. You may omit the `type` attribute and hope the browser figures out the kind of styles you are using, but we suggest you always include the `type` attribute, so there is no opportunity for confusion. [[Section 12.4](#)]

### 8.1.2.2 The media attribute

HTML and XHTML documents can wind up in the strangest places these days, such as on cellular phones. To help the browser figure out the best way to render your documents, include the `media` attribute within the `<style>` tag. The value of this attribute is the document's intended medium, although it doesn't preclude rendering by other media. The default value is `screen` (computer display). Other values include `tty` (text only), `tv` (television), `projection` (theaters), `handheld` (PDAs and cell phones), `print` (ink on paper), `braille` (tactile devices), `embossed` (Braille printers), `aural` (audio; speech synthesis, for instance), and `all` (many different types of media).

If you want to explicitly list several types of media, rather than specifying `all`, use a quote-enclosed, comma-separated list of media types as the value of the `media` attribute. For example:

```
<style type="text/css" media="screen,print">
```

tells the browser that your document contains CSS both for printing and for computer displays.

Take caution when specifying media, because the browser cannot apply the styles you define unless the document is being rendered on one of your specified media. Thus, the browser would not apply our example set of styles designed for `media="screen,print"` if the user is, for instance, connected to the Web with a handheld computer.

How do you create different style definitions for different media without creating multiple copies of your document? The CSS2 standard lets you define media-specific style sheets through its extension to the `@import` at-rule and through the `@media` at-rule, which we describe in [Section 8.1.4](#).

### 8.1.2.3 The dir, lang, and title attributes

As with any HTML/XHTML element, you can associate a descriptive title with the `<style>` tag. If the browser displays this title to the user, it uses the values of the `dir` and `lang` attributes to render it correctly. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)] [[Section 4.1.1.4](#)]

### 8.1.3 External Style Sheets

You can also place style definitions into a separate document (a text file with the MIME type of `text/css`) and import this "external" style sheet into your document. The same style sheet can actually be used for multiple documents. Because an external style sheet is a separate file and is loaded by the browser over the network, you can store it anywhere, reuse it often, and even use others' style sheets. But most importantly, external style sheets give you the power to influence the display styles of all related tags not only in a single document but in an entire collection of documents.

For example, suppose we create a file named *gen\_styles.css* containing the following style rule:

```
h1 {color: blue; font-style: italic}
```

For each and every one of the documents in our collections, we can tell the browser to read the contents of the *gen\_styles.css* file, which in turn colors all the `<h1>` tag contents blue and renders the text in italic. Of course, that is true only if the user's machine is capable of these style tricks, she's using a styles-conscious browser such as Netscape or Internet Explorer, and the style isn't overridden by a document-level or inline style definition.

You can load external style sheets into your document in two different ways: linking them or importing them.

#### 8.1.3.1 Linked external style sheets

One way to load an external style sheet is to use the `<link>` tag within

the `<head>` of your document:

```
<head>

<title>Style linked</title>

<link rel=stylesheet type="text/css"

        href="http://www.kumquats.com/styles/gen_styles.

        title="The blues">

</head>

<body>

<h1>I'm so bluuuuooooo!</h1>

...

<h1> I am ba-loooooo, toooooo!</h1>
```

Recall that the `<link>` tag creates a relationship between the current document and some other document on the Web. In this example, we tell the browser that the document named in the `href` attribute is a cascading style sheet (`css`), as indicated by the `type` attribute. These two attributes are required. We also explicitly tell the browser that the file's relationship to our document is that it is a `stylesheet` and we provide a `title` making it available for later reference by the browser. [\[Section 6.7.2\]](#)

The style sheet-specifying `<link>` tag and its required `href` and `type` attributes must appear in the `<head>` of a document. The URL of the style sheet may be absolute or relative to the document's base URL.

### 8.1.3.2 Imported external style sheets

The second technique for loading an external style sheet imports the file with a special command (a.k.a. at-rule) within the `<style>` tag:

```
<head>

<title>Imported style sheet</title>

<style type="text/css">

    <!--

        @import url(http://www.kumquats.com/styles/gen_sty

        @import "http://www.kumquats.com/styles/spec_style

        body {background: url(backgrounds/marble.gif)}

    -->

</style>

</head>
```

The `@import` at-rule expects a single URL for the network path to the external style sheet. As shown in this example, the URL may be either a string enclosed in double quotes and ending with a semicolon or the contents of the `url` keyword, enclosed in parentheses, with a trailing semicolon. The URL may be absolute or relative to the document's base URL.

The `@import` at-rule must appear *before* any conventional style rules, either in the `<style>` tag or in an external style sheet. Otherwise, the standard insists that the browser ignore the errant `@import`. By first importing all the various style sheets, then processing document-level style rules, the CSS2 standard cascades: the last one standing wins. [\[Section 8.4.1.4\]](#)

The `@import` at-rule can appear in a document-level style definition or even in another external style sheet, letting you create nested style sheets.

### 8.1.4 Media-Specific Styles

Besides the `media` attribute for the `<style>` tag, the CSS2 standard has two other features that let you apply different style sheets, depending on the agent or device that renders your document. This way, for instance, you can have one style or whole style sheet take effect when your document gets rendered on a computer screen and another set of styles for when the contents get punched out on a Braille printer. And what about those cell phones that access the Web?

Like the `media` attribute for the `<style>` tag that affects the entire style sheet, you can specify whether the user's document processor loads and uses an imported style sheet. Do that by adding a media-type keyword or a series of comma-separated keywords to the end of the `@import` at-rule. For instance, the following example lets the user agent decide whether to import and use the speech-synthesis style sheet or a common PC-display and print style sheet, if it is able to render the specified media types:

```
@import url(http://www.kumquats.com/styles/visual_style.  
@import "http://www.kumquats.com/styles/speech_styles."
```

The `@import` CSS2 media types are the same as those for the `<style>` tag's `media` attribute, including `all`, `aural`, `braille`, `embossed`, `handheld`, `print`, `projection`, `screen`, `tty`, and `tv`.

Another CSS2 way to select media is through the explicit `@media` at-rule, which lets you include media-specific rules within the same style sheet, either at the document level or in an external style sheet. At the document level, like `@import`, the `@media` at-rule must appear within the contents of the `<style>` tag. The at-rules may not appear within

another rule. Unlike `@import`, `@media` may appear subsequent to other style rules, and its style-rule contents override previous rules according to the cascading standard.

The contents of `@media` include one or more comma-separated media-type keywords followed by a curly brace (`{}`)-enclosed set of style rules. For example:

```
body {background: white}

@media tv, projection {

    body {background: lt_blue}

}
```

The `lt_blue` attribute to the `@media` at-rule causes the body's background color to display light blue, rather than the default white set in the general style rule, when the document is rendered on a television or projection screen (as specified by the `tv` and `projection` attributes).

### 8.1.5 Linked Versus Imported Style Sheets

At first glance, it may appear that linked and imported style sheets are equivalent, using different syntax for the same functionality. This is true if you use just one `<link>` tag in your document. However, special CSS2-standard rules come into play if you include two or more `<link>` tags within a single document.

With one `<link>` tag, the browser should load the styles in the referenced style sheet and format the document accordingly, with any document-level and inline styles overriding the external definitions. With two or more `<link>` tags, the browser should present the user with a list of all the linked style sheets. The user then selects one of the linked sheets, which the browser loads and uses to format the document; the other linked style sheets get ignored.



On the other hand, the styles-conscious browser merges, as opposed to separating, multiple `@import`d style sheets to form a single set of style rules for your document. The last imported style sheet takes precedence if there are duplicate definitions among the style sheets. Hence, if the external *gen\_styles.css* style sheet specification first tells the browser to make `<h1>` contents blue and italic, and then a later *spec\_styles.css* tells the browser to make `<h1>` text red, then the `<h1>` tag contents appear red and italic. And if we later define another color say, yellow for `<h1>` tags in a document-level style definition, the `<h1>` tags are all yellow and italic. Cascading effects. See?

In practice, the popular browsers treat linked style sheets just like imported ones by cascading their effects. The browsers do not currently let you choose from among linked choices. Imported styles override linked external styles, just as the document-level and inline styles override external style definitions. To bring this all together, consider the example:

```
<html>

<head>

<link rel=stylesheet href=sheet1.css type=text/css>

<link rel=stylesheet href=sheet2.css type=text/css>

<style>

<!--

    @import url(sheet3.css);

    @import url(sheet4.css);

-->

</style>
```

`</head>`

Using the CSS2 model, the browser should prompt the user to choose *sheet1.css* or *sheet2.css*. It should then load the selected sheet, followed by *sheet3.css* and *sheet4.css*. Duplicate styles defined in *sheet3.css* or *sheet4.css*, and in any inline styles, override styles defined in the selected sheet. In practice, the popular browsers cascade the style-sheet rules as defined in the example order *sheet1* through *sheet4*.

### 8.1.6 Limitations of Current Browsers

Internet Explorer and Netscape support the `<link>` tag to apply an external style sheet to a document. Neither Netscape nor Internet Explorer supports multiple, user-selectable `<link>` style sheets, as proposed by the CSS2 standard. Instead, they treat the `<link>` style sheets as they do `@import` or document-level styles, by cascading the rules.

Netscape Version 6, but not earlier versions, and Internet Explorer Versions 5 and later honor the `@import` as well as the `@media` at-rules, for both document-level and external sheets, allowing sheets to be nested.

Achieving media-specific styles through external style sheets with earlier Netscape browsers is hopeless. Assume, therefore, that most people who have Netscape Version 4 will render your documents on a common PC screen, so make that medium the default. Then embed all other media-specific styles, such as those for print or Braille, within `@media` at-rules, so that Internet Explorer and other CSS-compliant agents properly select styles based on the rendering medium. The only other alternative is to create media-specific `<style>` tags within each document. Run, do not walk, away from that idea.

### 8.1.7 Style Comments

Comments are welcome inside the `<style>` tag and in external style sheets, but don't use standard HTML comments; style sheets aren't HTML. Rather, enclose style comments between `/*` and `*/` markers, as we did in the example in [Section 8.1.2](#). (Those of you who are familiar with the C programming language will recognize these comment markings.) Use this comment syntax for both document-level and external style sheets. Comments cannot be nested.

We recommend documenting your styles whenever possible, especially in external style sheets. Whenever the possibility exists that your styles may be used by other authors, comments make it much easier to understand your styles.

### 8.1.8 Handling Styleless Browsers

We have to do some fancy footwork to allow our HTML documents to work with both older, styleless browsers and newer, styles-conscious browsers. The order of the tags is very important. Here's the approach, which you may have noticed in our document-level style examples:

```
<style>

<!--

    @import url(sheet3.css);

    @import url(sheet4.css);

-->

</style>
```

First, we use a `<style>` tag, followed by an HTML comment, followed by our style rules. We close the comment, and we close the `</style>` tag.

Newer browsers ignore HTML comments within `<style>` tags, so these browsers implement our styles correctly. Older browsers ignore what is

placed between HTML comments, so they ignore our style rules (which they would otherwise print on the screen, to the confusion of the user).

XHTML documents require a slightly different approach. In those documents, we enclose document-level styles in a CDATA section instead of in HTML comments. See [Section 16.3.7](#) for details.

In the style sheets themselves, use style comments rather than HTML comments. The styleless browsers won't load the style sheets, and newer browsers interpret them correctly.

## 8.1.9 Style Precedence

You may import multiple external style sheets and combine them with document-level and inline style effects in many different ways. Their effects cascade (hence the name, of course). You may specify the font type for our example `<h1>` tag, for instance, in an external style definition, whereas its color may come from a document-level style sheet.

Style-sheet effects are not cumulative, however: of the many styles that may define different values for the same property colors for the contents of our example tag, for instance the one that takes precedence can be found by following these rules, listed here in order:

### *Sort by origin*

A style defined "closer" to a tag takes precedence over a more "distant" style; an inline style takes precedence over a document-level style, which takes precedence over the effects of an external style.

### *If more than one applicable style exists, sort by class*

A property defined as a class of a tag (see [Section 8.3](#)) takes precedence over a property defined for the tag in general.

### *If multiple styles still exist, sort by specificity*

The properties for a more specific contextual style (see [Section 8.2.3](#)) take precedence over properties defined for a less specific context.

*If multiple styles still exist, sort by order*

The property specified latest takes precedence.

The relationship between style properties and conventional tag attributes is almost impossible to predict. Style sheet-dictated background and foreground colors whether defined externally, at the document level, or inline override the various `color` attributes that may appear within a tag. But the `align` attribute of an inline image usually takes precedence over a style-dictated alignment.

There is an overwhelming myriad of style and tag presentation-attribute combinations. You need a crystal ball to predict which combination wins and which loses the precedence battle. The rules of redundancy and style versus attribute precedence are elucidated in the W3C CSS2 standard, but no clear pattern of precedence is implemented in the styles-conscious browsers. This is particularly unfortunate because there will be an extended period, perhaps several years, in which users may or may not use styles-conscious browsers. Authors must implement both styles and non-style presentation controls to achieve the same effects.

Nonetheless, our recommendation is to run as fast as you can away from one-shot, inline, localized kinds of presentation effects such as those afforded by the `<font>` tag and `color` attribute. They have served their temporary purpose; it's now time to bring consistency (without the pain!) back into your document presentation. Use styles. It's the HTML way.

## 6.7 Relationships

Very few documents stand alone. Instead, a document is usually part of a collection of documents, each connected by one or several of the hypertext strands we describe in this chapter. One document may be a part of several collections, linking to some documents and being linked to by others. Readers move between the document families as they follow the links that interest them.

When you link two documents, you establish an explicit relationship between them. Conscientious authors use the `rel` attribute of the `<a>` tag to indicate the nature of the link. In addition, two other tags may be used within a document to further clarify the location of a document within a document family and its relationship to the other documents in that family. These tags, `<base>` and `<link>`, are placed within the body of the `<head>` tag. [`<head>`]

### 6.7.1 The `<base>` Header Element

As we previously explained, URLs within a document can be either absolute (with every element of the URL explicitly provided by the author) or relative (with certain elements of the URL omitted and supplied by the browser). Normally, the browser fills in the blanks of a relative URL by drawing the missing pieces from the URL of the current document. You can change that with the `<base>` tag.

## **<base>**

### *Function*

Defines the base URL for other anchors in the document

### *Attributes*

`href`, `target`

### *End tag*

None in HTML; `</base>` or `<base ... />` in XHTML

### *Contains*

Nothing

### *Used in*

*head\_content*

The `<base>` tag should appear only in the document header, not in its body contents. The browser thereafter uses the specified base URL, not the current document's URL, to resolve all relative URLs, including those found in `<a>`, `<img>`, `<link>`, and `<form>` tags. It also defines the URL that will be used to resolve queries in searchable documents containing the `<isindex>` tag. [Section 6.2]

### **6.7.1.1 The href attribute**

The `href` attribute must have a valid URL as its value, which the browser then uses to define the absolute URL against which relative URLs are based within the document.

For example, the `<base>` tag in this XHTML document head:

```
<head>

<base href="http://www.kumquat.com/" />

</head>

...
```

tells the browser that any relative URLs within this document are relative to the top-level document directory on *www.kumquat.com*, regardless of the address and directory of the machine from which the user retrieved the current document.

Contrary to what you may expect, you can make the base URL relative, not absolute. The browser should (but doesn't always) form an absolute base URL out of this relative URL by filling in the missing pieces with the URL of the document itself. This property can be used to good advantage. For instance, in this next HTML example:

```
<head>

<base href="/info/">

</head>

...
```

the browser makes the `<base>` URL into one relative to the server's */info* directory, which probably is not the same directory of the current document. Imagine if you had to re-address every link in your document with that common directory. Not only does the `<base>` tag help you shorten those URLs in your document that have a common root, it also lets you constrain the directory from which relative references are retrieved without binding the document to a specific server.

### 6.7.1.2 The target attribute



When working with documents inside frames, the `target` attribute with the `<a>` tag ensures that a referenced URL gets loaded into the correct frame. Similarly, the `target` attribute for the `<base>` tag lets you establish the default name of one of the frames or windows in which the browser is to display redirected hyperlinked documents. [Section 11.1]

If you have no other default target for your hyperlinks within your frames, you may want to consider using `<base target=_top>`. This ensures that links that are not specifically targeted to a frame or window will load in the top-level browser window. This eliminates the embarrassing and common error of having references to pages on other sites appear within a frame on your pages, instead of within their own pages. A minor bit of HTML, to be sure, but it makes life much easier for your readers.

### 6.7.1.3 Using `<base>`

The most important reason for using `<base>` is to ensure that any relative URLs within the document will resolve into correct document addresses, even if the documents themselves are moved or renamed. This is particularly important when creating a document collection. By placing the correct `<base>` tag in each document, you can move the entire collection between directories and even servers without breaking all of the links within the documents. You also need to use the `<base>` tag for a searchable document (`<isindex>`) if you want user queries posed to a URL different from that of the host document.

A document that contains both the `<isindex>` tag and other relative URLs may have problems if the relative URLs are not relative to the desired index-processing URL. Since this is usually the case, don't use relative URLs in searchable documents that use the `<base>` tag to specify the query URL for the document.

### 6.7.2 The `<link>` Header Element

Use the `<link>` tag to define the relationship between the current

document and another in a web collection.

## <link>

### Function

Defines a relationship between this document and another document

### Attributes

charset, class, dir, href, hreflang, id, lang, media, onClick, onDoubleClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, rel, rev, style, target, title, type

### End tag

None in HTML; `</link>` or `<link ... />` in XHTML

### Contains

Nothing

### Used in

*head\_content*

The `<link>` tags belongs in the `<head>` content and nowhere else. The attributes of the `<link>` tag are used like those of the `<a>` tag, but their effects serve only to document the relationship between documents. The `<link>` tag has no content, and only XHTML supports the closing `</link>` tag.

### 6.7.2.1 The href attribute

As with its other tag applications, the `href` attribute specifies the URL of the target `<link>` tag. It is a required attribute, and its value is any valid document URL. The specified document is assumed to have a

relationship to the current document.

### 6.7.2.2 The `rel` and `rev` attributes

The `rel` and `rev` attributes express the relationship between the source and target documents. The `rel` attribute specifies the relationship from the source document to the target; the `rev` attribute specifies the relationship from the target document to the source document. Both attributes can be included in a single `<link>` tag.

The value of either attribute is a space-separated list of relationships. The actual relationship names are not specified by the HTML standard, although some have come into common usage, as listed in [Section 6.3.1.5](#). For example, a document that is part of a sequence of documents might use:

```
<link href="part-14.html" rel=next rev=prev>
```

when referencing the next document in the series. The relationship from the source to the target is that of moving to the next document; the reverse relationship is that of moving to the previous document.

### 6.7.2.3 The `title` attribute

The `title` attribute lets you specify the title of the document to which you are linking. This attribute is useful when referencing a resource that does not have a title, such as an image or a non-HTML document. In this case, the browser might use the `<link>` title when displaying the referenced document. For example:

```
<link href="pics/kumquat.gif"
      title="A photograph of the Noble Fruit">
```

tells the browser to use the indicated title when displaying the referenced image.

The value of the attribute is an arbitrary character string, enclosed in quotation marks.

#### 6.7.2.4 The `type` attribute

The `type` attribute provides the MIME content type of the linked document. Supported by both Internet Explorer and Netscape, the HTML 4 and XHTML standard `type` attribute can be used with any linked document. It is often used to define the type of a linked style sheet. In this context, the value of the `type` attribute is usually `text/css`. For example:

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

creates a link to an external style sheet within the `<head>` of a document. See [Chapter 8](#) for details.

#### 6.7.2.5 How browsers might use `<link>`

Although the standards do not require browsers to do anything with the information provided by the `<link>` tag, it's not hard to envision how this information might be used to enhance the presentation of a document.

As a simple example, suppose you consistently provide `<link>` tags for each of your documents that define `next`, `prev`, and `parent` links. A browser could use this information to place at the top or bottom of each document a standard toolbar containing buttons that would jump to the appropriate related document. By relegating the task of providing simple navigational links to the browser, you are free to concentrate on the more important content of your document.

As a more complex example, suppose that a browser expects to find a `<link>` tag defining a glossary for the current document and that this glossary document is itself a searchable document. Whenever a reader clicked on a word or phrase in the document, the browser could

automatically search the glossary for the definition of the selected phrase, presenting the result in a small pop-up window.

As the Web evolves, expect to see more and more uses of the `<link>` tag to define document relationships explicitly.

#### 6.7.2.6 Other `<link>` attributes

The HTML 4 and XHTML standards also include the ubiquitous collection of attributes related to style sheets and user events, and language for the `<link>` tag. You can refer to the corresponding section describing these attributes for the `<a>` tag for a complete description of their usage.

[[Section 6.3.1](#)]

Since you put the `<link>` tag in the `<head>` section, whose contents are not displayed, it may seem that these attributes are useless. It is entirely possible that some future browser may find some way to display the `<link>` information to the user, possibly as a navigation bar or a set of hot-list selections. In those cases, the display and rendering information would prove useful. Currently, no browser provides these capabilities.

## Chapter 9. Forms

Forms, forms, forms, forms: we fill 'em out for nearly everything, from the moment we're born, 'til the moment we die. Pretty mundane, really. So what's to explain all the hoopla and excitement over HTML forms? Simply this: they make HTML and, of course, XHTML truly interactive.

When you think about it, interacting with a web page is basically a lot of button pushing: click here, click there, go here, go there there's no real user feedback, and it's certainly not personalized. Programs like applets, servlets, JSPs, and ASPs provide extensive user-interaction capability but can be difficult to write. Forms, on the other hand, are easily made in HTML/XHTML and make it possible to create documents that collect and process user input and to formulate personalized replies.

This powerful mechanism has far-reaching implications, particularly for electronic commerce. It finishes an online catalog by giving buyers a way to immediately order products and services. It gives nonprofit organizations a way to sign up new members. It lets market researchers collect user data. It gives you an automated way to interact with your readers.

Mull over the ways you might want to interact with your readers while we take a look at both the client- and server-side details of creating forms.

## 12.3 JavaScript

All the executable content elements we've discussed so far have had one common trait: they are separate from the browser and the HTML/XHTML document — separate data, separate execution engine.

JavaScript is different. It is a scripting language that taps the native functionality of the browser. You may sprinkle JavaScript statements throughout your documents, either as blocks of code or single statements attached to individual tags. The JavaScript-enabled browsers, including both Netscape and Internet Explorer, interpret and act upon the JavaScript statements you provide to do such things as alter the appearance of the document, control the display, validate and manipulate form elements, and perform general computational tasks.

As with Java, we do not pretend to teach JavaScript programming in this book. We'll show you how to embed and execute JavaScript within your documents, but we ask that you turn to books like *JavaScript: The Definitive Guide* (O'Reilly) for a complete reference.

### 12.3.1 The `<script>` Tag

One way to place JavaScript code in your document is via the HTML and XHTML standard `<script>` tag.

Everything between `<script>` and `</script>` is processed by the browser as executable JavaScript statements and data. You cannot place HTML or XHTML within this tag; it is flagged as an error by the browser.

However, browsers that do not support `<script>` process its contents as regular HTML, to the confusion of the user. For this reason, we recommend that you include the contents of the `<script>` tag inside HTML comments:

```
<script language="JavaScript">
```



```
<!--  
  
    JavaScript statements go here  
  
// -->  
  
</script>
```

For browsers that ignore the `<script>` tag, the contents are masked by the comment delimiters `<!--` and `-->`. JavaScript-enabled browsers, on the other hand, automatically recognize and interpret the JavaScript statements delimited by the comment tags. By using this skeleton for all your `<script>` tags, you can be sure that all browsers handle your document gracefully, if not completely.

Unfortunately, as we discuss in [Chapter 16](#), script content for XHTML documents must be within a special CDATA declaration, rather than within comments. Hence, HTML browsers won't honor XHTML scripts, and vice versa. Our only recommendation at this point is to follow the popular browsers: write in HTML, but use as many of the features of XHTML as you can in preparation for the future.

You may include more than one `<script>` tag in a document, located in either the `<head>` or the `<body>`. The JavaScript-enabled browser executes the statements in order. Variables and functions defined within one `<script>` tag may be referenced by JavaScript statements in other `<script>` tags. In fact, one common JavaScript programming style is to use a single `<script>` in the document `<head>` to define common functions and global variables for the document and then to call those functions and reference their variables in other JavaScript statements sprinkled throughout the document.

## **<script>**

### *Function:*

Defines an executable script within a document

### *Attributes:*

`charset, defer, language, src, type`

### *End tag:*

`</script>`; never omitted

### *Contains:*

*scripts*

### *Used in:*

*head\_content, body\_content*

### **12.3.1.1 The language and type attributes**

Use the `language` or `type` attribute in the `<script>` tag to declare the scripting language that you used to compose the contents of the tag. The `language` attribute is deprecated by the HTML 4 and XHTML standards in favor of the `type` attribute. Regrettably, the value for each attribute is different.

If you are using JavaScript by far the most common scripting language on the Web use `language="JavaScript"` or `type="text/javascript"`. You may occasionally see the `language` value `VBScript` (`text/vbscript` for `type`), indicating that the enclosed code is written in Microsoft's Visual Basic Script.

With JavaScript, you may also use the language value `"JavaScript 1.2"`, indicating that the enclosed script is written for browsers that support Version 1.2 of the language (most current browsers do). Versioning can be a problem, but it's not too severe. Netscape 2.0, for instance, supports JavaScript 1.0 but does not process scripts identified as `"JavaScript 1.1"`. Then again, what proportion of your audience is running Netscape 2.0?

### 12.3.1.2 The `src` and `charset` attributes

For particularly large JavaScript programs and ones you reuse often, you should store the code in a separate file. In these cases, have the browser load that separate file through the `src` attribute. The value of the `src` attribute is the URL of the file containing the JavaScript program. The stored file should have a MIME type of `application/x-javascript`, but it will be handled automatically by a properly configured server if the filename suffix is `.js`.

For example:

```
<script language="JavaScript" src="http://www.kumquat.  
</script>
```

tells the `<script>`-able browser to load a JavaScript program named *quatscript.js* from the server. Although there are no `<script>` contents, the ending `</script>` still is required.

Used in conjunction with the `src` attribute, the `charset` attribute tells the browser the character set used to encode the JavaScript program. Its value is the name of any ISO standard character set encoding.

### 12.3.1.3 The `defer` attribute

Some JavaScript scripts create actual document content using the

`document.write` method. If your scripts do not alter the contents of the document, add the `defer` attribute to the `<script>` tag to speed its processing. Since the browser knows that it can safely read the remainder of the document without executing your scripts, it defers interpretation of the script until after the document has been rendered for the user.

### 12.3.2 The `<noscript>` Tag

Use the `<noscript>` tag to tell users of browsers that do not support the `<script>` tag that they are missing something. You've already seen many examples of this type of tag. You know the drill...

## <noscript>

### *Function:*

Supplies content to <script>-challenged browsers

### *Attributes:*

`class`, `dir`, `id`, `lang`, `onClick`, `onDblClick`, `onKeyDown`,  
`onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`,  
`onMouseOut`, `onMouseOver`, `onMouseUp`, `style`, `title`

### *End tag:*

`</noscript>`; never omitted

### *Contains:*

*body\_content*

### *Used in:*

*text*

Very old, albeit <script>-able, browsers like Netscape 2 and Internet Explorer 3 blithely display the contents of the <noscript> tag, to the confusion of their users. Given the paucity of users of these browsers, we question the need, but there are ways to detect and handle <script>-challenged browsers, detailed in any good JavaScript book.

The <noscript> tag supports the 16 standard HTML 4/XHTML attributes: `class` and `style` for style management, `lang` and `dir` for language type and display direction, `title` and `id` for titling and naming the enclosed content, and the event attributes for user-initiated processing. [\[Section 3.6.1.1\]](#) [\[Section 3.6.1.2\]](#) [\[Section 4.1.1.4\]](#) [\[Section 4.1.1.4\]](#) [\[Section 8.1.1\]](#) [\[Section 8.3\]](#) [\[Section 12.3.3\]](#)

### 12.3.3 JavaScript Event Handlers

One of the most important features JavaScript provides is the ability to detect and react to events that occur while a document is loading, rendering, and being browsed by the user. The JavaScript code that handles these events may be placed within the `<script>` tag, but more commonly, it is associated with a specific tag via one or more special tag attributes.

For example, you might want to invoke a JavaScript function when the user passes the mouse over a hyperlink in a document. The JavaScript-aware browsers support a special "mouse over" event-handler attribute for the `<a>` tag, called `onMouseOver`, to do just that:

```
<a href="doc.html" onMouseOver="status='Click me!';  
return true">
```

When the mouse passes over this example link, the browser executes the JavaScript statements. (Notice that the two JavaScript statements are enclosed in quotes and separated by a semicolon, and that single quotes surround the text-message portion of the first statement.)

While a complete explanation of this code is beyond our scope, the net result is that the browser places the message "Click me!" in the status bar of the browser window. Commonly, authors use this simple JavaScript function to display a more descriptive explanation of a hyperlink, in place of the often cryptic URL that the browser traditionally displays in the status window.

HTML and XHTML both support a rich set of event handlers through related "on"-event tag attributes. The value of any of the JavaScript event-handler attributes is a quoted string containing one or more JavaScript statements separated by semicolons. Extremely long statements can be broken across several lines, if needed. Care should also be taken in using entities for embedded double quotes in the statements, to avoid syntax errors when processing the attribute values.

### 12.3.3.1 Standard event handler attributes

Table 12-1 presents the current set of event handlers as tag attributes. Most are supported by the popular browsers, which also support a variety of nonstandard event handlers (tagged with asterisks in the table).

We put the event handlers into two categories: user- and document-related. The user-related ones are the mouse and keyboard events that occur when the user handles either device on the computer. User-related events are quite ubiquitous, appearing as standard attributes in nearly all the standard tags (even though they may not yet be supported by any browser), so we don't list their associated tags in Table 12-1. Instead, we'll tell you which tags *do not* accept these event attributes: `<applet>`, `<base>`, `<basefont>`, `<bdo>`, `<br>`, `<font>`, `<frame>`, `<frameset>`, `<head>`, `<html>`, `<iframe>`, `<isindex>`, `<meta>`, `<param>`, `<script>`, `<style>`, and `<title>`.

**Table 12-1. Event handlers**

Event handler	HTML/XHTML tags
onAbort*	<code>&lt;img&gt;</code>
onBlur	<code>&lt;a&gt;</code> <code>&lt;area&gt;</code> <code>&lt;body&gt;</code> <code>&lt;button&gt;</code> <code>&lt;frameset&gt;</code> <code>&lt;input&gt;</code> <code>&lt;label&gt;</code> <code>&lt;select&gt;</code> <code>&lt;textarea&gt;</code>
onChange	<code>&lt;input&gt;</code> <code>&lt;select&gt;</code> <code>&lt;textarea&gt;</code>
onClick	Most tags
onDbClick	Most tags
onError*	<code>&lt;img&gt;</code>

onFocus	<code>&lt;a&gt;&lt;area&gt;&lt;body&gt;&lt;button&gt;&lt;frameset&gt;&lt;input&gt;&lt;label&gt;&lt;select&gt;&lt;textarea&gt;</code>
onKeyDown	<b>Most tags</b>
onKeyPress	<b>Most tags</b>
onKeyUp	<b>Most tags</b>
onLoad	<code>&lt;body&gt;&lt;frameset&gt;&lt;img&gt;*</code>
onMouseDown	<b>Most tags</b>
onMouseMove	<b>Most tags</b>
onMouseOut	<b>Most tags</b>
onMouseOver	<b>Most tags</b>
onMouseUp	<b>Most tags</b>
onReset	<code>&lt;form&gt;</code>
onSelect	<code>&lt;input&gt;&lt;textarea&gt;</code>
onSubmit	<code>&lt;form&gt;</code>
onUnload	<code>&lt;body&gt;&lt;frameset&gt;</code>

Some events, however, occur rarely and with special tags. These relate



to the special events and states that occur during the display and management of a document and its elements by the browser.

### 12.3.3.2 The mouse-related events

The `onClick`, `onDbClick`, `onMouseDown`, and `onMouseUp` attributes refer to the mouse button. The `onClick` event happens when the user presses down and then quickly releases the mouse button, unless the user then quickly clicks the mouse button for a second time. In that latter case, the `onDbClick` event gets triggered in the browser.

If you need to detect both halves of a mouse click as separate events, use `onMouseDown` and `onMouseUp`. When the user presses the mouse button, the `onMouseDown` event occurs. The `onMouseUp` event happens when the user releases the mouse button.

The `onMouseMove`, `onMouseOut`, and `onMouseOver` events happen when the user drags the mouse pointer. The `onMouseOver` event occurs when the mouse first enters the display region occupied by the associated HTML element. After entry, `onMouseMove` events are generated as the mouse moves about within the element. Finally, when the mouse exits the element, `onMouseOut` occurs.

For some elements, the `onFocus` event corresponds to `onMouseOver`, and `onBlur` corresponds to `onMouseOut`.

### 12.3.3.3 The keyboard events

Only three events relating to user keyboard actions currently are supported by the HTML 4 and XHTML standards: `onKeyDown`, `onKeyUp`, and `onKeyPress`. The `onKeyDown` event occurs when the user depresses a key on the keyboard; `onKeyUp` happens when the key is released. The `onKeyPress` event is triggered when a key is pressed and released. Usually, you'll have handlers for either the up and down events or the composite key-press event, but not for both.

### 12.3.3.4 Document events

Most of the document-related event handlers relate to the actions and states of form controls. For instance, `onReset` and `onSubmit` happen when the user activates the respective reset or submit button. Similarly, `onSelect` and `onChange` occur as users interact with certain form elements. See [Chapter 9](#) for a detailed discussion of these forms-related events.

There also are some document-related event handlers that occur when various document elements get handled by the browser. For instance, the `onLoad` event may happen when a frameset is complete or when the body of an HTML or XHTML document gets loaded and displayed by the browser. Similarly, `onUnload` occurs when a document is removed from a frame or window.

### 12.3.4 Javascript URLs

You can replace any conventional URL reference in a document with one or more JavaScript statements. The browser then executes the JavaScript code, rather than downloading another document, whenever the browser references the URL. The result of the last statement is taken to be the "document" referenced by the URL and is displayed by the browser accordingly. The result of the last statement is *not* the URL of a document; it is the actual content to be displayed by the browser.

To create a javascript URL, use `javascript` as the URL's protocol:

```
<a href="javascript:generate_document( )">
```

In this example, the JavaScript function `generate_document( )` gets executed whenever the user selects the hyperlink. The value returned by the function, presumably a valid HTML or XHTML document, is rendered and displayed by the browser.

It may be that the executed statement returns no value. In this case, the

current document is left unchanged. For example, this javascript URL:

```
<a href="javascript:alert('Error!') ">
```

pops up an alert dialog box and does nothing else. The document containing the hyperlink is still visible after the dialog box is displayed and dismissed by the user.

### 12.3.5 JavaScript Entities

Character entities in HTML and XHTML consist of an ampersand (&), an entity name or number, and a closing semicolon. For instance, to insert the ampersand character itself in a document text flow, use the character sequence `&amp;`. Similarly, JavaScript entities consist of an ampersand, one or more JavaScript statements enclosed in curly braces, and a closing semicolon. For example:

```
&{document.fgColor};
```

Multiple statements must be separated by semicolons within the curly braces. The value of the last (or only) statement is converted to a string and replaces the entity in the document.

Normally, entities can appear anywhere in an document. JavaScript entities, however, are restricted to values of tag attributes. This lets you write "dynamic tags" whose attributes are not known until the document is loaded and the JavaScript is executed. For example, this tag sets the text color of the document to the color value returned by the individual's `favorite_color( )` function:

```
<body text=&{favorite_color( )};>
```

### 12.3.6 The <server> Tag

The `<server>` tag is a strange beast. It is processed by the web server and never seen by the browser, so what you can do with this tag depends

on the server you are using, not on the reader's browser.

Netscape's web servers (not to be confused with their browser) uses the `<server>` tag to let you to place JavaScript statements within a document that the server processes. The results of the executed JavaScript are then inserted into the document, replacing the `<server>` tag. A complete discussion of this so-called "server-side" JavaScript is completely beyond this book; we include this brief reference only to document the `<server>` tag.

## **<server>** ☐

### *Function:*

Defines server-side JavaScript

### *Attributes:*

None

### *End tag:*

`</server>;` never omitted

### *Contains:*

*JavaScript*

### *Used in:*

*head\_content*

Like the `<script>` tag, the `<server>` tag contains JavaScript code. However, the latter tag and content code must appear inside the document `<head>`. It is extracted from the document and executed by the server when the document is requested for download.

Obviously, server-side JavaScript is tightly coupled to the server, not to the browser. To fully exploit this tag and the benefits of server-side JavaScript or other server-side programming languages, consult your web server's documentation.

## Chapter 12. Executable Content

One of the most useful web technologies is the ability to deliver applications directly to the browser. These typically small programs perform simple tasks on the client computer, from responding to user mouse or keyboard actions to spicing up your web page displays with multimedia-enabling software.

You can embed scripts in your documents using a language known as JavaScript. Or you can load and execute small, Java-based, platform-independent applications known as *applets*. During execution, these programs may generate dynamic content, interact with the user, validate form data, or even create windows and run entire applications independent of your pages. The possibilities are endless, and they go far beyond the simple document model originally envisioned for HTML.

In this chapter, we show you, with simple examples, how to include two kinds of executable content—scripts and applets—in your documents. We won't, however, teach you how to write and debug executable content. This is a book about HTML and XHTML, after all. Rather, get an expert opinion: turn to any of the many excellent texts from O'Reilly, including *JavaScript: The Definitive Guide*, by David Flanagan, *Java in a Nutshell*, also by David Flanagan, and *Learning Java*, by Pat Niemeyer and Jonathan Knudsen.

## 6.3 Creating Hyperlinks

Use the HTML/XHTML `<a>` tag to create links to other documents and to name anchors for fragment identifiers within documents.

### 6.3.1 The `<a>` Tag

You will use the `<a>` tag most commonly with its `href` attribute to create a hypertext link, or *hyperlink*, to another place in the same document or to another document. In these cases, the current document is the source of the link; the value of the `href` attribute, a URL, is the target.<sup>[7]</sup>

<sup>[7]</sup> You may run across the terms "head" and "tail," which reference the target and source of a hyperlink. This naming scheme assumes that the referenced document (the head) has many tails that are embedded in many referencing documents throughout the Web. We find this naming convention confusing and stick to the concept of source and target documents throughout this book.

The other way you can use the `<a>` tag is with the `name` attribute, to mark a hyperlink target, or fragment identifier, in a document. This method, although part of the HTML 4 and XHTML standards, is slowly succumbing to the `id` attribute, which lets you mark nearly any element, including paragraphs, divisions, forms, and so on, as a hyperlink target.

## <a>

### *Function*

Defines anchors within a text flow

### *Attributes*

accesskey, charset, class, coords, dir, href, hreflang, id, lang, name, onBlur, onClick, onDblClick, onFocus, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, rel, rev, shape, style, tabindex, target, title, type

### *End tag*

</a>; never omitted

### *Contains*

*a\_content*

### *Used in*

*text*

The standards let you use both the `name` and `href` attributes within a single `<a>` tag, defining a link to another document and a fragment identifier within the current document. We recommend against this, since it overloads a single tag with multiple functions and some browsers may not be able to handle it. Instead, use two `<a>` tags when such a need arises. Your source will be easier to understand and modify and will work better across a wider range of browsers.

#### **6.3.1.1 Allowed content**

Between the `<a>` tag and its required end tag, you may put only regular



text, line breaks, images, and headings. The browser renders all of these elements normally, but with the addition of some special effects to indicate that they are hyperlinks to other documents. For instance, the popular graphical browsers typically underline and color the text and draw a colored border around images that are enclosed by `<a>` tags.

While the allowed content may seem restricted (the inability to place style markup within an `<a>` tag is a bit onerous, for instance), most browsers let you put just about anything within an `<a>` tag that makes sense. To be compliant with the HTML 4 and XHTML standards, place the `<a>` tag inside other markup tags, not the opposite. For example, while most browsers make sense of either variation on this anchor theme:

To subscribe to

```
<cite><a href="ko.html">Kumquat Online</a></cite>,
```

To subscribe to

```
<a href="ko.html"><cite>Kumquat Online</cite></a>,
```

only the first example is technically correct, and the second is most certainly incorrect for XHTML.

### 6.3.1.2 The href attribute

Use the `href` attribute to specify the URL of the target of a hyperlink. Its value is any valid document URL, absolute or relative, including a fragment identifier or a JavaScript code fragment. If the user selects the contents of the `<a>` tag, the browser will attempt to retrieve and display the document indicated by the URL specified by the `href` attribute or execute the list of JavaScript expressions, methods, and functions.

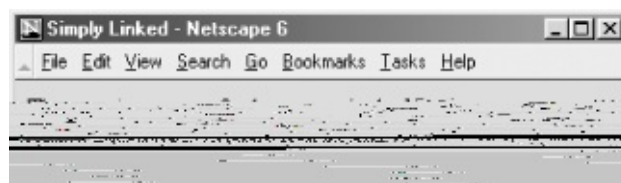
[[Section 6.2](#)]

A simple `<a>` tag that references another document might be:

```
The <a href="http:growing_season.html">growing
season</a> for kumquats in the Northeast.
```

which appears in the Netscape display shown in [Figure 6-1](#).

**Figure 6-1. Hyperlink to another HTML document**



Notice that the phrase "growing season" is specially rendered by the browser, letting the user know that it is a link to another document. Users usually have the option to set their own text color for the link and have the color change when a link is taken; blue initially and then red after it has been selected at least once, for instance. More complex anchors might include images:

```
<ul>
```

```
<li><a href="pruning_tips.html">
```

```

```

```
New pruning tips!</a>
```

```
<p>
```

```
<li><a href="xhistory.html">
```

```


```

```
Kumquats throughout history</a>
```

```
</ul>
```

Most graphical browsers, like Netscape and Internet Explorer, place a special border around images that are part of an anchor, as shown in [Figure 6-2](#). Remove that hyperlink border with the `border=0` attribute and value within the `<img>` tag for the image. [[Section 5.2.6.8](#)]

**Figure 6-2. Internet Explorer puts a special border around an image that is inside an anchor**



figs/htm5\_0602.gif

### 6.3.1.3 The name and id attributes

Use the `name` and `id` attributes with the `<a>` tag to create a fragment identifier within a document. Once created, the fragment identifier becomes a potential target of a link.

Prior to HTML 4.0, the only way to create a fragment identifier was to use the `name` attribute with the `<a>` tag. With the advent of the `id` attribute in HTML 4.0, and its ability to be used with almost any tag, any HTML or XHTML element can be a fragment identifier. The `<a>` tag retains the `name` attribute for historic purposes and honors the `id` attribute as well. These attributes can be used interchangeably, with `id` being the more "modern" version of the `name` attribute. Both `name` and `id` can be specified in conjunction with the `href` attribute, allowing a single `<a>` to be both a hyperlink and a fragment identifier.

An easy way to think of a fragment identifier is as the HTML analog of the `goto` statement label common in many programming languages. The `name` attribute within the `<a>` tag or the `id` attribute within the `<a>` or other tags places a label within a document. When that label is used in a link to that document, it is the equivalent of telling the browser to `goto` that label.

The value of the `id` or `name` attribute is any character string, enclosed in quotation marks. The string must be a unique label, not reused in any other `name` or `id` attribute in the same document, although it can be reused in different documents.

Here are some `name` and `id` examples:

```
<h2><a name="Pruning">Pruning Your Kumquat Tree</a></h2>
```

```
<h2 id="Pruning">Pruning Your Kumquat Tree</h2>
```

Notice that we set the anchor in a section header of a presumably large document. It's a practice we encourage you to use for all major sections of your work for easier reference and future smart processing, such as automated extraction of topics.

The following link, when taken by the user:

```
<a href="growing_guide.html#Pruning">
```

jumps directly to the section of the document we named in the previous examples.

The contents of the anchor `<a>` tag with the `name` or `id` attribute are not displayed in any special way.

Technically, you do not have to put any document content within the `<a>` tag with the `name` attribute, since it simply marks a location in the document. In practice, though, some browsers ignore the tag unless some document content—a word or phrase, even an image—is between the `<a>` and `</a>` tags. For this reason, it's probably a good idea to have

at least one displayable element in the body of any `<a>` tag.

#### 6.3.1.4 The event attributes

There are a number of event handlers built into the modern browsers. These handlers watch for certain conditions and user actions, such as a click of the mouse or when an image finishes loading into the browser window. With client-side JavaScript, you may include selected event handlers as attributes of certain tags and execute one or more JavaScript commands and functions when the event occurs.

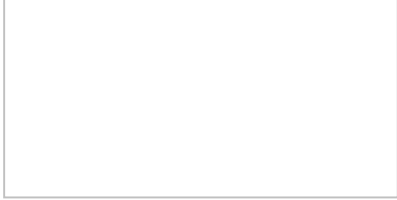
With the anchor (`<a>`) tag, you may associate JavaScript code with a number of mouse- and keyboard-related events. The value of the event handler is enclosed in quotation marks one or a sequence of semicolon-separated JavaScript expressions, methods, and function references that the browser executes when the event occurs. [Section 12.3.3]

A popular, albeit simple, use of the `onMouseOver` event with a hyperlink is to print an expanded description of the tag's destination in the JavaScript-aware browser's status box (Figure 6-3). Normally, the browser displays the frequently cryptic destination URL there whenever the user passes the mouse pointer over an `<a>` tag's contents:

```
<a href="http://www.ora.com/kumquats/homecooking/recipe.html"
onMouseOver="status='A yummy recipe for kumquat soup.'"
>
  
  A yummy recipe for kumquat soup.
</a>
```

**Figure 6-3. Use JavaScript to display a message in the browser's status box**

figs/htm5\_0603.gif



We argue that the contents of the tag itself should explain the link, but there are times when window space is tight and an expanded explanation is helpful, such as when the link is in a table of contents.

See [Chapter 12](#) for more about JavaScript.

### 6.3.1.5 The `rel` and `rev` attributes

The optional `rel` and `rev` attributes for the `<a>` tag express a formal relationship and direction between source and target documents. The `rel` attribute specifies the relationship from the source document to the target, and the `rev` attribute specifies the relationship from the target to the source. Both attributes can be placed in a single `<a>` tag, and the browser may use them to specially alter the appearance of the anchor content or to automatically construct document navigation menus. Other tools also may use these attributes to build special link collections, tables of contents, and indexes.

The value of either the `rel` or `rev` attribute is a space-separated list of relationships. The actual relationship names and their meanings are up to you: they are not formally addressed by the HTML or XHTML standards. For example, a document that is part of a sequence of documents might include its relationship in a link:

```
<a href="part-14.html" rel=next rev=prev>
```

The relationship from the source to the target is that of moving to the next document; the reverse relationship is that of moving to the previous document.

These document relationships are also used in the `<link>` tag in the document `<head>`. The `<link>` tag establishes the relationship without actually creating a link to the target document; the `<a>` tag creates the link and imbues it with the relationship attributes. [`<link>`]

Commonly used document relationships include:

`next`

Links to the next document in a collection

`prev`

Links to the previous document in a collection

`head`

Links to the top-level document in a collection

`toc`

Links to a collection's table of contents

`parent`

Links to the document above the source

`child`

Links to a document below the source

`index`

Links to the index for this document

`glossary`

Links to the glossary for this document

Few browsers take advantage of these attributes to modify the link

appearance. However, these attributes are a great way to document links you create, and we recommend that you take the time to insert them whenever possible.

#### 6.3.1.6 The `style` and `class` attributes

Use the `style` and `class` attributes for the `<a>` tag to control the display style for the content enclosed by the tag and to format the content according to a predefined class of the `<a>` tag. [[Section 8.1.1](#)] [[Section 8.3](#)]

#### 6.3.1.7 The `lang` and `dir` attributes

Like almost all other tags, the `<a>` tag accepts the `lang` and `dir` attributes, denoting the language used for the content within the `<a>` tag and the direction in which that language is rendered. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)]

#### 6.3.1.8 The `target` attribute

The `target` attribute lets you specify where to display the contents of a selected hyperlink. Commonly used in conjunction with frames or multiple browser windows, the value of this attribute is the name of the frame or window in which the referenced document should be loaded. If the named frame or window exists, the document is loaded in that frame or window. If not, a new window is created and given the specified name, and the document is loaded in that new window. For more information, including a list of special target names, see [Section 11.7](#).

#### 6.3.1.9 The `title` attribute

The `title` attribute lets you specify a title for the document to which you are linking. The value of the attribute is any string, enclosed in quotation



marks. The browser might use it when displaying the link, perhaps flashing the title when the mouse passes over the link. The browser might also use the `title` attribute when adding this link to a user's bookmarks or favorites.

The `title` attribute is especially useful for referencing an otherwise unlabeled resource, such as an image or a non-HTML document. For example, the browser might include the following title on this otherwise wordless image display page:

```
<a href="pics/kumquat.gif"
    title="A photograph of the Noble Fruit">
```

Ideally, the value specified should match the title of the referenced document, but it's not required.

#### 6.3.1.10 The `charset`, `hreflang`, and `type` attributes

According to the HTML 4 and XHTML standards, the `charset` attribute specifies the character encoding used in the document that is the destination of the link. The value of this attribute must be the name of a standard character set: "euc-jp," for example. The default value is "ISO-8859-1".

The `hreflang` attribute may be specified only when the `href` attribute is used. Like the `lang` attribute, its value is an ISO standard two-character language code. Unlike the `lang` attribute, the `hreflang` attribute does not address the language used by the contents of the tag. Instead, it specifies the language used in the document referenced by the `href` attribute. [Section 3.6.1.2]

The `type` attribute specifies the content type of the resource referenced by the `<a>` tag. Its value is any MIME encoding type. For example, you might inform the browser that you are linking to a plain ASCII document with:

```
<a href="readme.txt" type="text/plain">
```

The browser might use this information when displaying the referenced document, or might even present the link differently based upon the content type.

#### 6.3.1.11 The `coords` and `shape` attributes

These are two more attributes defined in the HTML and XHTML standards for the `<a>` tag that are not supported by the currently popular browsers. Like the attributes of the same names for the `<area>` tag, the `coords` and `shape` attributes define a region of influence for the `<a>` tag. These attributes should be used with the `<a>` tag only when that tag is part of the content of a `<map>` tag, as described later in this chapter. [\[Section 6.5.3\]](#) [\[Section 6.5.4.2\]](#) [\[Section 6.5.4.7\]](#)

#### 6.3.1.12 The `accesskey` and `tabindex` attributes

Traditionally, users of graphical browsers select and execute a hyperlink by pointing and clicking the mouse device on the region of the browser display defined by the anchor. What is less well known is that you may choose a hyperlink, among other objects in the browser window, by pressing the Tab key and then activate that link by pressing the Enter key. With the `tabindex` attribute, you may reorder the sequence in which the browser steps through to each object when the user presses the Tab key. The value of this attribute is an integer greater than 0. The browser starts with the object whose tab index is 1 and moves through the other objects in increasing order.

With the `accesskey` attribute, you may select an alternative "hot-key" that, when pressed, activates the specific link. The value of this attribute is a single character that is pressed in conjunction with an "alt" or "meta" key, depending on the browser and computing platform. Ideally, this character should appear in the content of the `<a>` tag; if so, the browser may choose to display the character differently to indicate that it is a hot-

key.

See an expanded description for both of these attributes in [Chapter 9](#).

### 6.3.2 Linking to Other Documents

Say you make a hyperlink to another document with the `<a>` tag and its `href` attribute, which defines the URL of the target document. The contents of the `<a>` tag are presented to the user in some distinctive manner in order to indicate that the link is available.

When creating a link to another document, you should consider adding the `title`, `rel`, and `rev` attributes to the `<a>` tag. They help document the link you are creating and allow the browser to embellish the display anchor contents.

### 6.3.3 Linking Within a Document

Creating a link within the same document or to a specific fragment of another document is a two-step process. The first step is to make the target fragment; the second is to create the link to the fragment.

Use the `<a>` tag with its `name` attribute to identify a fragment. Here's a sample fragment identifier:

```
<h3><a name="Section_7">Section 7</a></h3>
```

Alternatively, use the `id` attribute and embed the hyperlink target directly in a defining tag, such as a header:<sup>[8]</sup>

<sup>[8]</sup> We prefer the `id` way, although not all browsers support it, yet.

```
<h3 id="Section_7">Section 7</h3>
```

A hyperlink to the fragment is an `<a>` tag with the `href` attribute, in which the attribute's value the target URL ends with the fragment's name,

preceded by the pound sign (#). A reference to the previous example's fragment identifier, then, might look like:

```
See <a href="index.html#Section_7">Section 7</a>  
for further details.
```

By far the most common use of fragment identifiers is in creating a table of contents for a lengthy document. Begin by dividing your document into several logical sections, using appropriate headers and consistent formatting. At the start of each section, add a fragment identifier for that section, typically as part of the section title. Finally, make a list of links to those fragment identifiers at the beginning of your document.

Our sample document extolling the life and wonders of the mighty kumquat, for example, is quite long and involved, including many sections and subsections of interest. It is a document to be read and read again. In order to make it easy for kumquat lovers everywhere to find their section of interest quickly, we've included fragment identifiers for each major section and placed an ordered list of links a hotlinked table of contents, as it were at the beginning of each of the Kumquat Lover's documents, a sample of which appears below, along with sample fragment identifiers that appear in the same document. The ellipsis symbol (...) means that there are intervening segments of content, of course:

...

```
<h3>Table of Contents</h3>
```

```
<ol>
```

```
<li><a href="#soil_prep">Soil Preparation</a>
```

```
<li><a href="#dig_hole">Digging the Hole</a>
```

```
<li><a href="#planting">Planting the Tree</a>
```

```
</ol>
```

```
...
```

```
<h3 id=soil_prep>Soil Preparation</h3>
```

```
...
```

```
<h3 id=dig_hole>Digging the Hole</h3>
```

```
...
```

```
<h3 id=planting>Planting the Tree</h3>
```

```
...
```

The kumquat lover can thereby click the desired link in the table of contents and jump directly to the section of interest, without lots of tedious scrolling.

Notice also that this example uses relative URLs a good idea if you ever intend to move or rename the document without breaking all the hyperlinks.

## 6.5 Mouse-Sensitive Images

Normally, an image placed within an anchor simply becomes part of the anchor content. The browser may alter the image in some special way (usually with a special border) to alert the reader that it is a hyperlink, but users click the image in the same way they click a textual hyperlink.

The HTML and XHTML standards provide a feature that lets you embed many different links inside the same image. Clicking different areas of the image causes the browser to link to different target documents. Such mouse-sensitive images, known as *image maps*, open up a variety of creative linking styles.

There are two ways to create image maps, known as *server-side* and *client-side* image maps. The former, enabled by the `ismap` attribute for the `<img>` tag, requires access to a server and related image-map processing applications. The latter is created with the `usemap` attribute for the `<img>` tag, along with corresponding `<map>` and `<area>` tags.

Translation of the mouse position in the image to a link to another document happens on the user's machine, so client-side image maps don't require a special server connection and can even be implemented in non-Web environments, such as on a local hard drive or in a CD-ROM-based document collection. Any HTML/XHTML can implement a client-side (`usemap`) image map. [`<map>`] [`<area>`] [[Section 5.2.6](#)]

### 6.5.1 Server-Side Image Maps

You add an image to an anchor simply by placing an `<img>` tag within the body of the `<a>` tag. Make that embedded image into a mouse-sensitive one by adding the `ismap` attribute to the `<img>` tag. This special `<img>` attribute tells the browser that the image is a special map containing more than one link. (The `ismap` attribute is ignored by the browser if the `<img>` tag is not within an `<a>` tag.) [[Section 5.2.6](#)]

When the user clicks some place within the image, the browser passes the coordinates of the mouse pointer along with the URL specified in the `<a>` tag to the document server. The server uses the mouse-pointer coordinates to determine which document to deliver back to the browser.

When `ismap` is used, the `href` attribute of the containing `<a>` tag must contain the URL of a server application or, for some HTTP servers, a related map file that contains the coordinate and linking information. If the URL is simply that of a conventional document, errors may result, and the desired document probably will not be retrieved.

The coordinates of the mouse position are screen pixels counted from the upper-left corner of the image, beginning with (0,0). The coordinates, preceded by a question mark, are added to the end of the URL.

For example, if a user clicks 43 pixels over and 15 pixels down from the upper-left corner of the image displayed from the following link:

```
<a href="/cgi-bin/imagemap/toolbar.map">  
  
  
  
</a>
```

the browser sends the following search parameters to the HTTP server:

```
/cgi-bin/imagemap/toolbar.map?43,15
```

In the example, *toolbar.map* is a special image map file located inside the *cgi-bin/imagemap* directory and containing coordinates and links. A special image map process uses that file to match the passed coordinates (43,15 in our example) and return the selected hyperlink document.

### 6.5.1.1 Server-side considerations

With mouse-sensitive, `ismap`-enabled image maps, the browser is

required to pass along only the URL and mouse coordinates to the server. Converting the coordinates into a specific document is handled by the document server. The conversion process differs between servers and is not defined by the HTML or XHTML standards.

You need to consult with your web server administrators and perhaps even read your server's documentation to determine how to create and program an image map. Most servers come with some software utility, typically located in a *cgi-bin/imagemap* directory, to handle image maps. And most of these use a text file containing the image map regions and related hyperlinks that is referenced by your image map URL to process the image map query.

Here's an example image map file that describes the sensitive regions in our example image:

```
# Imagemap file=toolbar.map

default                dflt.html

circ 100,30,50          link1.html

rect 180,120,290,500    link2.html

poly 80,80,90,72,160,90 link3.html
```

Each sensitive region of the image map is described by a geometric shape and defining coordinates in pixels, such as the circle with its center point and radius, the rectangle's upper-left and lower-right edge coordinates, and the loci of a polygon. All coordinates are relative to the upper-left corner of the image (0,0). Each shape has a related URL.

An image-map processing application typically tests each shape in the order in which it appears in the image file and returns the document specified by the corresponding URL to the browser if the user's mouse x,y coordinates fall within the boundaries of that shape. That means it's



okay to overlap shapes; just be aware which takes precedence. Also, the entire image need not be covered with sensitive regions: if the passed coordinates don't fall within a specified shape, the default document gets sent back to the browser.

This is just one example of how an image map may be processed and the accessory files required for that process. Please huddle with your webmaster and server manuals to discover how to implement a server-side image map for your own documents and system.

## 6.5.2 Client-Side Image Maps

The obvious down side to server-side image maps is that they require a server. That means you need access to the required HTTP server or its */cgi-bin* directory, either of which is rarely available to anyone other than owners or system administrators. And server-side image maps limit portability, since not all image-map processing applications are the same.

Server-side image maps also mean delays for the user while browsing, since the browser must get the server's attention to process the image coordinates. That's even if there's no action to take, such as when the user clicks on a section of the image that isn't hyperlinked and doesn't lead anywhere.

Client-side image maps suffer from none of these difficulties. Enabled by the `usemap` attribute for the `<img>` tag and defined by special `<map>` and `<area>` extension tags, client-side image maps let authors include in their documents maps of coordinates and links that describe the sensitive regions of an image. The browser on the client computer translates the coordinates of the mouse position within the image into an action, such as loading and displaying another document. And special JavaScript-enabled attributes provide a wealth of special effects for client-side image maps. [[Section 12.3.3](#)]

To create a client-side image map, include the `usemap` attribute as part of the `<img>` tag.<sup>[9]</sup> Its value is the URL of a `<map>` segment in an HTML document that contains the map coordinates and related link URLs. The

document in the URL identifies the HTML or XHTML document containing the map; the fragment identifier in the URL identifies the map to be used. Most often, the map is in the same document as the image itself, and the URL can be reduced to the fragment identifier: a pound sign (#) followed by the map name.

[9] Alternatively, according to the HTML 4 standard, you may reference a client-side image map by including the `usemap` attribute with the `<object>` and form `<input>` tags. See [Chapter 12](#) for details.

For example, the following source fragment tells the browser that the *map.gif* image is a client-side image map and that its mouse-sensitive coordinates and related link URLs are found in the `map` section of the document named *map*:

```

```

### 6.5.3 The `<map>` Tag

For client-side image maps to work, you must include somewhere in your document a set of coordinates and URLs that define the mouse-sensitive regions of a client-side image map and the hyperlink to take for each region that may be clicked or otherwise selected<sup>[10]</sup> by the user. Include those coordinates and links as values of attributes in conventional `<a>` tags or special `<area>` tags; the collection of `<area>` specifications or `<a>` tags are enclosed within the `<map>` tag and its end tag, `</map>`. The `<map>` segment may appear anywhere in the body of the document.

[10] The Tab key also steps through the hyperlinks in a document, including client-side image maps. Select a chosen hyperlink with the Enter key.

## <map>

### *Function*

Encloses client-side image map (`usemap`) specifications

### *Attributes*

`class`, `dir`, `id`, `lang`, `name`, `onClick`, `onDblClick`,  
`onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`,  
`onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`,  
`style`, `title`

### *End tag*

`</map>`; never omitted

### *Contains*

*map\_content*

### *Used in*

*body\_content*

More specifically, the `<map>` tag may contain either a sequence of `<area>` tags or conventional HTML/XHTML content including `<a>` tags. You cannot mix and match `<area>` tags with conventional content. Conventional content within the `<map>` tag may be displayed by the browser; `<area>` tags will not. If you are concerned about compatibility with older browsers, use only `<map>` tags containing `<area>` tags.

If you do place `<a>` tags within a `<map>` tag, they must include the `shape` and `coords` attributes that define a region within the objects that reference the `<map>` tag.

### 6.5.3.1 The name attribute

The value of the `name` attribute in the `<map>` tag is the name used by the `usemap` attribute in an `<img>` or `<object>` tag to locate the image map specification. The name must be unique and not used by another `<map>` in the document, but more than one image map may reference the same `<map>` specifications. [[Section 5.2.6.14](#)]

### 6.5.3.2 The `class`, `id`, `style`, and `title` attributes

The style sheet display-related `style` and `class` attributes for the `<map>` tag are useful only when the `<map>` tag contains conventional content, in which case they apply to the content of the tag. [[Section 8.1.1](#)] [[Section 8.3](#)]

The `id` and `title` attributes, on the other hand, are straightforward. They are standard ways to respectively label the tag for later reference by a hyperlink or program or entitle the section for later review. [[Section 4.1.1.4](#)] [[Section 4.1.1.4](#)]

### 6.5.3.3 The event attributes

The various event attributes allow you to assign JavaScript handlers to events that may occur within the confines of the map. [[Section 12.3.3](#)]

## 6.5.4 The `<area>` Tag

The guts of a client-side image map are the `<area>` tags within the map segment. These `<area>` tags define each mouse-sensitive region and the action the browser should take if it is selected by the user in an associated client-side image map.

## <area>

### Function

Defines coordinates and links for a region on a client-side image map

### Attributes

`accesskey`, `alt`, `class`, `coords`, `dir`, `href`, `id`, `lang`, `nohref`, `notab`, `onBlur`, `onClick`, `onDblClick`, `onFocus`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `shape`, `style`, `tabindex`, `taborder` (☐) , `target` (☐☐) , `title`, `type`

### End tag

None in HTML; `</area>` or `<area ... />` in XHTML

### Contains

Nothing

### Used in

*map\_content*

The region defined by an `<area>` tag acts just like any other hyperlink: when the user moves the mouse pointer over the region of the image, the pointer icon changes, typically into a hand, and the browser may display the URL of the related hyperlink in the status box at the bottom of the browser window.<sup>[11]</sup> Regions of the client-side image map not defined in at least one `<area>` tag are not mouse-sensitive.

<sup>[11]</sup> That is, unless you activate a JavaScript event handler that writes the contents of the status box. See the `onMouse` event handlers in [Section 6.5.4.6](#).

### 6.5.4.1 The alt attribute

Like its cousin for the `<img>` tag, the `alt` attribute for the `<area>` tag attaches a text label to the image, except in this case the label is associated with a particular area of the image. The popular browsers display this label to the user when the mouse passes over the area, and nongraphical browsers may use it to present the client-side image map as a list of links identified by the `alt` labels.

### 6.5.4.2 The coords attribute

The required `coords` attribute of the `<area>` tag defines coordinates of a mouse-sensitive region in a client-side image map. The number of coordinates and their meanings depend upon the region's shape as determined by the `shape` attribute, discussed later in this chapter. You may define hyperlink regions as rectangles, circles, and polygons within a client-side image map.

The appropriate values for each shape include:

`circle` or `circ`

`coords="x, y, r "`, where `x` and `y` define the position of the center of the circle (0,0 is the upper-left corner of the image) and `r` is its radius in pixels.

`polygon` or `poly`

`coords="x1, y1, x2, y2, x3, y3, . . ."`, where each pair of `x,y` coordinates defines a vertex of the polygon, with 0,0 being the upper-left corner of the image. At least three pairs of coordinates are required to define a triangle; higher-order polygons require a larger number of vertices. The polygon is automatically closed, so it is not necessary to repeat the first coordinate at the end of the list to close the region.

rectangle or rect

`coords="x1,y1,x2,y2"`, where the first coordinate pair is one corner of the rectangle and the other pair is the corner diagonally opposite, with 0,0 being the upper-left corner of the image. Note that a rectangle is just a shortened way of specifying a polygon with four vertices.

For example, the following XHTML fragment defines a single mouse-sensitive region in the lower-right quarter of a 100 x 100-pixel image and another circular region smack in the middle:

```
<map name="map1">

  <area shape="rect" coords="75,75,99,99" nohref="nohref"

  <area shape="circ" coords="50,50,25" nohref="nohref"

</map>
```

If the coordinates in one `<area>` tag overlap with another region, the first `<area>` tag takes precedence. The browsers ignore coordinates that extend beyond the boundaries of the image.

#### 6.5.4.3 The href attribute

Like the `href` attribute for the anchor (`<a>`) tag, the `href` attribute for the `<area>` tag defines the URL of the desired link if its region in the associated image map is clicked. The value of the `href` attribute is any valid URL, relative or absolute, including JavaScript code.

For example, the browser will load and display the *link4.html* document if the user clicks in the lower-left quarter of a 100 x 100-pixel image, as defined by the first image map `<area>` tag in the following HTML example:

```
<map name="map">
```

```
<area coords="75,75,99,99" href="link4.html">  
  
<area coords="0,0,25,25" href="javascript:window.ale:  
  
</map>
```

The second `<area>` tag in the example uses a javascript URL, which, when the user clicks in the upper-left quadrant of the image map, executes a JavaScript alert method that displays the silly message in a dialog box.

#### 6.5.4.4 The `nohref` attribute

The `nohref` attribute for the `<area>` tag defines a mouse-sensitive region in a client-side image map for which no action is taken, even though the user may select it. You must include either an `href` or a `nohref` attribute for each `<area>` tag.

#### 6.5.4.5 The `notab`, `taborder`, and `tabindex` attributes

As an alternative to the mouse, a user may choose a document "hot spot," such as a hyperlink embedded in an image map, by pressing the Tab key. Once chosen, the user activates the hyperlink by pressing the Enter key. By default, the browser steps to each hot spot in the order in which they appear in the document. Originally introduced by Internet Explorer with the `taborder` attribute, and now standardized as the `tabindex` attribute, you may change that default order. The value of the attribute is an integer indicating the position of this area in the overall tab sequence for the document.

Supported by Internet Explorer only and not part of either the HTML 4 or XHTML standards, `notab` areas get passed over as the user presses the Tab key to move the cursor around the document. Otherwise, this area will be part of the tabbing sequence. The attribute is useful, of course, in



combination with the `nohref` attribute.

The `notab` and `taborder` attributes were supported by Internet Explorer Version 4. Versions 5 and later support `tabindex` too, so use the standard instead of the extension attributes.

#### 6.5.4.6 The event attributes

The same mouse-related JavaScript event handlers that work for the anchor (`<a>`) tag also work with client-side image map hyperlinks. The value of the event handler is enclosed in quotation marks one or a sequence of semicolon-separated JavaScript expressions, methods, and function references that the browser executes when the event occurs. [Section 12.3.3]

For example, a popular, albeit simple, use of the `onMouseOver` event is to print a more descriptive explanation in the browser's status box whenever the user passes the mouse pointer over a region of the image map:

```
<area href="http://www.oreilly.com/kumquats/homecooking.html"
      onMouseOver="self.status='A recipe for kumquat soup'"
```

We should point out that the current versions of the popular browsers automatically display the `alt` attribute's string value, ostensibly accomplishing the same task. So we recommend that you include the `alt` attribute and value in lieu of hacking JavaScript. And, in context with a text-based hyperlink, we argue that the contents of the tag itself should explain the link. But images can be deceptive, so we urge you to take advantage of both the `alt` attribute and event handlers to provide text descriptions with your image maps.

#### 6.5.4.7 The shape attribute

Use the `shape` attribute to define the shape of an image map's mouse-sensitive region: a circle (`circ` or `circle`), polygon (`poly` or `polygon`), or rectangle (`rect` or `rectangle`).

The value of the `shape` attribute affects how the browser interprets the value of the `coords` attribute. If you don't include a `shape` attribute, the value `default` is assumed. According to the standard, `default` means that the area covers the entire image. In practice, the browsers default to a rectangular area and expect to find four `coords` values. If you don't specify a shape and don't include four coordinates with the tag, the browsers ignore the area altogether.

In fact, Netscape is the only browser that even recognizes the `shape` value `default` to provide a catch-all area for clicks that fall outside all the other defined hot spots. Since areas are in a "first-come, first-served" order in the `<map>` tag, you should place the default area last. Otherwise, it covers up any and all areas that follow in your image map.

The browsers are lax in their implementation of the shape names. Netscape 4, for example, doesn't recognize "rectangle" but does recognize "rect" for a rectangular shape. For this reason, we recommend that you use the abbreviated names.

#### 6.5.4.8 The target attribute

The `target` attribute gives you a way to control where the contents of the selected hyperlink in the image map get displayed. Commonly used in conjunction with frames or multiple browser windows, the value of this attribute is the name of the frame or window in which the referenced document should be loaded. If the named frame or window exists, the document is loaded in that frame or window. If not, a new window is created and given the specified name, and the document is loaded in that new window. For more information, including a list of special target names, see [Section 11.7](#).

#### 6.5.4.9 The title attribute

The `title` attribute lets you specify a title for the document to which the image map's area links. The value of the attribute is any string, enclosed in quotes. The browser might use the title when displaying the link, perhaps flashing the title when the mouse passes over the area. The browser might also use the `title` attribute when adding this link to a user's bookmarks or favorites.

The `title` attribute is especially useful for referencing an otherwise unlabeled resource, such as an image or a non-HTML document. Ideally, the value specified should match the title of the referenced document, but this isn't required.

#### 6.5.4.10 The class, dir, id, lang, and style attributes

The `class` and `style` attributes allow you to supply display properties and class names to control the appearance of the area, although their value seems limited for this tag. The `id` attribute allows you to create a name for the area that might be referenced by a hyperlink. [[Section 4.1.1.4](#)] [[Section 8.1.1](#)] [[Section 8.3](#)]

The `lang` and `dir` attributes define the language used for this area and the direction in which text is rendered. Again, their use is not apparent with this tag. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)]

### 6.5.5 A Client-Side Image Map Example

The following example HTML fragment draws together the various components of a client-side image map discussed earlier in this section. It includes the `<img>` tag with the image reference and a `usemap` attribute with a `name` that points to a `<map>` that defines four mouse-sensitive regions (three plus a default) and related links:

```
<body>
```

...

```

```

...

```
<map name="map1">
```

```
  <area shape=rect coords="0,20,40,100"
```

```
    href="k_juice.html"
```

```
    onMouseOver="self.status='How to prepare kumquat  
    ;return true">
```

```
<area shape=rect coords="50,50,80,100"
```

```
    href="k_soup.html"
```

```
    onMouseOver="self.status='A recipe for hearty ku  
    ;return true">
```

```
<area shape=rect coords="90,50,140,100"
```

```
    href="k_fruit.html"
```

```
    onMouseOver="self.status='Care and handling of t  
    ;return true">
```


```
<area shape=default
```

```
    href="javascript:window.alert('Choose the cup or
```

```
onMouseOver="self.status='Select the cup or a bo'  
;return true">  
  
</map>
```

See [Figure 6-7](#) for the results.

**Figure 6-7. A simple client-side image map with JavaScript-enabled mouse events**



figs/htm5\_0607.gif

### 6.5.6 Handling Other Browsers

Unlike its server-side `ismap` counterpart, the client-side image map tag with attribute (`<img usemap>`) doesn't need to be included in an `<a>` tag. But it may be, so that you can gracefully handle browsers that are unable to process client-side image maps.

For example, the ancient Mosaic or early versions of Netscape simply load a document named *main.html* if the user clicks the *map.gif* image referenced in the following source fragment. More recent browsers, on the other hand, divide the image into mouse-sensitive regions, as defined in the associated `<map>`, and link to a particular name anchor within the same *main.html* document if the image map region is selected by the user:

```
<a href="main.html">
```

```


</a>

...

<map name="map1">

    <area coords="0,0,49,49" href="main.html#link1">

    <area coords="50,0,99,49" href="main.html#link2">

    <area coords="0,50,49,99" href="main.html#link3">

    <area coords="50,50,99,99" href="main.html#link4">

</map>
```

To make an image map backward-compatible with all image map-capable browsers, you may also include client-side and server-side processing for the same image map. Capable browsers will honor the faster client-side processing; all other browsers will ignore the `usemap` attribute in the `<img>` tag and rely upon the referenced server process to handle user selections in the traditional way. For example:

```
<a href="/cgi-bin/images/map.proc">

</a>

...

<map name="map2">

    <area coords="0,0,49,49" href="link1.html">
```

```
<area coords="50,0,99,49" href="link2.html">  
  
<area coords="0,50,49,99" href="link3.html">  
  
<area coords="50,50,99,99" href="link4.html">  
  
</map>
```

### 6.5.7 Effective Use of Mouse-Sensitive Images

Some of the most visually compelling pages on the Web have mouse- and hot-key-sensitive images: maps with regions that (when clicked or selected with the Tab and Enter keys) lead, for example, to more information about a country or town or result in more detail about the location and who to contact at a regional branch of a business. We've seen an image of a fashion model whose various clothing parts lead to their respective catalog entries, complete with detailed descriptions and prices for ordering.

The visual nature of these "hyperactive" pictures, coupled with the need for an effective interface, means that you should strongly consider having an artist, a user-interface designer, and even a human-factors expert evaluate your imagery. At the very least, engage in a bit of user testing to make sure people know what region of the image to select to move to the desired document. Make sure the sensitive areas of the image indicate this to the user using a consistent visual mechanism. Consider using borders, drop shadows, or color changes to indicate those areas that can be selected by the user.

Finally, always remember that the decision to use images is an explicit decision to exclude text-based and image-restricted browsers from your pages. This includes browsers connecting to the Internet via slow modem connections. For these people, downloading your beautiful images is simply too expensive. To keep from disenfranchising a growing population, make sure any page that has a mouse-sensitive image has a text-only equivalent easily accessible from a link on the image-enabled

version. Some thoughtful webmasters even provide separate pages for users preferring full graphics versus mostly text.



## Chapter 8. Cascading Style Sheets

Style sheets are the way publishing professionals manage the overall "look" of their publications backgrounds, fonts, colors, and so on from a single page to huge collections of documents. Most desktop-publishing software supports style sheets, as do popular word processors, so the necessity of style sheets for HTML documents was obvious.

From the start, HTML focused on content over style. Authors are encouraged to worry about providing high-quality information and leave it to the browser to worry about presentation. We strongly urge you to adopt this philosophy in your documents don't mistake style for substance.

However, presentation is for the benefit of the reader, and even the original designers of HTML understood the interplay between style and readability for example, through the physical style and header tags. Style sheets extend that presentation with several additional effects, including colors, a wider selection of fonts, and even sounds so that users can better distinguish elements of your document. But most importantly, style sheets let you control the presentation attributes for all the tags in a document for a single document or a collection of many documents from a single master.

In early 1996, the World Wide Web Consortium (W3C) put together a draft proposal defining Cascading Style Sheets (CSS) for HTML. This draft proposal quickly matured into a recommended standard. In mid-1998, the W3C extended the original specification to create CSS2, which includes presentation standards for a variety of media besides the familiar onscreen browser, along with a several other enhancements.

Currently, no browser or web agent fully complies with the CSS2 standard. However, because we realize that eventual compliance with the W3C standard is likely, we'll cover all the components of the CSS2 standard in this chapter, even if they are not yet supported by any browser. As always, we'll denote clearly what is real, what is proposed,

and what is actually supported.<sup>[1]</sup>

[1] In the fall of 2000, work began on CSS3. As CSS3 is still under construction and browsers have not yet even become fully compliant with CSS2, we focus on CSS2 throughout this chapter.

## 3.6 HTML/XHTML Document Elements

Every HTML document should conform to the HTML SGML DTD, the formal Document Type Definition that defines the HTML standard. The DTD defines the tags and syntax that are used to create an HTML document. You can inform the browser which DTD your document complies with by placing a special SGML (Standard Generalized Markup Language) command in the first line of the document:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN">
```

This cryptic message indicates that your document is intended to be compliant with the HTML 4.01 final DTD defined by the World Wide Web Consortium (W3C). Other versions of the DTD define more restricted versions of the HTML standard, and not all browsers support all versions of the HTML DTD. In fact, specifying any other doctype may cause the browser to misinterpret your document when displaying it for the user. It's also unclear what doctype to use when including in the HTML document the various tags that are not standards but are very popular features of a popular browser the Netscape extensions, for instance, or even the deprecated HTML 3.0 standard, for which a DTD was never released.

Almost no one precedes their HTML documents with the SGML doctype command. Because of the confusion of versions and standards, we don't recommend that you include the prefix with your HTML documents either.

On the other hand, we do strongly recommend that you include the proper doctype statement in your XHTML documents, in conformance with XML standards. Read [Chapter 15](#) and [Chapter 16](#) for more about DTDs and the XML and XHTML standards.

### 3.6.1 The <html> Tag

As we saw earlier, the `<html>` and `</html>` tags serve to delimit the beginning and end of a document. Since the typical browser can easily

infer from the enclosed source that it is an HTML or XHTML document, you don't really need to include the tag in your source HTML document.

## <html>

### *Function*

Delimits a complete HTML or XHTML document

### *Attributes*

`dirlangversion`

### *End tag*

`</html>`; may be omitted in HTML

### *Contains*

*head\_tag, body\_tag, frames*

That said, it's considered good form to include this tag so that other tools, particularly more mundane text-processing ones, can recognize your document as an HTML document. At the very least, the presence of the beginning and ending `<html>` tags ensures that the beginning or the end of the document has not inadvertently been deleted. Besides, XHTML requires the `<html>` tag.

Inside the `<html>` tag and its end tag are the document's head and body. Within the head, you'll find tags that identify the document and define its place within a document collection. Within the body is the actual document content, defined by tags that determine the layout and appearance of the document text. As you might expect, the document head is contained within a `<head>` tag and the body is within a `<body>` tag, both of which are defined later.

The `<body>` tag may be replaced by a `<frameset>` tag defining one or more display frames that, in turn, contain actual document content. See [Chapter 11](#) for more information. By far, the most common form of the `<html>` tag is simply:

```
<html>
```

```
document head and body content
```

```
</html>
```

When the `<html>` tag appears without the `version` attribute, the document server and browser assume the version of HTML used in this document is supplied to the browser by the server.

### 3.6.1.1 The `dir` attribute

The `dir` attribute specifies in which direction the browser should render text within the containing element. When used within the `<html>` tag, it determines how text will be presented within the entire document. When used within another tag, it controls the text's direction for just the content of that tag.

By default, the value of this tag is `ltr`, indicating that text is presented to the user left to right. Use the other value, `rtl`, to display text right to left, for languages like Chinese or Hebrew. Of course, the results depend on your content and the browser's support of HTML 4 or XHTML. Netscape and Internet Explorer Versions 4 and earlier ignore the `dir` attribute. The HTML 4-compliant Internet Explorer Version 5 simply right-justifies `dir=rtl` text, although if you look in [Figure 3-1](#), you'll notice the browser moves the punctuation (the period) to the other side of the sentence. Internet Explorer 6 does the same thing. Netscape 6 right-justifies everything, including the ending period.

```
<html dir=rtl>
```

```
<head>
```

```
<title>Display Directions</title>
```

```
</head>
```

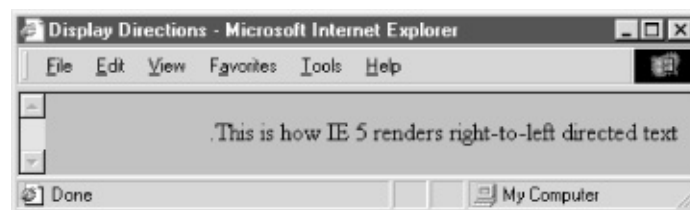
```
<body>
```

```
This is how IE 5 renders right-to-left directed text.
```

```
</body>
```

```
</html>
```

**Figure 3-1. Internet Explorer 5 implements the dir attribute**



### 3.6.1.2 The lang attribute

When included within the `<html>` tag, the `lang` attribute specifies the language you've generally used within the document. When used within other tags, the `lang` attribute specifies the language you used within that tag's content. Ideally, the browser will use `lang` to better render the text for the user.

Set the value of the `lang` attribute to an ISO-639 standard two-character language code. You may also indicate a dialect by following the ISO language code with a dash and a subcode name. For example, "en" is the ISO language code for English; "en-US" is the complete code for U.S. English. Other common language codes include "fr" (French), "de" (German), "it" (Italian), "nl" (Dutch), "el" (Greek), "es" (Spanish), "pt" (Portuguese), "ar" (Arabic), "he" (Hebrew), "ru" (Russian), "zh" (Chinese), "ja" (Japanese), and "hi" (Hindi).

### 3.6.1.3 The version attribute

The `version` attribute defines the HTML standard version used to compose the document. Its value, for HTML Version 4.01, should read exactly:

```
version="-//W3C//DTD HTML 4.01//EN"
```

In general, version information within the `<html>` tag is more trouble than it is worth, and this attribute has been deprecated in HTML 4. Serious authors should instead use an SGML `<!doctype>` tag at the beginning of their documents, like this:

```
<!DOCTYPE HTML PUBLIC "-//W3C/DTD HTML 4.01//EN"
```

```
"http://www.w3c.org/TR/html4/strict.dtd">
```



## 8.3 Style Classes

CSS2 classes allow you to create several different styles for the same element, at the document level or in an external style sheet. Later in a document, you explicitly select which style to apply to that particular instance of the tag by including the style-related `class` attribute with the `name` value of one of the previously defined styles.

### 8.3.1 Regular Classes

In a technical paper, you might want to define one paragraph style for the abstract, another for equations, and a third for centered quotations. Differentiate these paragraphs by defining each as a different style class:

```
<style type="text/css">

<!--

p.abstract {font-style: italic;

           margin-left: 0.5cm;

           margin-right: 0.5cm}

p.equation {font-family: Symbol;

           text-align: center}

h1, p.centered {text-align: center;

               margin-left: 0.5cm;

               margin-right: 0.5cm}

-->
```

```
</style>
```

Notice first in the example that defining a class is simply a matter of appending a period-separated class name as a suffix to the tag name as the selector in a style rule. Unlike the XHTML-compliant selector, which is the name of the standard tag and must be in lowercase, the class name can be any sequence of letters, numbers, and hyphens, but it must begin with a letter.<sup>[6]</sup> Careful, though: case does matter, so `abstract` is not the same as `AbsTRact`. Classes, like selectors, may be included with other selectors, separated by commas, as in the third example. The only restriction on classes is that they cannot be nested; for example, `p.equation.centered` is not allowed.

[6] Due to its support of JavaScript style sheets, Netscape 4 cannot handle class names that happen to match JavaScript keywords. The class "abstract," for instance, generates an error in Netscape 4.

Accordingly, the first rule in the example creates a class of paragraph styles named `abstract` whose text is italic and indented from the left and right margins by 0.5 centimeters. Similarly, the second paragraph style class, `equation`, instructs the browser to center the text and to use the Symbol typeface to display the text. The last style rule creates a style with centered text and 0.5-centimeter margins, applying this style to all level-1 headers as well as creating a class of the `<p>` tag named `centered` with that style.

To use a particular class of a tag, you add the `class` attribute to the tag, as in this example (rendered by Internet Explorer in [Figure 8-3](#)):

```
<p class=abstract>
```

```
This is the abstract paragraph. See how the margins a
```

```
</p>
```

```
<h3>The equation paragraph follows</h3>
```

```
<p class=equation>
```

```
a = b + 1
```


```
</p>
```

```
<p class=centered>
```

```
This paragraph's text should be centered.
```

```
</p>
```

**Figure 8-3. Use classes to distinguish different styles for the same tag**



figs/htm5\_0803.gif

For each paragraph, the value of the `class` attribute is the name of the class to be used for that tag.

### 8.3.2 Generic Classes

You may also define a class without associating it with a particular tag and apply that class selectively through your documents for a variety of tags. For example:

```
.italic {font-style: italic}
```

creates a generic class named `italic`. To use it, simply include its name with the `class` attribute. So, for instance, use `<p class=italic>` or `<h1 class=italic>` to create an italic paragraph or header.

Generic classes are quite handy and make it easy to apply a particular style to a broad range of tags. Netscape and Internet Explorer support CSS2 generic classes.

### 8.3.3 ID Classes

Almost all HTML tags accept the `id` attribute, which assigns to the element an identifier that is unique within the document. This identifier can be the target of a URL, used by automated document-processing tools, and can also be used to specify a style rule for the element.

To create a style class that the styles-conscious browser applies to only those contents of your document explicitly tagged with the `id` attribute, follow the same syntax as for style classes, except with a `#` character before the class name instead of a period. For example:

```
<style>

<!--

#yellow {color : yellow}

h1#blue {color : blue}

-->

</style>
```

Within your document, use that same `id` name to apply the style, such as `<h1 id=blue>` to create a blue heading. Or, as in the example, use `id=yellow` elsewhere in the document to turn a tag's contents yellow.

You can mix and match both `class` and `id` attributes, giving you a limited ability to apply two independent style rules to a single element.

There is a dramatic drawback to using style classes this way: the HTML and XHTML standards dictate that the value of the `id` attribute be unique for each instance in which it's used within the document. Yet here, we have to use the same value to apply the style class more than once.

Even though current browsers let you get away with it, we strongly discourage creating and using the `id` kinds of style classes. Stick to the standard style class convention to create correct, robust documents.

### 8.3.4 Pseudoclasses

In addition to conventional style classes, the CSS2 standard defines pseudoclasses, which allow you to define the display style for certain tag *states*, such as changing the display style when a user selects a hyperlink. You create pseudoclasses like regular classes, but with two notable differences: they are attached to the tag name with a colon instead of a period, and they have predefined names, not arbitrary ones you may give them. There are seven pseudoclasses, three of which are explicitly associated with the `<a>` tag.

#### 8.3.4.1 Hyperlink pseudoclasses

CSS2-compliant browsers distinguish three special states for the hyperlinks created by the `<a>` tag: not yet visited, currently being visited, and already visited. The browser may change the appearance of the tag's contents to indicate its state, such as with underlining or color. Through pseudoclasses, you can control how these states get displayed by defining styles for `a:link` (not visited), `a:active` (being visited), and `a:visited`.

The `:link` pseudoclass controls the appearance of links that are not selected by the user and have not yet been visited. The `:active`

pseudoclass defines the appearance of links that are currently selected by the user and are being processed by the browser. The `:visited` pseudoclass defines those links that the user has already visited.

To completely define all three states of the `<a>` tag, you might write:

```
a:link {color: blue}

a:active {color: red; font-weight: bold}

a:visited {color: green}
```

In this example, the styles-conscious browser is supposed to render unvisited links in blue. When the user selects a link, the browser should change its text color to red and make it bold. Once visited, the link reverts to conventional green text.

#### 8.3.4.2 Interaction pseudoclasses

The CSS2 standard defines two new pseudoclasses that, along with `:active`, relate to user actions and advise the interactive agent, such as a browser, how to display the affected element as the user interacts with the element. In other words, these two pseudoclasses `hover` and `focus` are dynamic.

For instance, when you drag the mouse over a hyperlink in your document, the browser may change the mouse-pointer icon. Hovering can be associated with a style that is in effect only while the mouse is over the element. For example, if you add the `:hover` pseudoclass to our example list of hyperlink style rules:

```
a:hover {color: yellow}
```

the text associated with unvisited links normally is rendered in blue but turns yellow when you point to it with the mouse, red while you visit it, and green after you're done visiting.

Similarly, the `:focus` pseudoclass lets you change the style for an element when it becomes the object of attention. An element may be under focus when you tab to it, click on it, or, depending on the browser, advance the cursor to it. Regardless of how the focus got to the element, the style rules associated with the focus pseudoclass are applied only while the element has the focus.

#### 8.3.4.3 Nesting and language pseudoclasses

The CSS2 `:first-child` pseudoclass lets you specify how an element may be rendered when it is the first instance, a.k.a. "child," of the containing element. For instance, the following rule gets applied to a paragraph when it is the first element of a division; there can be no intervening elements (notice the special greater-than bracket syntax relating the first child with its parent element):

```
div > p:first-child {font-style: italic}
```

Accordingly, the first paragraph in the following HTML fragment would be rendered in italics by a CSS2-compliant browser because it is the first child element of its division. Conversely, the second paragraph comes after a level-2 header, which is the first child of the second division. So, that second paragraph in the example gets rendered in plain text, because it is not the first child of its division:

```
<div>

  <p>

    I get to be in italics.

  </p>

</div>

<div>
```

```
<h2> New Division</h2>
```

```
<p>
```

```
I'm in plain text because my paragraph is a second
```

Finally, the CSS2 standard defines a new pseudoclass that lets you select an element based on its language. For instance, you might include the `lang=fr` attribute in a `<div>` tag to instruct the browser that the division contains French language text. The browser may specially treat the text. Or, you may impose a specific style with the pseudoclass `:lang`. For example:

```
div:lang(it) {font-family: Roman}
```

says that text in divisions of a document that contain the Italian language should use the Roman font family. Appropriate, don't you think? Notice that you specify the language in parentheses immediately after the `lang` keyword. Use the same two-letter ISO standard code for the pseudoclass `:lang` as you do for the `lang` attribute. [[Section 3.6.1.2](#)]

#### 8.3.4.4 Browser support of pseudoclasses

None of the popular browsers support the `:lang`, `:first-child`, or `:focus` pseudoclasses yet. All the current popular browsers support the `:link`, `:active`, `:hover`, and `:visited` pseudoclasses for the hyperlink tag (`<a>`). Even though `:active` also may be used for other elements, none of the browsers yet support applications beyond the `<a>` tag.

#### 8.3.5 Mixing Classes

You can mix pseudoclasses with regular classes by appending the pseudoclass name to the selector's class name. For example, here are some rules that define plain, normal, and fancy anchors:



```
a.plain:link, a.plain:active, a.plain:visited {color: blue}

a:link {color: blue}

a:visited {color: green}

a:active {color: red}

a.fancy:link {font-style: italic}

a.fancy:visited {font-style: normal}

a.fancy:active {font-weight: bold; font-size: 150%}
```

The `plain` version of `<a>` is always blue, no matter what the state of the link is. Accordingly, normal links start out blue, turn red when active, and convert to green when visited. The `fancy` link inherits the color scheme of the normal `<a>` tag but adds italic text for unvisited links, converts back to normal text after being visited, and actually grows 50% in size and becomes bold when active.

A word of warning about that last property of the `fancy` class: specifying a font-size change for a transient display property results in lots of browser redisplay activity when the user clicks on the link. Given that some browsers run on slow machines, this redisplay may be annoying to your readers. Given also that implementing that sort of display change is something of a pain, it is unlikely that most browsers will support radical appearance changes in `<a>` tag pseudoclasses.

### 8.3.6 Class Inheritance

Classes inherit the style properties of their generic base tags. For instance, all the properties of the plain `<p>` tag apply to a specially defined paragraph class, except where the class overrides a particular property.

Classes cannot inherit from other classes, only from the unclassed versions of the tags they represent. In general, therefore, you should put as many common styles as possible into the rule for the basic version of a tag and create classes only for those properties that are unique to that class. This makes maintenance and sharing of your style classes easier, especially for large document collections.

## 4.6 Precise Spacing and Layout

CSS notwithstanding, the original concept of HTML is for specifying document content without indicating format; to delineate the structure and semantics of a document, not how that document is to be presented to the user. Normally, you should leave word wrapping, character and line spacing, and other presentation details up to the browser. That way, the document's content its rich information, not good looks is what matters. When looks matter more, such as for commercial presentations, look to style sheets for layout control (see [Chapter 8](#)).

### 4.6.1 The `<br>` Tag

The `<br>` tag interrupts the normal line filling and word wrapping of paragraphs within an HTML or XHTML document. It has no ending tag with HTML;<sup>[4]</sup> it simply marks the point in the flow where a new line should begin. Most browsers simply stop adding words and images to the current line, move down and over to the left margin, and resume filling and wrapping.

[4] With XHTML, put the end inside the start tag: `<br />`. See [Chapter 16](#) for details.

## <br>

### *Function*

Inserts a line break into a text flow

### *Attributes*

`class, clear, id, style, title`

### *End tag*

None in HTML; `</br>` or `<br ... />` in XHTML

### *Contains*

Nothing

### *Used in*

*text*

This effect is handy when formatting conventional text with fixed line breaks, such as addresses, song lyrics, or poetry. Notice, for example, the lyrical breaks when the following source is rendered by Internet Explorer:

```
<h3>
```

```
Heartbreak Hotel</h3>
```

```
<p>
```

```
Ever since my baby left me<br>
```

```
I've found a new place to dwell.<br>
```

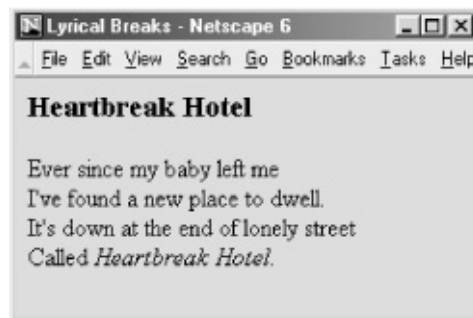
```
It's down at the end of lonely street<br>
```

Called `<cite>Heartbreak Hotel</cite>`.

`</p>`

The results are shown in [Figure 4-13](#).

**Figure 4-13. Give lyrics their breaks (`<br>`)**



Also notice how the `<br>` tag simply causes text to start a new line, while the browser, when encountering the `<p>` tag, typically inserts some vertical space between adjacent paragraphs. [[Section 4.1.2](#)]

#### 4.6.1.1 The `clear` attribute

Normally, the `<br>` tag tells the browser to stop the current flow of text immediately and resume at the left margin of the next line or against the right border of a left-justified inline graphic or table. Sometimes you'd rather the current text flow resume below any tables or images currently blocking the left or right margins.

HTML 4 and XHTML provide that capability with the `clear` attribute for the `<br>` tag. It can have one of three values `left`, `right`, or `all` each related to one or both of the margins. When the specified margin or margins are clear of images, the browser resumes the text flow.

[Figure 4-14](#) illustrates the effects of the `clear` attribute when the browser renders the following HTML fragment:

```

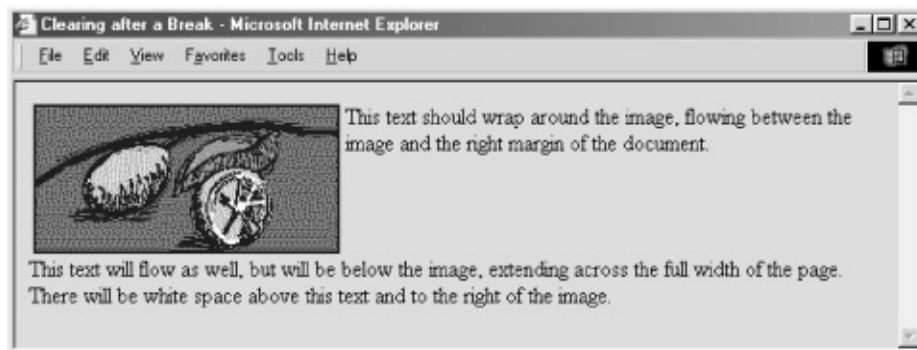
```

This text should wrap around the image, flowing between the image and the right margin of the document.

```
<br clear=left>
```

This text will flow as well, but will be below the image, extending across the full width of the page. There will be whitespace above this text and to the right of the image.

**Figure 4-14. Clearing images before resuming text flow after the `<br>` tag**



Inline images are just that normally in line with text, but usually only a single line of text. Additional lines of text flow below the image, unless that image is specially aligned by `right` or `left` attribute values for the `<img>` tag (similarly for `<table>`). Hence, the `clear` attribute for the `<br>` tag works only in combination with left- or right-aligned images or tables. [[Section 5.2.6.4](#)] [[Section 10.2.1.1](#)]

The following XHTML code fragment illustrates how to use the `<br>` tag and its `clear` attribute as well as the `<img>` tag's alignment attributes to place captions directly above, centered on the right, and below an image that is aligned against the left margin of the browser window:

Paragraph tags separate leading and following text flow from the captions.

```
<p>
```

I'm the caption on top of the image.

```
<br />
```

```

```

This one's centered on the right.

```
<br clear="left" />
```

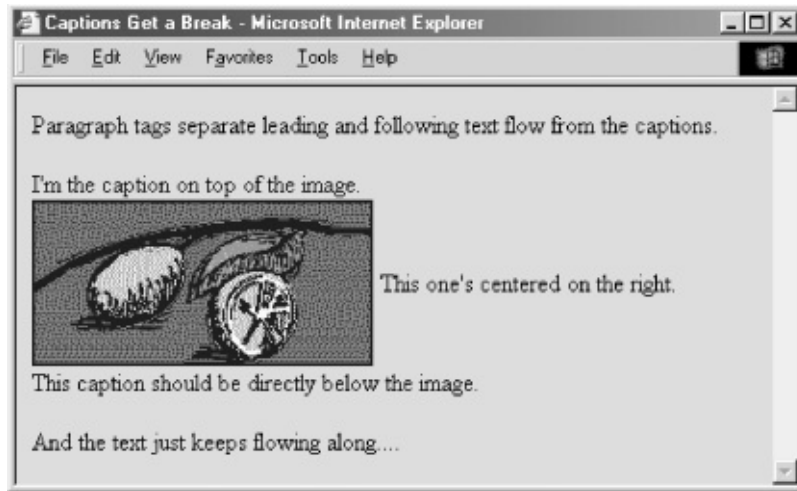
This caption should be directly below the image.

```
</p>
```

```
<p />
```

**Figure 4-15** illustrates the results of this example code.

**Figure 4-15. Captions placed on top, center-right, and below an image**



You might also include a `<br clear=all>` tag just after an `<img>` tag or table that is at the very end of a section of your document. That way, you ensure that the subsequent section's text doesn't flow up and against that image and confuse the reader. [[img](#)]

#### 4.6.1.2 The class, id, style, and title attributes

You can associate additional display rules for the `<br>` tag using style sheets. The rules can be applied to the `<br>` tag using either the `style` or `class` attribute. [[Section 8.1.1](#)] [[Section 8.3](#)]

You also may assign a unique id to the `<br>` tag, as well as a less rigorous title, using the respective attribute and accompanying quote-enclosed string value. [[Section 4.1.1.4](#)] [[Section 4.1.1.4](#)]

#### 4.6.2 The `<nobr>` Tag (Extension)

Occasionally, you may have a phrase that you want to appear unbroken on a single line in the user's browser window, even if that means the text extends beyond the visible region of the window. Computer commands are good examples. Typically, you type in a computer command even a multiword one on a single line. Because you cannot predict exactly how many words will fit inside an individual's browser window, the sequence of computer-command words may end up broken into two or more lines



of text. Command syntax is confusing enough; it doesn't need the extra cross-eyed effect of being wrapped onto two lines.

With standard HTML and XHTML, the way to make sure text phrases stay intact across the browser display is to enclose those segments in a `<pre>` tag and format it by hand. That's acceptable and nearly universal for all browsers. However, `<pre>` alters the display font from the regular text, and manual line breaks inside the `<pre>` tag are not always rendered correctly. [`<pre>`]

## **<no**br**>** □ □

### *Function*

Creates a region of nonbreaking text

### *Attributes*

None

### *End tag*

`</nobr>`; always used

### *Contains*

*text*

### *Used in*

*block*

The current browsers offer the `<nobr>` tag alternative to `<pre>`, which keeps enclosed text intact on a single line while retaining normal text style.<sup>[5]</sup> `<nobr>` makes the browser treat the tag's contents as though they were a single, unbroken word. The tag contents retain the current font style, and you can change to another style within the tag.

[5] Be aware that `<nobr>` and its colleague `<wbr>` are extensions to the language and not part of the HTML standard.

Here's the `<nobr>` tag in action with our computer command example:

When prompted by the computer, enter

```
<nobr>
```

```
<tt>find . -name \*.html -exec rm \{\}\; </tt>.
```

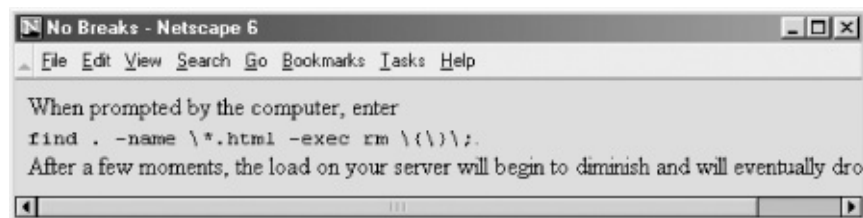
```
</nobr>
```

```
<br>
```

```
<nobr>After a few moments, the load on your server will  
to diminish and will eventually drop to zero.</nobr>
```

Notice in the example source and its display ([Figure 4-16](#)) that we've included the special `<tt>` tag inside the first `<nobr>` tag, thereby rendering the contents in monospaced font. If the `<nobr>`-tagged text cannot fit on a partially filled line of text, the extended browser precedes it with a line break, as shown in the figure. The second `<nobr>` segment in the example demonstrates that the text may extend beyond the right window boundary if the segment is too long to fit on a single line. [[Section 4.5.10](#)]

**Figure 4-16. The `<nobr>` extension suppresses text wrapping**



The `<nobr>` tag does not suspend the browser's normal line-filling process; it still collects and inserts images and believe it or not asserts forced line breaks caused by the `<br>` or `<p>` tags, for example. The `<nobr>` tag's only action is to suppress an automatic line break when the current line reaches the right margin.

In addition, you might think this tag is needed only to suppress line breaks for phrases, not a sequence of characters without spaces that can exceed the browser window's display boundaries. Today's browsers do not hyphenate words automatically, but someday soon they probably will. It makes sense to protect any break-sensitive sequences of characters

with the `<nobr>` tag.

### 4.6.3 The `<wbr>` Tag (Extension)

The `<wbr>` tag is the height of text-layout finesse, offered as an extension to the languages by the popular browsers. Used with the `<nobr>` tag, `<wbr>` advises the extended browser when it may insert a line break in an otherwise nonbreakable sequence of text. Unlike the `<br>` tag, which always causes a line break, even within a `<nobr>`-tagged segment, the `<wbr>` tag works only when placed inside a `<nobr>`-tagged content segment and causes a line break only if the current line has already extended beyond the browser's display window margins.

**<wbr>** □ □

*Function*

Defines a potential line break point if needed

*Attributes*

None

*End tag*

None in HTML; `</wbr>` or `<wbr ... />` in XHTML

*Contains*

Nothing

*Used in*

*text*

Now, `<wbr>` may seem incredibly esoteric to you, but scowl not. There may come a time when you want to make sure portions of your document appear on a single line, but you don't want to overrun the browser window margins so far that readers will have to camp on the horizontal scrollbar just to read your fine prose. By inserting the `<wbr>` tag at appropriate points in the nonbreaking sequence, you let the browser gently break the text into more manageable lines:

`<p>`

`<nobr>`

`This is a very long sequence of text that is`

`forced to be on a single line, even if doing so causes`

`<wbr>`

the browser to extend the document window beyond the size of the viewing pane and the poor user must scroll

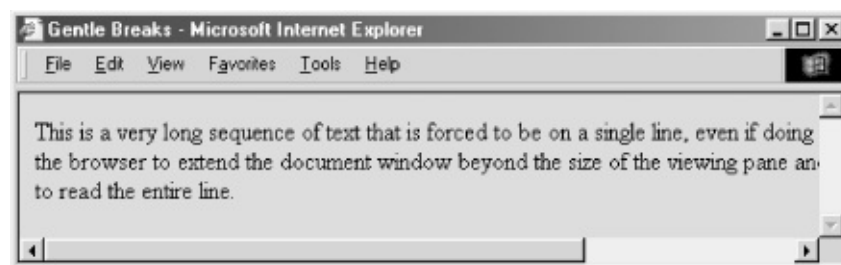
`<wbr>`

to read the entire line.

`</nobr>`

You'll notice in our rendered version ([Figure 4-17](#)) that both `<wbr>` tags take effect. By increasing the horizontal window size or reducing the font size, you may fit all of the segment before the first `<wbr>` tag within the browser window. In that case, only the second `<wbr>` would have an effect; all the text leading up to it would extend beyond the window's margins.

**Figure 4-17. Gentle line breaks with `<wbr>`**



#### 4.6.4 Better Line-Breaking Rules

Unlike some browsers, and to their credit, Netscape Navigator and Internet Explorer do not consider tags to be line-break opportunities. Consider the unfortunate consequences to your document's display if, while rendering the example segment below, the browser puts the comma adjacent to the "du" or the period adjacent to the word "df" on a separate line. Netscape and Internet Explorer will not.

Make sure you type `<tt>du</tt>`, not `<tt>df</tt>`.

### 4.6.5 The `<pre>` Tag

The HTML/XHTML standard `<pre>` tag and its required end tag (`</pre>`) define a segment inside which the browser renders text in exactly the character and line spacing written in the source document. Normal word wrapping and paragraph filling are disabled, and extraneous leading and trailing spaces are honored. Browsers display all text between the `<pre>` and `</pre>` tags in a monospaced font.

## **<pre>**

### *Function*

Renders a block of text without any formatting

### *Attributes*

`class, dir, id, lang, onClick, onDbClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, style, title, width`

### *End tag*

`</pre>;` never omitted

### *Contains*

*pre\_content*

### *Used in*

*block*

Authors most often use the `<pre>` formatting tag when the integrity of columns and rows of characters must be retained; for instance, in tables of numbers that must line up correctly. Another application for `<pre>` is to set aside a blank segment a series of blank lines in the document display, perhaps to clearly separate one content section from another or to temporarily hide a portion of the document when it first loads and is rendered by the user's browser.

Tab characters have their desired effect within the `<pre>` block, with tab stops defined at every eight character positions. We discourage their use, however, since tabs aren't consistently implemented among the various browsers. Use spaces to ensure correct horizontal positioning of text within `<pre>`-formatted text segments.



A common use of the `<pre>` tag is to present computer source code, as in the following example:

`<p>`

The processing program is:

`<pre>`

```
main(int argc, char **argv)

{
    FILE *f;

    int i;

    if (argc != 2)

        fprintf(stderr, "usage: %s <file>\n",

            argv[0]);

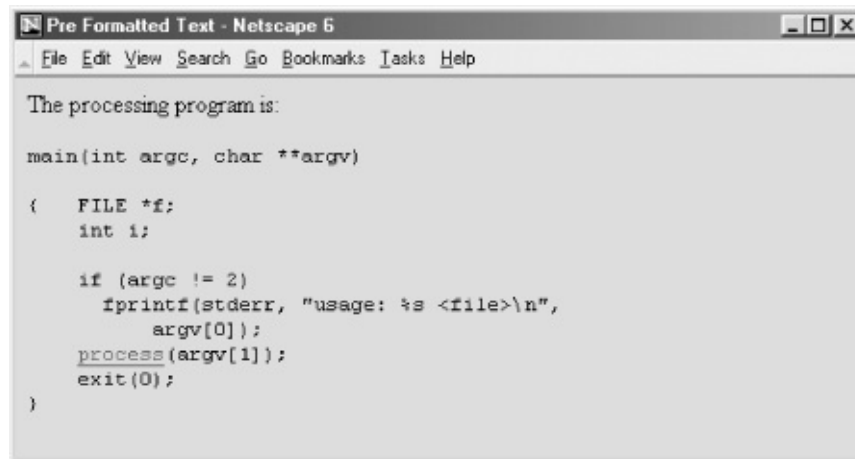
    <a href="http:process.c">process</a>(argv[1]);

    exit(0);
}

</pre>
```

The result is displayed by Netscape as shown in [Figure 4-18](#).

**Figure 4-18. Use the `<pre>` tag to preserve the integrity of columns and rows**



#### 4.6.5.1 Allowable content

The text within a `<pre>` segment may contain physical and content-based style changes, along with anchors, images, and horizontal rules. When possible, the browser should honor style changes, within the constraint of using a monospaced font for the entire `<pre>` block. Tags that cause a paragraph break (heading, `<p>`, and `<address>` tags, for example) must not be used within the `<pre>` block. Some browsers will interpret paragraph-ending tags as simple line breaks, but this behavior is not consistent across all browsers.

Style markup and other tags are allowed in a `<pre>` block, so you must use entity equivalents for the literal characters: `&lt;` for `<`, `&gt;` for `>`, and `&amp;` for the ampersand.

You place tags into the `<pre>` block as you would in any other portion of the HTML/XHTML document. For instance, study the reference to the "process" function in the previous example. It contains a hyperlink (using the `<a>` tag) to its source file, *process.c*.

#### 4.6.5.2 The width attribute

The `<pre>` tag has an optional attribute, `width`, that determines the number of characters to fit on a single line within the `<pre>` block. The

browser may use this value to select a font or font size that fits the specified number of characters on each line in the `<pre>` block. It does not mean that the browser will wrap and fill text to the specified width. Rather, lines longer than the specified width simply extend beyond the visible region of the browser's window.

The `width` attribute is only advice for the user's browser; it may or may not be able to adjust the view font to the specified width.

#### 4.6.5.3 The `dir` and `lang` attributes

The `dir` attribute lets you advise the browser which direction the text within the `<pre>` segment should be displayed in, and `lang` lets you specify the language used within that tag. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)]

#### 4.6.5.4 The `class`, `id`, `style`, and `title` attributes

Although the browsers usually display `<pre>` content in a defined style, you can override that style and add special effects, such as a background picture, by defining your own style for the tag. This new look can be applied to the `<pre>` tags using either the `style` or `class` attributes. [[Section 8.1.1](#)] [[Section 8.3](#)]

You also may assign a unique id to the `<pre>` tag, as well as a less rigorous title, using the respective attribute and accompanying quote-enclosed string value. [[Section 4.1.1.4](#)] [[Section 4.1.1.4](#)]

#### 4.6.5.5 Event attributes

As with most other tagged segments of content, user-related events can happen in and around `<pre>` content, such as when a user clicks or double-clicks within its display space. Many of these events are recognized by current browsers. With the respective "on" attribute and

value, you may react to those events by displaying a user dialog box or activating some multimedia event. [[Section 12.3.3](#)]

#### 4.6.6 The `<center>` Tag (Deprecated)

The `<center>` tag is another tag with obvious effects: its contents, including text, graphics, tables, and so on, are centered horizontally inside the browser's window. For text, this means that each line gets centered after the text flow is filled and wrapped. The `<center>` alignment remains in effect until canceled with its `</center>` end tag.

## **<center>**

### *Function*

Centers a section of text

### *Attributes*

`align, class, dir, id, lang, onClick, onDbClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, style, title`

### *End tag*

`</center>`; never omitted

### *Contains*

*body\_content*

### *Used in*

*block*

Line by line is a common, albeit primitive, way to center text, and it should be used judiciously; browsers do not attempt to balance a centered paragraph or other block-related elements, such as elements in a list, so keep your centered text short and sweet. Titles make good centering candidates; a centered list usually is difficult to follow. HTML authors commonly use `<center>` to center a table or image in the display window, too. There is no explicit center alignment option for inline images or tables, but there are ways to achieve the effect using style sheets.

Because users will have varying window widths, display resolutions, and so on, you may also want to employ the `<nobr>` and `<wbr>` extension tags (see [Section 4.6.2](#) and [Section 4.6.3](#)) to keep your centered text

intact and looking good. For example:

```
<center>

<nobr>

Copyright 1995 by QuatCo Enterprises.<br>

All rights reserved.

</nobr>

</center>
```

The `<nobr>` tags in the sample source help ensure that the text remains on a single line, and the `<br>` tag controls where the line may be broken if it exceeds the browser's display-window width.

Centering is useful for creating distinctive section headers, although you may achieve the same effect with an explicit `align=center` attribute in the respective heading tag. You might also center text using `align=center` in conjunction with the `<div>` or `<p>` tags. In general, the `<center>` tag can be replaced by an equivalent `<div align=center>` or similar tag, and its use is discouraged.

Indeed, like `<font>` and other HTML 3.2 standard tags that have fallen into disfavor in the wake of style sheets, the `<center>` tag is deprecated in the HTML 4 and XHTML standards. Nonetheless, its use in HTML documents is nearly universal, and the popular browsers are sure to support it for many revisions to come. Still, be aware of its eventual demise.

#### 4.6.6.1 The `dir` and `lang` attributes

The `dir` attribute lets you advise the browser which direction the text within the `<center>` segment should be displayed in, and `lang` lets you

specify the language used within the tag. [\[Section 3.6.1.1\]](#) [\[Section 3.6.1.2\]](#)

#### 4.6.6.2 The class, id, style, and title attributes

Use the `style` attribute to specify an inline style for the `<center>` tag, or use the `class` attribute to apply a predefined style class to the tag. [\[Section 8.1.1\]](#) [\[Section 8.3\]](#)

You may assign a unique id to the `<center>` tag, as well as a title, using the respective attribute and accompanying quote-enclosed string value. [\[Section 4.1.1.4\]](#) [\[Section 4.1.1.4\]](#)

#### 4.6.6.3 Event attributes

As with most other tagged segments of content, user-related events can happen in and around the `<center>` tag, such as when a user clicks or double-clicks within its display space. Many of these events are recognized by the current browsers. With the respective "on" attribute and value, you may react to those events by displaying a user dialog box or activating some multimedia event. [\[Section 12.3.3\]](#)

### 4.6.7 The `<listing>` Tag (Obsolete)

The `<listing>` tag is an obsolete tag, explicitly removed from the HTML 4 standard, meaning that you shouldn't use it. We include it here for historical reasons, since it is supported by some browsers and has the same effect on text formatting as the `<pre>` tag with a specified width of 132 characters.

## <listing>

### *Function*

Renders a block of text without any formatting

### *Attributes*

`class (□, □), style (□, □)`

### *End tag*

`</listing>;` never omitted

### *Contains*

*literal\_text*

### *Used in*

*block*

The only difference between `<pre>` and `<listing>` is that no other markup is allowed within the `<listing>` tag, so you don't have to replace the literal `<`, `>`, and `&` characters with their entity equivalents in a `<listing>` block, as you must inside a `<pre>` block.

Since the `<listing>` tag is the same as a `<pre width=132>` tag, and because it might not be supported in later versions of the popular browsers, we recommend that you stay away from using `<listing>`.

## 4.6.8 The `<xmp>` Tag (Obsolete)

Like the `<listing>` tag, the `<xmp>` tag is obsolete and should not be used. We include it here mostly for historical reasons.



## <xmp>

### Function

Renders a block of text without any formatting

### Attributes

`class (□, □), style (□, □)`

### End tag

`</xmp>;` never omitted

### Contains

*literal\_text*

### Used in

*block*

The `<xmp>` tag formats text just like the `<pre>` tag with a specified width of 80 characters. However, unlike the `<pre>` tag, you don't have to replace the literal `<`, `>`, and `&` characters with their entity equivalents within an `<xmp>` block. The name `<xmp>` is short for "example"; the language's designers intended that the tag be used to format examples of text originally displayed on 80-column-wide displays. Because the 80-column display has mostly gone the way of green screens and teletypes and the effect of an `<xmp>` tag is basically the same as `<pre width=80>`, don't use `<xmp>`; it may disappear in subsequent versions of HTML.

## 4.6.9 The `<plaintext>` Tag (Obsolete)

Throw the `<plaintext>` tag out of your bag of HTML tricks; it's

obsolete, like `<listing>` and `<xmp>`. Included here for historical reasons, authors once used `<plaintext>` to tell the browser to treat the rest of your document's text just as written, with no markup allowed. There was no ending tag for `<plaintext>` (of course, no markup!), but there was an end to `<plaintext>`. Forget about it.

## <plaintext>

### *Function*

Renders a block of text without any formatting

### *Attributes*

None

### *End tag*

None

### *Contains*

*literal\_text*

### *Used in*

*block*

## Chapter 16. XHTML

Despite its name, you don't use Extensible Markup Language (XML) to directly create and mark up web documents. Instead, you use XML technology to define a new markup language, which you then use to mark up web documents. This should come as no surprise to anyone who has read the previous chapter in this book. Nor, then, should it surprise you that one of the first languages defined using XML is an XML-ized version of HTML, the most popular markup language ever. HTML is being disciplined and cleaned up by XML, to bring it back into line with the larger family of markup languages. This standard is XHTML 1.0.<sup>[1]</sup>

[1] Throughout this chapter, we use "XHTML" to mean the XHTML 1.0 standard. There is a nascent XHTML 1.1 standard that diverges from HTML 4.01 and is more restrictive than XHTML 1.0. We describe the salient features of XHTML 1.1 in [Section 16.4](#).

Because of HTML's legacy features and oddities, using XML to describe HTML was not an easy job for the W3C. In fact, certain HTML rules, as we'll discuss later, cannot be represented using XML. Nonetheless, if the W3C has its way, XHTML will ultimately replace the HTML we currently know and love.

So much of XHTML is identical to HTML's current standard, Version 4.01, that almost everything presented elsewhere in this book can be applied to both HTML and XHTML. The differences, both good and bad, are detailed in this chapter. To become fluent in XHTML, you'll first need to absorb the rest of this book, and then adjust your thinking to embrace what we present in this chapter.

## 12.4 JavaScript Style Sheets (Antiquated)

Much of a browser's work is manipulating the display, and much of its display code already has been exposed for JavaScripting. So it seemed only natural, perhaps even relatively easy, for the developers at Netscape to implement JavaScript Style Sheets ( JSS). Based on the W3C-recommended Cascading Style Sheet (CSS) model, outlined in [Chapter 8](#), this alternative document style technology lets you prescribe display properties for all the various HTML elements, either inline as tag attributes, at the document level, or for an entire document collection.

JSS is a Netscape invention. In fact, for a short time, Netscape appeared ready to eschew the CSS methodology, which Internet Explorer already had implemented, and use JSS exclusively for HTML document designers with its then-current browser, Navigator 4. In the end, Netscape supported both JSS and CSS technologies. Today, Netscape 6 eschews support for JSS entirely in favor of the standard CSS2. At this point, CSS should be seen as Netscape's long-term direction.

We are strong proponents of reasonable standards, and now that the CSS2 model is fully supported in HTML 4 and XHTML, we can't recommend that you use anything but CSS-standard style sheets. Evidently, Netscape now agrees with us on this point.

We thoroughly discuss the concepts and ideas behind style sheets specifically, Cascading Style Sheets in [Chapter 8](#), so we won't repeat ourselves here. Rather, we address only how to create and manipulate styles with JavaScript (purely for historical reasons, since no current browser supports them). Before forging ahead in this section, we recommend that you first absorb the information in [Chapter 8](#).

### 12.4.1 JavaScript Style Sheet Syntax

Netscape Versions 4 and earlier implement JSS by extending several existing HTML tags and defining a few objects that store your document's

styles. Netscape 6 no longer supports JSS.

### 12.4.1.1 External, document-level, and inline JSS

As with CSS, you can reference and load external JSS files with the `<link>` tag. For example:

```
<link href="styles.js" rel=stylesheet type=text/JavaScript>
```

The only real difference between this tag and the one for a CSS external style sheet is that the `type` attribute of the `<link>` tag is set to `text/JavaScript` instead of `text/CSS`. The referenced file, *styles.js*, contains JavaScript statements that define styles and classes that Netscape then uses to control display of the current document.

Document-level JSS is defined within a `<style>` tag in the `<head>` of the document, just like with CSS. Again, there is only one real difference: the `type` attribute of the `<style>` tag is set to `text/JavaScript` instead of `text/CSS`.

The contents of the `<style>` tag for JSS are quite different from those for CSS, however. For example:

```
<style type=text/JavaScript>

<!--

    tags.BODY.marginLeft = "20px";

    tags.P.fontWeight = "bold";

// -->

</style>
```

First, notice that we use the standard JavaScript and HTML comments to surround our JSS definitions, preventing noncompliant browsers from

processing them as HTML content. Also notice that the syntax of the style definition is that of JavaScript, where letter case, among other things, *does* make a difference.

You associate inline JavaScript-based style rules with a specific tag using the `style` attribute, just like with CSS inline styles. The value of the attribute is a list of JSS assignments, separated by semicolons. For example:

```
<p style="color = 'green'; fontWeight = 'bold'">
```

creates a green, bold-faced text paragraph. Notice first that you need to enclose inline style values within single quotation marks, not double quotation marks, as you might use for document-level and external JSS styles. This is reasonable, since the `style` attribute value itself must be enclosed in double quotation marks.

Also note that inline JSS definitions use only the property name, not the containing tag object that owns the property. This makes sense, since inline JSS styles affect only the current tag, not all instances of the tag.

#### 12.4.1.2 JSS values

In general, all of the values you may use for CSS may also be used in JSS definitions. For keyword, length, and percentage values, simply enclose the value in quotes and use it as you would any string value in JavaScript. Thus, the CSS value `bold` becomes `"bold"` or `'bold'` for JSS document-level or inline styles, respectively; `12pt` in CSS becomes `'12pt'` or `"12pt"` in JSS.

Specify color values as the color name or a hexadecimal color value, enclosed in single or double quotes. The CSS decimal RGB notation is not supported in JSS.

JSS URL values are strings containing the desired URL. Thus, the CSS URL value `url(http://www.kumquat.com)` becomes `'http://www.kumquat.com'` for a JSS inline style, or

"http://www.kumquat.com" at the document level.

One unique power of JSS is that any value can be computed dynamically when the document is processed by the browser. Instead of statically specifying the font size, for example, you can compute it on the fly:

```
tags.P.fontSize = favorite_font_size( );
```

We assume that the JavaScript function `favorite_font_size( )` somehow determines the desired font size and returns a string value containing that size. This, in turn, is assigned to the `fontSize` property for the `<p>` tag, defining the font size for all paragraphs in the document.

### 12.4.1.3 Defining styles for tags

JavaScript defines a document property called `tags` that contains the style properties for all HTML tags. To define a style for a tag, simply set the appropriate property of the desired style property within the `tag` property of the `document` object. For example:

```
document.tags.P.fontSize = '12pt';
```

```
document.tags.H2.color = 'blue';
```

These two JSS definitions set the font size for the `<p>` tag to 12 points and render all `<h2>` tags in blue. The equivalent CSS definitions are:

```
p {font-size : 12pt}
```

```
h2 {color : blue}
```

Since the `tags` property always refers to the current document, you may omit `document` from any JSS tag style definition. We could have written the previous two styles as:

```
tags.P.fontSize = '12pt';
```



```
tags.H2.color = 'blue';
```

Moreover, as we mentioned previously, you may omit the tag name, as well as the `document` and `tags` properties for inline JSS using the `style` attribute.

Capitalization and case are significant in JSS. The tag names within the `tags` property must always be fully capitalized. The embedded capital letters within the tag properties are significant: any deviation from the exact lettering produces an error, and Netscape won't honor your JSS declaration. All of the following JSS definitions are invalid, though the reasons are not overly apparent:

```
tags.p.fontSize = '12pt';
```

```
tags.Body.Color = 'blue';
```

```
tags.P.COLOR = 'red';
```

The correct versions are:

```
tags.P.fontSize = '12pt';
```

```
tags.BODY.color = 'blue';
```

```
tags.P.color = 'red';
```

It can be very tedious to specify a number of properties for a single tag, so you can take advantage of the JavaScript `with` statement to reduce your typing burden. These styles:

```
tags.P.fontSize = '14pt';
```

```
tags.P.color = 'blue';
```

```
tags.P.fontWeight = 'bold';
```

```
tags.P.leftMargin = '20%';
```

can more easily be written as:

```
with (tags.P) {  
  
    fontSize = '14pt';  
  
    color = 'blue';  
  
    fontWeight = 'bold';  
  
    leftMargin = '20%';  
  
}
```

You can apply similar styles to diverse tags just as easily:

```
with (tags.P, tags.LI, tags.H1) {  
  
    fontSize = '14pt';  
  
    color = 'blue';  
  
    fontWeight = 'bold';  
  
    leftMargin = '20%';  
  
}
```

#### 12.4.1.4 Defining style classes

Like CSS, JSS lets you target styles for specific ways that a tag can be used in your document. JSS uses the `classes` property to define separate styles for the same tag. There are no predefined properties within the `classes` property; instead, any property you reference is defined as a class to be used by the current document. For example:

```
classes.bold.P.fontWeight = 'bold';
```

```
with (classes.abstract.P) {  
  
    leftMargin = '20pt';  
  
    rightMargin = '20pt';  
  
    fontStyle = 'italic';  
  
    textAlign = 'justify';  
  
}
```

The first style defines a class of the `<p>` tag named `bold` whose font weight is set to bold. The next style uses the `with` statement to create a class of the `<p>` tag named `abstract` with the specified properties. The equivalent CSS rules would be:

```
P.bold {font-weight : bold}  
  
P.abstract {left-margin : 20pt;  
  
    right-margin : 20pt;  
  
    font-style : italic;  
  
    text-align : justify  
  
}
```

Once defined, use a JSS class just like any CSS class: with the `class` attribute and the class name.

Like CSS, JSS also lets you define a class without defining the tag that uses the class. This lets you define generic classes that you can later apply to any tag. To create a generic style class in JSS, use the special tag property `all`:

```
classes.green.all.color = "green";
```

You can then add `class="green"` to any tag to have Netscape render its contents in green. The equivalent CSS is:

```
.green {color : green}
```

### 12.4.1.5 Using contextual styles

One of the most powerful aspects of CSS is its contextual style capability, wherein the browser applies a style to tags only if they appear in the document in a certain nesting. JSS supports contextual styles as well, through the special `contextual( )` method within the `tags` property. The parameters to this method are the tags and classes that define the context in which Netscape applies the style. For example:

```
tags.contextual(tags.UL, tags.UL, tags.LI).listStyleType = 'disc';
```

defines a context wherein the elements (`tags.LI`) of an unordered list nested within another unordered list (`tags.UL, tags.UL`) use the disc as their bullet symbol. The CSS equivalent is:

```
ul ul li {list-style-type : disc}
```

You can mix tags and classes in the `contextual( )` method. For instance:

```
tags.contextual(classes.abstract.P, tags.EM).color = 'red';
```

tells the browser to display in red `<em>` tags that appear within paragraphs that are of the `abstract` class. The CSS equivalent is:

```
p.abstract em {color : red}
```

Since the `tags` object is unambiguously included within the `contextual( )` method, you may omit it from the definition. Hence, our nested list example may be rewritten as:

```
tags.contextual(UL, UL, LI).listStyleType = 'disc';
```

## 12.4.2 JavaScript Style Sheet Properties

A subset of the CSS style properties are supported in JSS. The JSS style properties, their CSS equivalents, and the sections in which those properties are fully documented are shown in [Table 12-2](#).

**Table 12-2. JSS properties and CSS equivalents**

JSS property	CSS property	See section
<code>align</code>	<code>float</code>	<a href="#">Section 8.4.7.9</a>
<code>backgroundImage</code>	<code>background-image</code>	<a href="#">Section 8.4.5.3</a>
<code>backgroundColor</code>	<code>background-color</code>	<a href="#">Section 8.4.5.2</a>
<code>borderBottomWidth</code>	<code>border-bottom-width</code>	<a href="#">Section 8.4.7.4</a>
<code>borderLeftWidth</code>	<code>border-left-width</code>	<a href="#">Section 8.4.7.4</a>
<code>borderRightWidth</code>	<code>border-right-width</code>	<a href="#">Section 8.4.7.4</a>
<code>borderStyle</code>	<code>border-style</code>	<a href="#">Section 8.4.7.5</a>
<code>borderTopWidth</code>	<code>border-top-width</code>	<a href="#">Section 8.4.7.4</a>
<code>clear</code>	<code>clear</code>	<a href="#">Section 8.4.7.7</a>
<code>display</code>	<code>display</code>	<a href="#">Section 8.4.10.1</a>

fontSize	font-size	<a href="#">Section 8.4.3.2.</a>
fontStyle	font-style	<a href="#">Section 8.4.3.5</a>
height	height	<a href="#">Section 8.4.7.10</a>
lineHeight	line-height	<a href="#">Section 8.4.6.2</a>
listStyleType	list-style-type	<a href="#">Section 8.4.8.3</a>
marginBottom	margin-bottom	<a href="#">Section 8.4.7.11</a>
marginLeft	margin-left	<a href="#">Section 8.4.7.11</a>
marginRight	margin-right	<a href="#">Section 8.4.7.11</a>
marginTop	margin-top	<a href="#">Section 8.4.7.11</a>
paddingBottom	padding-bottom	<a href="#">Section 8.4.7.12</a>
paddingLeft	padding-left	<a href="#">Section 8.4.7.12</a>
paddingRight	padding-right	<a href="#">Section 8.4.7.12</a>
paddingTop	padding-top	<a href="#">Section 8.4.7.12</a>
textDecoration	text-decoration	<a href="#">Section 8.4.6.4</a>
textTransform	text-transform	<a href="#">Section 8.4.6.7</a>

<code>textAlign</code>	<code>text-align</code>	<a href="#">Section 8.4.6.3</a>
<code>textIndent</code>	<code>text-indent</code>	<a href="#">Section 8.4.6.5</a>
<code>verticalAlign</code>	<code>vertical-align</code>	<a href="#">Section 8.4.6.7</a>
<code>whiteSpace</code>	<code>white-space</code>	<a href="#">Section 8.4.10.2</a>
<code>width</code>	<code>width</code>	<a href="#">Section 8.4.7.16</a>

JSS also defines three methods that allow you to define margins, padding, and border widths within a single style property. The three methods, `margins( )`, `padding( )`, and `borderWidths( )`, accept four parameters, corresponding to the top, right, bottom, and left margin, padding, or border width, respectively. Unlike their CSS counterparts (`margin`, discussed in [Section 8.4.7.11](#); `padding`, discussed in [Section 8.4.7.12](#); and `border-width`, discussed in [Section 8.4.7.4](#)), these JSS methods require that you always specify all four parameters. There is no shorthand way in JSS to set multiple margins, paddings, or border widths with a single value.

## 8.4 Style Properties

At the heart of the CSS2 specification are the many properties that let you control how the styles-conscious browser presents your documents to the user. The standard collects these properties into six groups: fonts, colors and backgrounds, text, boxes and layout, lists, and tag classification. We'll stick with that taxonomy and preface the whole shebang with a discussion of property values and inheritance before diving into the properties themselves.

You'll find a summary of the style properties in [Appendix C](#).

### 8.4.1 Property Values

Most properties set a value to some characteristic of your document for rendering by the browser; for example, the size of the characters in a font or the color of level-2 headers. As we discussed earlier, when describing the syntax of styles, you give value to a CSS2 property by following the property's keyword with a colon (:) and one or more space- or comma-separated numbers or value-related keywords. For example:

```
color:blue
```

```
font-family: Helvetica, Univers, sans-serif
```

`color` and `font-family` are the properties in these two style examples; `blue` and the various comma-separated font names are their values, respectively.

There are eight kinds of property values: keywords, length values, percentage values, URLs, colors, angles, time, and frequencies.

#### 8.4.1.1 Keyword property values



A property may have a `keyword` value that expresses action or dimension. For instance, the effects of `underline` and `line-through` are obvious property values. And you can express property dimensions with such keywords as `small` and `xx-large`. Some keywords are even relational: `bolder`, for instance, is an acceptable value for the `font-weight` property. Keyword values are not case-sensitive: `Underline`, `UNDERLINE`, and `underline` are all acceptable keyword values.

#### 8.4.1.2 Length property values

So-called `length` values (a term taken from the CSS2 standard) explicitly set the size of a property. They are numbers, some with decimals, too. Length values may have a leading + or - sign to indicate that the value is to be added to or subtracted from the immediate value of the property. Length values must be followed immediately by a two-letter unit abbreviation, with no intervening spaces.

There are three kinds of length-value units: relative, pixels, and absolute. Relative units specify a size that is relative to the size of some other property of the content. Currently, there are only two relative units: `em`, which is the width of the lowercase letter "m" in the current font; and x-height, abbreviated `ex`, which is the height of the letter "x" in the current font.

Pixels are the tiny dots of colored light that make up the onscreen text and images on a computer-monitor or TV image. The pixels unit, abbreviated `px`, is equal to the minute size of 1 pixel, so you may express the size of some properties by how many pixels across or down they run.

Absolute property value units are more familiar to us all. They include inches (`in`), centimeters (`cm`), millimeters (`mm`), points (`pt`;  $\frac{1}{72}$  of an inch), and picas (`pc`; 12 points).

All of the following are valid length values, although not all units are recognized by the current styles-conscious browsers:

`1in`

`1.5cm`

`+0.25mm`

`-3pt`

`-2.5pc`

`+100em`

`-2.75ex`

`250px`

### **8.4.1.3 Percentage property values**

Similar to the relative length-value type, a percentage value describes a proportion relative to some other aspect of the content. It has an optional sign, meaning it may be added to or subtracted from the current value for that property, and optional decimal portion to its numeric value.

Percentage values have the percent sign (%) suffix. For example:

```
line-height: 120%
```

computes the separation between lines to be 120% of the current line height (usually relative to the text font height). Note that this value is not dynamic: changes made to the font height after the rule has been processed by the browser do not affect the computed line height.

### **8.4.1.4 URL property values**

Some properties also accept, if not expect, a URL as a value. The syntax for a CSS2 URL property value is different from that in HTML/XHTML:

```
url(service://server.com/pathname)
```

With CSS2 properties, the keyword `url` is required, as are the opening and closing parentheses. Do not leave any spaces between `url` and the opening parenthesis. The `url` value may contain either an absolute or a relative URL. However, note that the URL is relative to the style sheet's URL. This means that if you use a `url` value in a document-level or inline style, the URL is relative to the HTML document containing the style document. Otherwise, the URL is relative to the `@import`ed or `<link>`ed external style sheet's URL.

#### 8.4.1.5 Color property values

Color values specify colors in a property (surprised?). You can specify a color as a color name or a hexadecimal RGB triple, as for common HTML/XHTML attributes, or as a decimal RGB triple unique to style properties. Both color names and hexadecimal RGB triple notation are described in [Appendix G](#).

With CSS2, too, you may assign just one hexadecimal digit instead of two to the red, green, and blue (RGB) components of a color. That digit is simply doubled to create a conventional six-digit triple. Thus, the color `#78C` is equivalent to `#7788CC`. In general, three-digit color values are handy only for simple colors.

The decimal RGB triple notation is unique:

```
rgb(red, green, blue)
```

The *red*, *green*, and *blue* intensity values are decimal integers in the range 0 to 255, or integer percentages. As with a URL value, do not leave any spaces between `rgb` and the opening parenthesis.

For example, in decimal RGB convention, the color white is `rgb(255, 255, 255)` or `rgb(100%, 100%, 100%)`, and a medium yellow is `rgb(127, 127, 0)` or `rgb(50%, 50%, 0%)`.

#### 8.4.1.6 Angle, time, and frequency property values

A few properties require a value that expresses an angle, such as the heading of a compass. These properties take a numeric value followed by the units `deg` (degrees), `grad` (gradations), or `rad` (radians).

Similarly, express time values as numbers followed by either `ms` (milliseconds) or `s` (seconds) units.

Finally, frequency values are numbers followed by `Hz` (Hertz) or `kHz` (kiloHertz). Interestingly, there is no corresponding `mHz` unit, because frequencies in CSS2 refer to audio, not TV, radio broadcast, or other electromagnetic waves.

#### 8.4.2 Property Inheritance

In lieu of a specific rule for a particular element, properties and their values for tags within tags are inherited from the parent tag. Thus, setting a property for the `<body>` tag effectively applies that property to every tag in the body of your document, except for those that specifically override it. So, to make all the text in your document blue, you need only say:

```
body {color: blue}
```

rather than creating a rule for every tag you use in your document.

This inheritance extends to any level. If you later created a `<div>` tag with text of a different color, the styles-conscious browser would display all the text contents of the `<div>` tag and all its enclosed tags in that new color. When the `<div>` tag ends, the color reverts to that of the containing `<body>` tag.

In many of the following property descriptions, we refer to the tag containing the current tag as the "parent element" of that tag.

#### 8.4.3 Font Properties

The loudest complaint that we hear about HTML and its progeny, XHTML, is that they lack font styles and characteristics that even the simplest of text editors implement. The various `<font>` attributes address part of the problem, but they are tedious to use, because each text font change requires a different `<font>` tag.

Style sheets change all that, of course. The CSS2 standard provides seven font properties that modify the appearance of text contained within the affected tag: `font-family`, `font-size`, `font-size-adjust`, `font-style`, `font-variant`, `font-stretch`, and `font-weight`. In addition, there is a universal `font` property in which you can declare all the font values.

Please be aware that style sheets cannot overcome limitations of the user's display/document-rendering system, and the browser cannot conjure effects if the fonts it uses do not provide the means.

#### 8.4.3.1 The font-family property

The `font-family` property accepts a comma-separated list of font names. The browser uses the first font named in the list that also is installed and available for display on the client machine for text display.

Font-name values are for specific font styles, such as Helvetica or Courier, or a generic font style, as defined by the CSS2 standard: `serif`, `sans-serif`, `cursive`, `fantasy`, or `monospace`. The browser defines which font it actually uses for each generic font. For instance, Courier is the most popular choice for a monospace font.

Because fonts vary wildly among browsers, you should usually provide several choices when specifying a font style, ending with a suitable generic font. For example:

```
h1 {font-family: Helvetica, Univers, sans-serif}
```

causes the browser to look for and use Helvetica, and then Univers. If neither font is available for the client display, the browser uses the

generic sans-serif typeface.

Enclose font names that contain spaces New Century Schoolbook, for example in quotation marks. For example:

```
p {font-family: Times, "New Century Schoolbook", Palatino}
```

With inline styles, that extra set of double quotation marks causes problems. The solution is to use single quotation marks in an inline style:

```
<p style="font-family: Times, 'New Century Schoolbook'"
```

In practice, you don't have to use quotation marks, because font-name values are comma-separated, so the browser normally ignore spaces. Hence:

```
p {font-family: Times, New Century Schoolbook, Palatino}
```

```
<p style="font-family: Times, New Century Schoolbook, Palatino"
```

are both legal. Nonetheless, we recommend that you use quotation marks. It's a good habit to get into, and it makes things that much less ambiguous.

### 8.4.3.2 The font-size property

The `font-size` property lets you prescribe absolute or relative length values, percentages, and keywords to define the font size. For example:

```
p {font-size: 12pt}
```

```
p {font-size: 120%}
```

```
p {font-size: +2pt}
```

```
p {font-size: medium}
```

```
p {font-size: larger}
```

The first rule is probably the most used, because it is the most familiar: it sets the font size for text enclosed in your document's paragraph(s) to a specific number of points (12 in this example). The second example rule sets the font size to be 20% larger than the parent element's font size. The third increases the font's normal size by 2 points.

The fourth example selects a predefined font size set by the browser, identified by the `medium` keyword. Valid absolute-size keywords are `xx-small`, `x-small`, `small`, `medium`, `large`, `x-large`, and `xx-large`; these usually correspond to the seven font sizes used with the `size` attribute of the `<font>` tag.

The last `font-size` rule selects the next size larger than the font associated with the parent element. Thus, if the size were normally `medium`, it would be changed to `large`. You can also specify `smaller`, with the expected results.

None of the current browsers handles the incremented font size correctly. Rather, they ignore the increment/decrement sign and use its value as an absolute size. For instance, in the middle example in this section, the font size would end up as 2 points, not 2 points larger than the normal size.

#### 8.4.3.3 The `font-stretch` property

In addition to different sizes, font families sometimes contain condensed and expanded versions, in which the characters are squeezed or stretched, respectively. Use the `font-stretch` property to choose more compressed or stretched-out characters from your font.

Use the property value of `normal` to select the normal-sized version of the font. The relative values `wider` and `narrower` select the next-wider or next-narrower variant of the font's characters, respectively, but not wider or narrower than the most ("ultra") expanded or contracted one in the family.

The remaining `font-stretch` property values choose specific variants from the font family. Starting from the most condensed and ending with the most expanded, the values are `ultra-condensed`, `extra-condensed`, `condensed`, `semi-condensed`, `semi-expanded`, `expanded`, `extra-expanded`, and `ultra-expanded`.

The `font-stretch` property, of course, assumes that your display fonts support stretchable fonts. Even so, the currently popular browsers ignore this property.

#### 8.4.3.4 The `font-size-adjust` property

Without too many details, the legibility and display size of a font depend principally on its *aspect ratio*: the ratio of its rendered size to its x-height, which is a measure of the font's lowercase glyph height. Fonts with aspect ratios approaching 1.0 tend to be more legible at smaller sizes than fonts with aspect ratios approaching 0.

Also, because of aspect ratios, the actual display size of one font may appear smaller or larger than another font at the same size. So, when one font is not available for rendering, the substituted font may distort the presentation.

The `font-size-adjust` property lets you readjust the substituted font's aspect ratio so that it better fits the display. Use the property value of `none` to ignore the aspect ratio. Otherwise, include your desired aspect ratio (a decimal value less than one), typically the aspect ratio for your first-choice display font. The styles-conscious browser computes and displays the substituted font at a size adjusted to your specified aspect ratio:

$$s = (n/a) * fs$$

where `s` is the new, computer font size for display of the substituted font, calculated as the `font-size-adjust` value `n` divided by the substituted font's aspect ratio `a` times the current font size `fs`.



For example, let's imagine that your first-choice font is Times New Roman, which has an aspect ratio of 0.45. If it's not available, the browser may then substitute Comic Sans MS, which has an aspect ratio of 0.54. So that the substitution maintains nearly equivalent sizing for the font display say, at an 18-px font size with the `font-size-adjust` property set to 0.45, the CSS2-compliant browser would display or print the text with the substituted Comic Sans MS font at the smaller size of  $(0.45/0.54 \times 18 \text{ px}) = 15 \text{ px}$ .

Unfortunately, we can't show you how the popular browsers would do this, because they don't support it.

#### 8.4.3.5 The `font-style` property

Use the `font-style` property to slant text. The default style is `normal` and may be changed to `italic` or `oblique`. For example:

```
h2 {font-style: italic}
```

makes all level-2 header text italic. Netscape 4 supports only the `italic` value for `font-style`; Netscape 6 and Internet Explorer 4 and later support both values, although it is usually difficult to distinguish italic from oblique.

#### 8.4.3.6 The `font-variant` property

The `font-variant` property lets you select a variant of the desired font. The default value for this property is `normal`, indicating the conventional version of the font. You may also specify `small-caps` to select a version of the font in which the lowercase letters have been replaced with small capital letters.

Netscape 6 and Internet Explorer 6 support this property. Internet Explorer versions 4 and 5 incorrectly display `small-caps` as all uppercase letters.

### 8.4.3.7 The font-weight property

The `font-weight` property controls the weight or boldness of the lettering. The default value of this property is `normal`. You may specify `bold` to obtain a bold version of a font or use the relative `bolder` and `lighter` values to obtain a version of the font that is bolder or lighter than the parent element's font.

To specify varying levels of lightness or boldness, set the value to a multiple of 100, between the values `100` (lightest) and `900` (boldest). The value `400` is equal to the `normal` version of the font, and `700` is the same as specifying `bold`.

Internet Explorer and Netscape 6 fully support this property.

### 8.4.3.8 The font property

More often than not, you'll find yourself specifying more than one font-related property at a time for a tag's text content display. A complete font specification can get somewhat unwieldy. For example:

```
p {font-family: Times, Garamond, serif;
    font-weight: bold;
    font-size: 12pt;
    line-height: 14pt}
```

To mitigate this troublesome and potentially unreadable collection, use the comprehensive `font` property and group all the attributes into one set of declarations:

```
p {font: bold 12pt/14pt Times, Garamond, serif}
```

The grouping and ordering of font attributes is important within the `font`

property. The font style, weight, and variant attributes must be specified first, followed by the font size and the line height separated by a slash character, and ending with the list of font families. Of all the properties, the size and family are required; the others may be omitted.

Here are some more sample `font` properties:

```
em {font: italic 14pt Times}
```

```
h1 {font: 24pt/48pt sans-serif}
```

```
code {font: 12pt Courier, monospace}
```

The first example tells the styles-conscious browser to emphasize `<em>` text using a 14-point italic Times face. The second rule has `<h1>` text displayed in the boldest 24-point sans-serif font available, with an extra 24 points of space between the lines of text. Finally, text within a `<code>` tag is set in 12-point Courier or the browser-defined monospace font.

We leave it to your imagination to conjure up examples of the abuses you could foster with font styles. Perhaps a recent issue of *Wired* magazine, notorious for avant-garde fonts and other print-related abuses, would be helpful in that regard?

## 8.4.4 Font Selection and Synthesis

The original cascading style sheet standard, CSS1, had a simplistic font-matching algorithm: if your specified font does not exist in the local client's font collection, substitute a generic font. Of course, the results are often less than pleasing to the eye and can wreak havoc with the display. Moreover, there are often more suitable font substitutes than generic ones. The CSS2 standard significantly extends the CSS1 font-matching model and includes a new at-rule that lets authors define, download, and use new fonts in their documents.

### 8.4.4.1 CSS2 font-matching steps

The CSS2 font-matching algorithm has four steps. The first step is simply to use the specified font when it is found on the user's machine; this could be one of several font families specified in the style-sheet rule, parsed in their order of appearance.

The second step, taken when none of the fonts specified in the rule exist on the user's machine, has the browser attempt to find a close match among similar local fonts. For example, a request for Helvetica might wind up using Arial, a similar sans-serif font.

The third step in the CSS2 font-matching algorithm has the browser try to synthesize a font, taking a local font and changing it to match the specified one. For example, a request for 72-point Helvetica might be satisfied by taking the local 12-point Arial font and scaling it up to match the desired size.

Failing all, the browser may take a fourth step and download the desired font, provided the author has supplied suitable external font definitions. These external font definitions are created with the `@font-face` at-rule, whose general syntax is:

```
@font-face {  
  
    descriptor : value;  
  
    ...  
  
    descriptor : value  
  
}
```

Each `@font-face` at-rule defines a new font to the browser. Subsequent requests for fonts can be satisfied by these new fonts. The browser uses the various descriptor values to ensure that the font supplied matches the font requested.

#### **8.4.4.2 Basic font descriptors**

The basic font descriptors that you use in the `@font-face` at-rule correspond to the CSS2 font properties and accept the same values as those properties. Thus, you can use the `font-family`, `font-style`, `font-variant`, `font-weight`, `font-stretch`, and `font-size` descriptors and their associated values to define a new font to the browser. For example:

```
@font-face {  
  
    font-family : "Kumquat Sans";  
  
    font-style  : normal, italic;  
  
    src : url("http://www.kumquat.com/foundry/kumquat-s.  
  
}
```

defines a font named Kumquat Sans that is available for download from [kumquat.com](http://www.kumquat.com). Within that downloadable font, both the normal and italic versions of Kumquat Sans are available. Since no other font descriptors are provided, the browser assumes that all other font properties (weight, variant, etc.) can be satisfied within this font.

In general, omitting a font descriptor lets the browser match any value provided for that descriptor. By providing one or more values for a font descriptor, you are restricting the browser to matching only those values in later font requests. Hence, you should be as specific as possible when defining a font this way, to better ensure that the browser makes good matches later. For example, if a font does not contain an italic version and you fail to tell the browser, it may use an incorrect font when attempting to fulfill a request for an italic style of that font.

#### 8.4.4.3 The `src` descriptor

The `src` descriptor in the `@font-face` at-rule tells the browser where to retrieve the font. For downloadable fonts, the value of this descriptor is its

document URL, expressed in CSS2 syntax with the `url` keyword. You can also reference locally installed fonts — ones stored on the user's machine — with `src`, but use the keyword `local` instead of `url` and supply the local name of the font instead.

The `src` descriptor's value may also be a list of locations, separated by commas. In our previous example, we might have used:

```
src : url("http://www.kumquat.com/foundry/kumquat-sans
```

to tell the browser to try to download and use Kumquat Sans from *kumquat.com* and, if that fails, to look for a locally installed copy of Lucida Sans.

You can even provide hints to the browser. CSS2 is decidedly nonpartisan when it comes to the format of the font file. Recognizing that a number of different font formats exist, the standard lets you use any format you want, presuming that the browser can make sense of it. To provide a format hint, use the keyword `format` followed one or more format names, such as:

```
src : url("http://www.kumquat.com/foundry/kumquat-sans
```

```
    local("Lucida Sans") format("truetype", "intellityp
```

In this case, the external font is in Type 1 format, while the local flavors of Lucida Sans are available in both TrueType and Intellifont formats. Other recognized font formats include `truedoc-pfr`, `opentype`, `embedded-opentype`, `truetype`, `truetype-gx`, and `speedo`.

#### 8.4.4.4 Advanced font descriptors

In addition to the standard font descriptors, CSS2 supports a number of more esoteric descriptors that further refine the defined font. Typical page designers do not have much need for these descriptors, but more discriminating typographers may find them useful.

The `unicode-range` descriptor accepts a comma-separated list of Unicode values, each beginning with `U+` followed by a hexadecimal value. Ranges of values can be specified by adding a dash and another hexadecimal value; the question mark matches any value in that position.

The purpose of the `unicode-range` descriptor is to define exactly which character glyphs are defined in the font. If characters used in your document are not available, the browser does not download and use the font. For example, a value of `U+2A70` indicates that the font contains the glyph at that position in the font. Using `U+2A7?` represents characters in the range 2A70 to 2A7F, while `U+2A70-2A9F` defines a broader range. For the most part, this descriptor is used to restrict the use of special symbol fonts to just those symbols defined in the font.

The `units-per-em` descriptor accepts a single numeric value defining the size of the font's em area. This value is important if you specify the values of other descriptors using em units.

The `panose-1` descriptor accepts exactly 10 integer values, separated by spaces, corresponding to the Panose-1 characterization of this font. Defining the actual Panose-1 values is well beyond the scope of this book; interested authors should refer to appropriate documentation for the Panose-1 system for more information.

The `stemv` and `stemh` descriptors define the thickness, in ems, of the vertical and horizontal strokes of the font. Similarly, the `cap-height` and `x-height` descriptors define the height of the upper- and lowercase glyphs in the font. Finally, the `ascent` and `descent` descriptors define the font's maximum height and depth. If you use any of these descriptors, you must also specify the `units-per-em` descriptor.

The `slope` descriptor defines the slope of the vertical stroke of the font. This is important for matching italic and oblique versions of a font.

The `baseline`, `centerline`, `mathline`, and `topline` descriptors define the conventional baseline, center baseline, mathematical baseline, and top baseline of the font. All accept a numeric value expressed in

ems. All require that you specify the `units-per-em` descriptor, too.

The `bbox` descriptor accepts exactly two coordinate (x, y) pairs, specifying the lower-left and upper-right corners of the font's bounding box. The `bbox` descriptor is important if the browser chooses to synthesize a font based on this font. By specifying the size of the bounding box, you ensure that the synthesized font occupies the same space as the desired one.

The `widths` descriptor accepts a comma-separated list of Unicode ranges, followed by space-separated values which define the widths of the characters in the indicated range. If you supply one value for a range, all the characters in that range have the same width. Multiple values are assigned to successive characters in a range. Like the `bbox` descriptor, the `widths` descriptor is used to ensure good fidelity between a synthesized font and its requested counterpart.

Finally, the optional `definitions-src` descriptor provides the URL of a file that contains all of the descriptors for a font. This is handy if you need to define a font in great detail. Rather than including the lengthy descriptors in each document or style sheet that uses the font, you define the descriptors once in a separate file and reference that file using the `definitions-src` descriptor.

## 8.4.5 Color and Background Properties

Every element in your document has a foreground and a background color. In some cases, the background is not one color, but a colorful image. The `color` and `background` style properties control these colors and images.

The children of an HTML/XHTML element normally inherit the foreground color of their parent. For instance, if you make `<body>` text red, the styles-conscious browser also displays header and paragraph text in red.

Background properties behave differently, however they are not



inherited. Instead, each element has a default background that is transparent, allowing the parent's background to show through. Thus, setting the background image of the `<body>` tag does not cause that image to be reloaded for every element within the body tag. Instead, the browser loads the image once and displays it behind the rest of the document, serving as the background for all elements that do not themselves have an explicit background color or image.

#### 8.4.5.1 The background-attachment property

If you specify a background image for an element, use the `background-attachment` property to control how that image is attached to the browser's display window. With the default value `scroll`, the browser moves the background image with the element as the user scrolls through the document. A value of `fixed` prevents the image from moving.

Both Netscape 6 and Internet Explorer support this style property.

#### 8.4.5.2 The background-color property

The `background-color` property controls the (you guessed it!) background color of an element. Set it to a color value or to the keyword `transparent` (the default value). The effects should be obvious.

While you may have become accustomed to setting the background color of an entire document through the special attributes for the `<body>` tag, the `background-color` style property can be applied to any element. For example, to set the background color of one item in a bulleted list, you could use:

```
<li style="background-color: blue">
```

Similarly, all the table header cells in a document could be given a reverse video effect with:



```
<li class="marble">Mashed Potatoes and Gravy</li>

<li class="marble">Green Beans</li>

<li class="marble">Choice of Milk, Tea, or Coffee</li>

</ul>

<h2>And for dessert:</h2>

<ul>

  <li>Creamed Quats in Milk (YUM! YUM!)</li>

</ul>
```

produces a result like that in [Figure 8-4](#).

**Figure 8-4. Placing a background image behind an element**



If the image is larger than the containing element, it is clipped to the area occupied by the element. If the image is smaller, it is repeated to tile the area occupied by the element, as dictated by the value of the `background-repeat` attribute.

You control the position of the image within the element with the

`background-position` property. The scrolling behavior of the image is managed by the `background-attachment` property.

While it may seem that a background color and a background image are mutually exclusive, you should usually define a background color even if you are using a background image. That way, if the image is unavailable for example, when the user doesn't automatically download images the browser displays the background color instead. In addition, if the background image has transparent areas, the background color is used to fill in those areas.

#### 8.4.5.4 The `background-position` property

By default, the styles-conscious browser renders a background image starting in the upper-left corner of the allotted display area and tiled (if necessary) down and over to the lower-right corner of that same area. With the `background-position` property, you can offset the starting position of the background image down and to the right of that default point by an absolute (length) or relative (percentage or keyword) offset. The resulting image fills the area from that offset starting point, and tiling (if specified) occurs left to right and top to bottom from this point to fill the display space.

You may specify one or two values for the `background-position` property. If you use a single value, it applies to both the vertical and horizontal positions. With two values, the first is the horizontal offset and the second is the vertical offset.

Length values (with their appropriate units; see [Section 8.4.1.2](#)) indicate an absolute distance from the upper-left corner of the element behind which you display the background image.

For example:

```
table {background-image: url(backgrounds/marble.gif);  
        background-position: 10px 20px}
```

offsets the marble background 10 pixels to the right and 20 pixels down from the upper-left corner of any `<table>` element in your document.

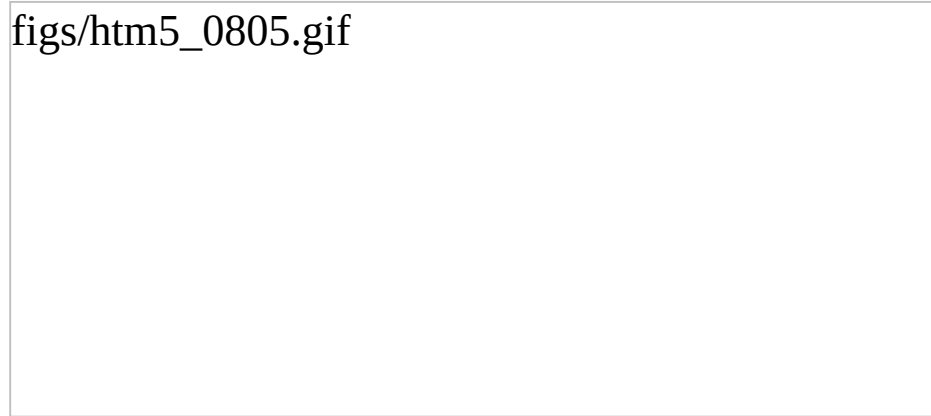
Percentage values are a bit trickier but somewhat easier to use. Measured from 0-100% from left to right and top to bottom, the center of the element's content display space is at 50%, 50%. Similarly, the position one-third of the way across the area and two-thirds of the way down is at 33%, 66%. So, to offset the background for our example dinner menu to the center of the element's content display space, we use:

```
background-position: 50% 50%
```

Notice that the browser places the first *marble.gif* tile at the center of the content display area and tiles to the right and down the window, as shown in [Figure 8-5](#).<sup>[7]</sup>

[7] Interestingly, this property worked as advertised with Internet Explorer Versions 4 and 5 but is broken in Version 6, as it is with Netscape 6: the offset works only if you set the `background-repeat` property.

### Figure 8-5. Marbled background offset to the center of the display



figs/htm5\_0805.gif

However, why use a number when a single word will do? You can use the keywords `left`, `center`, and `right`, as well as `top`, `center`, and `bottom`, for 0%, 50%, and 100%, respectively. To center an image in the tag's content area, use:

```
background-position: center center
```

You can mix and match length and percentage values,<sup>[8]</sup> so that:

[8] That is, if the browser supports the value units. So far, Internet Explorer and Netscape support only a meager repertoire of length units: pixels and percents.

```
background-position: 1cm 50%
```

places the image one centimeter to the right of the tag's left edge, centered vertically in the tag's area.

#### 8.4.5.5 The background-repeat property

Normally, the browser tiles a background image to fill the allotted space, repeating the image both horizontally and vertically. Use the `background-repeat` property to alter this "repeat" (default value) behavior. To have the image repeat horizontally but not vertically, use the value `repeat-x`. For only vertical repetition, use `repeat-y`. To suppress tiling altogether, use `no-repeat`.

A common use of this property is to place a watermark or logo in the background of a page without repeating the image over and over. For instance, this code places the watermark image in the background at the center of the page:

```
body {background-image: url(backgrounds/watermark.gif)

      background-position: center center;

      background-repeat: no-repeat

}
```

A popular trick is to create a vertical ribbon down the right-hand side of the page:

```
body {background-image: url(backgrounds/ribbon.gif);  
  
      background-position: top right;  
  
      background-repeat: repeat-y  
  
}
```

#### 8.4.5.6 The background property

Like the various font properties, the many background CSS2 properties can get cumbersome to write and hard to read later. So, like the `font` property, there is also a general `background` property.

The `background` property accepts values from any and all of the `background-color`, `background-image`, `background-attachment`, `background-repeat`, and `background-position` properties, in any order. If you do not specify values for some of the properties, those properties are explicitly set to their default values. Thus:

```
background: red
```

sets the `background-color` property to red and resets the other background properties to their default values. A more complex example:

```
background: url(backgrounds/marble.gif) blue repeat-y
```

sets all the background image and color properties at once, resulting in a marble image on top of a blue background (blue showing through any transparent areas). The image repeats vertically, starting from the center of the content display area, and does not scroll when the user scrolls the display. Notice that we include just a single position value (`center`), and the browser uses it for both the vertical and horizontal positions.

#### 8.4.5.7 The color property

The `color` property sets the foreground color for a tag's contents the color of the text lettering, for instance. Its value is either the name of a color, a hexadecimal RGB triple, or a decimal RGB triple, as outlined in [Section 8.4.1.5](#). The following are all valid property declarations:

```
color: mauve
```

```
color: #ff7bd5
```

```
color: rgb(255, 125, 213)
```

```
color: rgb(100%, 49%, 84%)
```

Generally, you'll use the `color` property with text, but you may also modify non-textual content of a tag. For example, the following example produces a green horizontal rule:

```
hr {color: green}
```

If you don't specify a color for an element, it inherits the color of its parent element.

## 8.4.6 Text Properties

Cascading style sheets make a distinction between font properties, which control the size, style, and appearance of text, and text properties, which control how text is aligned and presented to the user.

### 8.4.6.1 The letter-spacing property


The `letter-spacing` property puts additional space between text letters as they are displayed by the browser. Set the property with either a length value or the default keyword `normal`, indicating that the browser should use normal letter spacing. For example:

```
blockquote {letter-spacing: 2px}
```



puts an additional 2 pixels between adjacent letters within the `<blockquote>` tag. Figure 8-6 illustrates what happens when you put 5 pixels between characters.

**Figure 8-6. The letter-spacing property lets you stretch text out**



figs/htm5\_0806.gif

Internet Explorer and Netscape both support this property.

#### 8.4.6.2 The line-height property

Use the `line-height` property to define the minimum spacing between lines of a tag's text content. Normally, browsers single-space text lines the top of the next line is just a few points below the last line. By adding to that line height, you increase the amount of space between lines.

The `line-height` value can be an absolute or relative length, a percentage, a scaling factor, or the keyword `normal`. For example:

```
p {line-height: 14pt}
```

```
p {line-height: 120%}
```

```
p {line-height: 2.0}
```

The first example sets the line height to exactly 14 points between baselines of adjacent lines of text. The second computes the line height to 120% of the font size. The last example uses a scaling factor to set the line height to twice as large as the font size, creating double-spaced text. The value `normal`, the default, is usually equal to a scaling factor of 1.0 to 1.2.

Keep in mind that absolute and percentage values for `line-height` compute the line height based on the value of the `font-size` property. Children of the element inherit the computed property value. Subsequent changes to `font-size` by either the parent or child elements do not change the computed line height.

Scaling factors, on the other hand, defer the line-height computation until the browser actually displays the text. Hence, varying font sizes affect line height locally. In general, it is best to use a scaling factor for the `line-height` property so that the line height changes automatically as the font size changes.

Although it is usually considered separate from font properties, you may include this text-related `line-height` property's value as part of the shorthand notation of the `font` property. [\[Section 8.4.3.8\]](#)

### 8.4.6.3 The text-align property

Text justified with respect to the page margins is a rudimentary feature of nearly all text processors. The `text-align` property brings that capability to HTML for any block-level tag. (The W3C standards people prefer that you use CSS2 `text-align` styles rather than the explicit `align` attribute for block-level tags like `<div>` and `<p>`.) Use one of four values: `left`, `right`, `center`, or `justify`. The default value is, of course, `left`.<sup>[9]</sup> For example:

[9] For left-to-right locales. In right-to-left locales, the default is `right`.

```
div {text-align: right}
```

tells the styles-conscious browser to align all the text inside `<div>` tags against the right margin. The `justify` value tells the browser to align the text to both the left and right margins, spreading the letters and words in the middle to fit.

#### 8.4.6.4 The text-decoration property

The `text-decoration` property produces text embellishments, some of which are also available with the original physical style tags. Its value is one or more of the keywords `underline`, `overline`, `line-through`, and `blink`. The value `none` is the default, which tells the styles-conscious browser to present text normally.

The `text-decoration` property is handy for defining different link appearances. For example:

```
a:visited, a:link, a:active {text-decoration: underline}
```

puts lines above and below the links in your document.

This text property is not inherited, and non-textual elements are not affected by the `text-decoration` property.

Netscape and Internet Explorer support the `text-decoration` property but, thankfully, not its `blink` value.

#### 8.4.6.5 The text-indent property

Although less common today, it is still standard practice to indent the first line of a paragraph of text.<sup>[10]</sup> And some text blocks, such as definitions, typically "out-dent" the first line, creating what is called a *hanging indent*.

<sup>[10]</sup> But not, obviously, in this book.

The CSS2 `text-indent` property lets you apply these features to any block tag and thereby control the amount of indentation of the first line of the block. Use length and percentage values: negative values create the hanging indent, and percentage values compute the indentation as a percentage of the parent element's width. The default value is 0.

To indent all the paragraphs in your document, for example, you could

use:

```
p {text-indent: 3em}
```

The length unit `em` scales the indent as the font of the paragraph changes in size on different browsers.

Hanging indents are a bit trickier, because you have to watch out for the element borders. Negative indentation does not shift the left margin of the text; it simply shifts the first line of the element left, possibly into the margin, border, or padding of the parent element. For this reason, hanging indents work as expected only if you also shift the left margin of the element to the right by an amount equal to or greater than the size of the hanging indent. For example:


```
p.wrong {text-indent: -3em}
```

```
p.hang   {text-indent: -3em; margin-left: 3em}
```

```
p.large  {text-indent: -3em; margin-left: 6em}
```

creates three paragraph styles. The first creates a hanging indent that extends into the left margin, the second creates a conventional hanging indent, and the third creates a paragraph whose body is indented more than the hanging indent. All three styles are shown in use in [Figure 8-7](#).

**Figure 8-7. The effects of `text-indent` and `margin-left` on a paragraph**



figs/htm5\_0807.gif

Both Internet Explorer and Netscape support the `text-indent` property.

#### 8.4.6.6 The `text-shadow` property

The `text-shadow` property lets you give your text a three-dimensional appearance through the time-honored use of shadowing. Values for the property include a required offset and optional blur-radius and color. The property may include more than one set of values, separated with commas, to achieve a stack of shadows, with each subsequent set of values layered on top the previous one but always beneath the original text.

The property's required offset is comprised of two length values: the first specifies the horizontal offset, and the second specifies the vertical offset. Positive values place the shadow to the right and below the respective length distance from the text. Negative values move the shadow left and up, respectively.

The optional blur-radius is also a length value that specifies the boundaries for blurring, an effect that depends on the rendering agent. The other shadow value is color. This, of course, may be an RGB triple or color name, as for other properties, and specifies the shadow color. If you don't specify this value, `text-shadow` uses the color value of the `color` property. For example:

```
h1 {text-shadow: 10px 10px 2px yellow}
```

```
p:first-letter {text-shadow: -5px -5px purple, 10px 10px orange}
```

The first `text-shadow` example puts a 2-pixel blurred-yellow shadow behind, 10 pixels below, and 10 pixels to the right of level-1 headers in your document. The second example puts two shadows behind the first letter of each paragraph. The purple shadow sits 5 pixels above and 5 pixels to the left of that first letter. The other shadow, like in the first example (although orange in this case), goes 10 pixels to the right and 10 pixels below the first letter of each paragraph.

Unfortunately, we can't show you any of these effects, because none of the popular browsers support this property.

#### 8.4.6.7 The `text-transform` property

The `text-transform` property lets you automatically convert portions or all of your document's text into uppercase or lowercase lettering. Acceptable values are `capitalize`, `uppercase`, `lowercase`, or `none`.

`capitalize` renders the first letter of each word in the text into uppercase, even if the source document's text is in lowercase. The `uppercase` and `lowercase` values respectively render all the text in the corresponding case. `none`, of course, cancels any transformations. For example:

```
h1 {text-transform: uppercase}
```

makes all the letters in level-1 headers, presumably titles, appear in uppercase text, whereas:

```
h2 {text-transform: capitalize}
```

makes sure that each word in level-2 headers begins with a capital letter, a convention that might be appropriate for section heads, for instance.

Note that while `uppercase` and `lowercase` affect the entire text, `capitalize` affects only the first letter of each word in the text. Consequently, transforming the word "html" with `capitalize` generates "HtMI."

The `text-transform` property is supported by both Netscape and Internet Explorer.

#### 8.4.6.8 The `vertical-align` property

The `vertical-align` property controls the relative position of an

element with respect to the line containing the element. Valid values for this property include:

`baseline`

Align the baseline of the element with the baseline of the containing element.

`middle`

Align the middle of the element with the middle (usually the x-height) of the containing element.

`sub`

Subscript the element.

`super`

Superscript the element.

`text-top`

Align the top of the element with the top of the font of the parent element.

`text-bottom`

Align the bottom of the element with the bottom of the font of the parent element.

`top`

Align the top of the element with the top of the tallest element in the current line.

`bottom`

Align the bottom of the element with the bottom of the lowest element in the current line.

In addition, a percentage value indicates a position relative to the current baseline, so that a position of `50%` puts the element halfway up the line height above the baseline. A position value of `-100%` puts the element an entire line-height below the baseline of the current line.

Netscape supports all values except `middle` for text elements and all but the `sub` value when applying the `vertical-align` property to the `<img>` tag or other non-text inline elements. Internet Explorer supports only `sub` and `super` when applied to text elements and all values when applied to non-text inline elements.

#### 8.4.6.9 The word-spacing property

Use the `word-spacing` property to add space between words within a tag. You can specify a length value, or use the keyword `normal` to revert to normal word spacing. For example:

```
h3 {word-spacing: 25px}
```

places an additional 25 pixels of space between words in the `<h3>` tag.

Netscape and Internet Explorer Version 6, but not earlier versions, support the `word-spacing` property.

### 8.4.7 Box Properties

The CSS2 model assumes that HTML and XHTML elements always fit within rectangular boxes. Using the properties defined in this section, you can control the size, appearance, and position of the boxes containing the elements in your documents.

#### 8.4.7.1 The CSS2 formatting model


Each element in a document fits into a rectangular space or box. The CSS2 authors call this box the "core content area" and surround it with



three more boxes: the padding, the border, and the margin. [Figure 8-8](#) shows these boxes and defines some useful terminology.

**Figure 8-8. The CSS2 formatting model and terminology**

figs/htm5\_0808.gif



The top, bottom, left-outer, and right-outer edges bound the content area of an element and all of its padding, border, and margin spaces. The inner-top, inner-bottom, left-inner, and right-inner edges define the sides of the core content area. The extra space around the element is the area between the inner and outer edges, including the padding, border, and margin. A browser may omit any and all of these extra spaces for any element, and for many, the inner and outer edges are the same.

When elements are vertically adjacent, the bottom margin of the upper elements and the top margin of the lower elements overlap, so that the total space between the elements is the greater of the adjacent margins. For example, if one paragraph has a bottom margin of 1 inch, and the next paragraph has a top margin of 0.5 inches, the greater of the two margins, 1 inch, is placed between the two paragraphs. This practice is known as *margin collapsing* and generally results in better document appearance.

Horizontally adjacent elements do not have overlapping margins. Instead, the CSS2 model adds together adjacent horizontal margins. For

example, if a paragraph has a left margin of 1 inch and is adjacent to an element with a right margin of 0.5 inches, the total space between the two is 1.5 inches. This rule also applies to nested elements, so that a paragraph within a division has a left margin equal to the sum of the division's left margin and the paragraph's left margin.

As shown in [Figure 8-8](#), the total width of an element is equal to the sum of seven items: the left and right margins, the left and right borders, the left and right padding, and the element's content itself. The sum of these seven items must equal the width of the containing element. Of these seven items, only three (the element's width and its left and right margins) can be given the value `auto`, indicating that the browser can compute a value for that property. When this becomes necessary, the browser follows these rules:

- If none of these properties is set to `auto` and the total width is less than the width of the parent element, the `margin-right` property is set to `auto` and made large enough to make the total width equal to the width of the parent element.
- If exactly one property is set to `auto`, that property is made large enough to make the total width equal to the width of the parent element.
- If `width`, `margin-left`, and `margin-right` are set to `auto`, the CSS2-compliant browser sets both `margin-left` and `margin-right` to 0 and sets `width` large enough to make the total equal to the width of the parent element.
- If both the left and right margins are set to `auto`, they are always set to equal values, centering the element within its parent.

There are special rules for floating elements. A floating element (such as an image with `align=left` specified) does not have its margins collapsed with the margins of containing or preceding elements, unless the floating element has negative margins. [Figure 8-9](#) shows how this bit of HTML might be rendered:

```
<body>

<p>



Some sample text...

</body>
```

**Figure 8-9. Handling the margins of floating elements**



The browser moves the image, including its margins, as far as possible to the left and toward the top of the paragraph without overlapping the left and top margins of the paragraph or the document body. The left margins of the paragraph and the containing body are added, while their top margins are collapsed.

#### **8.4.7.2 The border properties**

The border surrounding an element has a color, a thickness, and a style. You can use various properties to control these three aspects of the border on each of the four sides of an element. Shorthand properties make it easy to define the same color, thickness, and style for the entire border, if desired. Border properties are not inherited; you must explicitly set them for each element that has a border.

### 8.4.7.3 The border-color property

Use the `border-color` property to set the border color. If not specified, the browser draws the border using the value of the element's `color` property.

The `border-color` property accepts from one to four color values. The number of values determines how they are applied to the borders (summarized in [Table 8-1](#)). If you include just one property value, all four sides of the border are set to the specified color. Two values set the top and bottom borders to the first value and the left and right borders to the second value. With three values, the first is the top border, the second sets the right and left borders, and the third color value is for the bottom border. Four values specify colors for the top, right, bottom, and left borders, in that order.

**Table 8-1. Order of effects for multiple border, margin, and padding property values**

Number of values	Affected border(s), margin(s), or padding
1	All items have the same value.
2	First value sets <i>top</i> and <i>bottom</i> ; second value sets <i>left</i> and <i>right</i> .
3	First value sets <i>top</i> ; second sets both <i>left</i> and <i>right</i> ; third value sets <i>bottom</i> .
3	First value sets <i>top</i> ; second sets both <i>left</i> and <i>right</i> ; third value sets <i>bottom</i> .
4	First value sets <i>top</i> ; second sets <i>right</i> ; third sets <i>bottom</i> ; fourth value sets <i>left</i> .

4	First value sets <i>top</i> ; second sets <i>right</i> ; third sets <i>bottom</i> ; fourth value sets <i>left</i> .
4	First value sets <i>top</i> ; second sets <i>right</i> ; third sets <i>bottom</i> ; fourth value sets <i>left</i> .

#### 8.4.7.4 The border-width property

The `border-width` property lets you change the width of the border. Like the `border-color` property, it accepts from one to four values that are applied to the various borders in a similar manner (see [Table 8-1](#)).

Besides a specific length value, you may also specify the width of a border as one of the keywords `thin`, `medium`, or `thick`. The default value, if the width is not explicitly set, is `medium`. Some typical border widths are:

```
border: 1px
```

```
border: thin thick medium
```

```
border: thick 2mm
```

The first example sets all four borders to exactly 1 pixel. The second makes the top border `thin`, the right and left borders `thick`, and the bottom border `medium`. The last example makes the top and bottom borders `thick` and the right and left borders 2 millimeters wide.

If you are uncomfortable defining all four borders with one property, you can use the individual `border-top-width`, `border-bottom-width`, `border-left-width`, and `border-right-width` properties to define the thickness of each border. Each property accepts just one value; the default is `medium`.

Netscape and Internet Explorer support this property.

### 8.4.7.5 The border-style property


According to the CSS2 model, there are a number of embellishments that you may apply to your HTML element borders.

The `border-style` property values include `none` (default), `dotted`, `dashed`, `solid`, `double`, `groove`, `ridge`, `inset`, and `outset`. The border-style-conscious browser applies one to four values for the property to each of the borders, in the same order as for the border colors and widths, as described in [Table 8-1](#).

The browser draws `dotted`, `dashed`, `solid`, and `double` borders as flat lines on top of the tag's background. The `groove`, `ridge`, `inset`, and `outset` values create three-dimensional borders: the `groove` is an incised line, the `ridge` is an embossed line, the `inset` border makes the entire tag area appear set into the document, and the `outset` border makes the entire tag area appear raised above the document. The effect of the three-dimensional nature of these last four styles on the tag's background image is undefined and left up to the browser. Netscape supports three-dimensional effects.

Both Internet Explorer and Netscape Version 6 support the border styles, as shown in [Figure 8-10](#).

**Figure 8-10. The border-style property nicely frames images**



figs/htm5\_0810.gif

### 8.4.7.6 Borders in shorthand

Specifying a complex border can get tedious, so the CSS2 standard provides five shorthand properties that accept any or all of the width, color, and style values for one or all of the border edges. The `border-top`, `border-bottom`, `border-left`, and `border-right` properties affect their respective borders' sides; the comprehensive `border` property controls all four sides of the border simultaneously. For example:

```
border-top: thick solid blue
```

```
border-left: 1ex inset
```

```
border-bottom: blue dashed
```

```
border: red double 2px
```

The first property makes the top border a thick, solid, blue line. The second sets the left border to use an inset effect that is as thick as the x-height of the element's font, while leaving the color the same as the element's color. The third property creates a blue dashed line at the bottom of the element, using the default medium thickness. Finally, the last property makes all four borders a red double line, 2 pixels thick.

That last property raises two issues. First, you cannot supply multiple values to the `border` property to selectively affect certain borders, as you can with the individual `border-color`, `border-width`, and `border-style` properties. The `border` property always affects all four borders around an element.

Secondly, a bit of reflection should reveal that it is not possible to create a double-line border just 2 pixels thick. In cases like this, the browser is free to adjust the thickness to render the border properly.

While we usually think of borders surrounding block elements like images, tables, and text flows, borders can also be applied to inline tags. This lets you put a box around a word or phrase within a text flow. The

implementation of borders on inline tags that span multiple lines is undefined and left to the browser.

Both Netscape and Internet Explorer support the `border` property, but only Internet Explorer supports the individual side properties.

#### 8.4.7.7 The `clear` property

Like its cousin attribute for the `<br>` tag, the `clear` property tells the browser whether to place a tag's contents adjacent to a "floating" element or on the first line below it. Text flows around floating elements like images and tables with an `align=left` or `align=right` attribute or any HTML/XHTML element with its `float` property set to anything but `none`. [\[Section 4.6.1\]](#) [\[Section 8.4.7.9\]](#)

The value of the `clear` property can be `none`, `left`, `right`, or `both`. A value of `none`, the default, means that the browser acts normally and places the tag's contents adjacent to floating elements on either side, if there is room to do so. The value `left` prevents contents from being placed adjacent to a floating element on its left; `right` prevents placement on the right side of a floating element; and `both` prevents the tag's contents from appearing adjacent to any floating element.

The effect of this style is the same as preceding the tag with a `<br>` tag with its `clear` attribute set. Hence:

```
h1 {clear: left}
```

has the same effect as preceding every `<h1>` tag with `<br clear=left>`.

#### 8.4.7.8 The `clip` property

Normally, the content of an element is completely visible within the display space of the element. The `clip` property defines a viewing



window within an element's display space, letting you hide unwanted elements and focus attention on some area or aspect of the content.

The default value of the `clip` property is `auto`, meaning that the viewing window matches the box of the element. Instead, you may specify a shape that creates a distinct viewing window into the element's display area. Currently, the only shape supported by CSS2<sup>[11]</sup> is a rectangle, denoted by the `rect` keyword. For example:

[11] Presumably, future versions of the standard will expand to include other shapes.

```
p {overflow : hidden;

    clip : rect(15px, -10px, 5px, 10px) }
```

The four values define the top, right, bottom, and left edges of the clipping rectangle. Each value is an offset relative to the box edges defined for the element. So, in this example, the top of the clipping area is 15 pixels below the top of the element's box, the right edge is 10 pixels to the right of the box, the bottom is 5 pixels above the bottom of the box, and the left edge is 10 pixels to the right of the left side of the box.

Note that the `clip` property takes effect only when the `overflow` property of an element is set to some value other than `visible`. When `overflow` is set to `visible`, no clipping occurs and the `clip` property is ignored.

The popular browsers don't yet support the `clip` property.

#### 8.4.7.9 The float property

The `float` property designates a tag's display space as a floating element and causes text to flow around it in a specified manner. It is generally analogous to the `align` attribute for images and tables, but it can be applied to any element, including text. [Section 5.2.6.4] [Section 10.2.1.1]

The `float` property accepts one of three values: `left`, `right`, or `none` (the default). Using `none` disables the `float` property. The others work like their `align` attribute-value counterparts, telling the browser to place the content to either side of the flow and allow other content to be rendered next to it.

Accordingly, the browser places a tag's contents (including its margins, padding, and borders) specified with `float: left` against the left margin of the current text flow, and subsequent content flows to its right, down and below the tag's contents. The `float: right` pair puts the tag contents against the right edge of the flow and flows other content on its left, down and below the tag's contents.


Although most commonly used with tables and images, it is perfectly acceptable to apply the `float` property to a text element. For example, the following creates a "run-in" header, with the text flowing around the header text, as shown in [Figure 8-11](#):

```
h2 {float: left;

text-align: center;

margin-right: 10px }
```

**Figure 8-11. Use the float property with text blocks to create run-in headers**



figs/htm5\_0811.gif

This property is supported by both Netscape and Internet Explorer.

#### **8.4.7.10 The height property**

As you might suspect, the `height` property controls the height of the associated tag's display region. You'll find it most often used with images and tables, but it can be used to control the height of other document elements as well.

The value of the `height` property is either a length value or the keyword `auto` (the default). Using `auto` implies that the affected tag has an initial height that should be used when displaying the tag. Otherwise, the height of the tag is set to the desired height. If an absolute value is used, the height is set to that length value. For example:

```
img {height: 100px}
```

tells the browser to display the image referenced by the `<img>` tag scaled so that it is 100 pixels tall. If you use a relative value, the base size to which it is relative is browser- and tag-dependent.

When scaling elements to a specific height, the aspect ratio of the object can be preserved by also setting the `width` property of the tag to `auto`. Thus:

```
img {height: 100px; width: auto}
```

ensures that the images are always 100 pixels tall, with an appropriately scaled width. [[Section 8.4.7.16](#)]

If you want to constrain the height of an element to a range rather than a specific value, use the `min-height` and `max-height` properties. These properties accept values like the `height` property and establish a range for the height of the element. The browser then adjusts the height of the element to fall within the desired range.

The `height` property is fully supported by Internet Explorer and Netscape. The `min-height` and `max-height` properties are not yet supported by any browser.

#### 8.4.7.11 The margin properties

Like the border properties, the various margin properties let you control the margin space around an element, just outside of its border (see [Figure 8-8](#)). Margins are always transparent, allowing the background color or image of the containing element to show through. As a result, you can specify only the size of a margin; it has no color or rendered style.

The `margin-left`, `margin-right`, `margin-top`, and `margin-bottom` properties all accept a length or percentage value indicating the amount of space to reserve around the element. In addition, the keyword `auto` tells the styles-conscious browser to revert to the margins it normally would place around an element. Percentage values are computed as a percentage of the containing element's width. The default margin, if not specified, is 0.

These are all valid margin settings:

```
body {margin-left: 1in; margin-top: 0.5in; margin-right: 1in; margin-bottom: 0.5in}

p {margin-left: -0.5cm}

img {margin-left: 10%}
```

The first example creates 1-inch margins down the right and left edges of the entire document and a 0.5-inch margin across the top of the document. The second example shifts the left edge of the `<p>` tag 0.5 centimeters left, into the left margin. The last example creates a margin to the left of the `<img>` tag equal to 10% of the parent element's width.

Like the shorthand `border` property, you can use the shorthand `margin` property to define all four margins, using from one to four values, which affect the margins in the order described in [Table 8-1](#). Using this notation, our `<body>` margins in the previous example could also have been specified as:

```
body {margin: 0.5in 1in}
```

The `margin-left` and `margin-right` properties interact with the

`width` property to determine the total width of an element, as described in [Section 8.4.7.1](#).

Netscape and Internet Explorer support the margin properties and values.

#### 8.4.7.12 The padding properties

Like the margin properties, the various padding properties let you control the padding space around an element, between the element's content area and its border (see [Figure 8-8](#)).

Padding always is rendered using the background color or image of the element. As a result, you can specify only the size of the padding; it has no color or rendered style.

The `padding-left`, `padding-right`, `padding-top`, and `padding-bottom` properties all accept a length or percentage value indicating the amount of space the styles-conscious browser should reserve around the element. Percentage values are computed as a percentage of the containing element's width. Padding can never be negative. The default padding is 0.

These are valid padding settings:

```
p {padding-left: 0.5cm}
```

```
img {padding-left: 10%}
```

The first example creates 0.5 centimeters of padding between the contents of the `<p>` tag and its left border. The second example creates padding to the left of the `<img>` tag equal to 10% of the parent element's width.

Like the shorthand `margin` and `border` properties, you can use the shorthand `padding` property to define all four padding amounts, using one to four values to affect the padding sides as described in [Table 8-1](#).

The `padding` property is not supported by Internet Explorer but is supported by Netscape.

#### 8.4.7.13 The overflow property

The `overflow` property tells the browser how to handle content that overflows the display area of an element. The default value of this property, `visible`, tells the browser to render all content, making it visible even if it falls outside of the element's display area.

Erring on the side of caution, you most often want the browser to display all of your document's contents. But in rare cases, elements may overlap, creating an ugly display. To prevent such mishaps, set the `overflow` property to either `hidden`, `scroll`, or `auto`.

The `hidden` value forces the browser to hide all content that overflows its allotted space, making it invisible to the user. The value `scroll` creates scrollbars for the element, which viewers may use to see the hidden content. However, scrollbars are added to the element even if the content does not overflow.

Adding permanent scrollbars ensures that the scrollbars do not come and go as the content of the element changes in size in a dynamic document. The down side to this is the clutter and distractions that scrollbars create. Avoid all this with the `auto` value for the `overflow` property. When on `auto`, scrollbars appear only when they are needed. If the element content changes so that it is not clipped, the scrollbars are removed from the element.

Neither Netscape nor Internet Explorer supports the `overflow` property.

#### 8.4.7.14 The position properties

Without intervention, the browser flows document elements together, positioned sequentially through the display. This standard behavior can

be changed with the CSS2 `position` property, in conjunction with the `top`, `bottom`, `left`, and `right` properties.

If the `position` property is set to `static`, conventional HTML/XHTML layout and positioning rules apply, with the left and top edges of the element's box determined by the browser. To shift an element with respect to its containing flow, set the `position` property to `relative`. In this case, the `top`, `bottom`, `left`, and `right` properties are used to compute the box position relative to its normal position in the flow. Subsequent elements are not affected by this position change and are placed in the flow as if this element had not been shifted.

Setting the `position` property to `absolute` removes the element from the containing flow, allowing subsequent elements to move up accordingly. The position of the element is then computed relative to the containing block, using the `top`, `bottom`, `left`, and `right` properties. This type of positioning allows an element to be placed in a fixed position with respect to its containing element but to move as that containing element moves.

Finally, setting the `position` property to `fixed` positions an element with respect to the window or page in which it is displayed. Like `absolute` positioning, the element is removed from the containing flow, with other elements shifting accordingly. The `top`, `bottom`, `left`, and `right` properties are used to set the element's position with respect to the containing window or page. Note that for continuous media (like a scrolling browser display), the element is displayed once at the desired position. For printed media, the element is printed on each page at the desired position. You might use `fixed` positioning to place headers and footers at the top and bottom of the browser window or at the top and bottom of each printed page.

The `top`, `bottom`, `left`, and `right` properties each accept a length or percentage value. When the `position` attribute is set to `relative`, the percentage is based on the size of the element's box. When `position` is set to `absolute` or `fixed`, the percentage is based on the size of the

containing element's box. When length values are used, they specify offsets from the corresponding edge of the element's containing box. For example, to position an element such that its bottom is 1 centimeter above the bottom of the browser window (or each printed page), you would set the `position` property to `fixed` and the `bottom` property to `1cm`.

#### 8.4.7.15 The visibility property

The `visibility` property determines whether the contents of an element are visible in the display. The space set aside for the element is still created and affects the layout of the document, but the content of the element may be made invisible within that space.

The default value for this property, `visible`, causes the element's content to be displayed. Setting this property to `hidden` makes the content invisible without removing the element's display box, altering the layout of the document. Note that removing an element's content and display box from the document is accomplished by setting the `display` property to `none`.

This property is often used in dynamic documents, where changing its value for an element removes its content from the display with reformatting the document.

When this property is used in conjunction with table rows, row groups, columns, and column groups, you may also specify the value `collapse`. Used in this context, the `collapse` value removes the associated row(s) or column(s) from the table without otherwise reformatting or redrawing the table. Within dynamic documents, this lets you remove elements from a table without reformatting the entire table. Used outside of a table, the `collapse` value has the same effect as the `hidden` value.

#### 8.4.7.16 The width property



The `width` property is the companion to the `height` property and controls the width of an associated tag. Specifically, it defines the width of the element's content area, as shown in [Figure 8-8](#). You'll see it most often used with images and tables, but you could conceivably use it to control the width of other elements as well.

The value for the `width` property is either a length or percentage value or the keyword `auto`. The value `auto` is the default and implies that the affected tag has an initial width that should be used when displaying the tag. If a length value is used, the width is set to that value; percentage values compute the width to be a percentage of the width of the containing element. For example:

```
img {width: 100px}
```

displays the image referenced by the `<img>` tag scaled to 100 pixels wide.

When scaling elements to a specific width, the aspect ratio of the object is preserved if the `height` property of the tag is set to `auto`. Thus:

```
img {width: 100px; height: auto}
```

makes all the images 100 pixels wide and scales their heights appropriately. [[Section 8.4.7.10](#)]

If you want to constrain the width of an element to a range rather than a specific value, use the `min-width` and `max-width` properties. These properties accept values like the `width` property and establish a range for the width of the element. The browser then adjusts the width of the element to fall within the desired range.

The `width` property interacts with the `margin-left` and `margin-right` properties to determine the total width of an element, as described in [Section 8.4.7.1](#).

### 8.4.7.17 The z-index property

In addition to the x and y position of an element within the browser window or on the printed page, each element has a vertical, or z, position. Elements with higher z positions are "closer" to the viewer and obscure elements underneath them.

Z positions are not absolute throughout a document. Instead, z positions are relative to the containing element. For example, two `<div>` elements within a document might be positioned to lie on top of one another. The first `<div>` might have a z position of 1, while the second has a z position of 2. The entire contents of the second `<div>` are displayed over (or in front of ) the first `<div>`. If elements within the first `<div>` have z positions of 3 or 4, they are still displayed within their containing `<div>`s and do not "jump out" in front of the second `<div>`.

You control the z position of an element with the `z-index` property. The value of the `z-index` property is a positive integer that sets the z position of the element with respect to its containing element. With the `z-index` property, you can dynamically alter the z position of an element to make it visible, or position a text element in front of an image to label items of interest.

No popular browsers yet support `z-index`.

## 8.4.8 List Properties

The CSS2 standard also lets you control the appearance of list elements specifically, ordered and unordered lists. Browsers format list items just like any other block item, except that the block has some sort of marker preceding the contents. For unordered lists, the marker is a bullet of some sort; for numbered lists, the marker is a numeric or alphabetic character or symbol. The CSS2 list properties let you control the appearance and position of the marker associated with a list item.

### 8.4.8.1 The `list-style-image` property

The `list-style-image` property defines the image that the browser uses to mark a list item. The value of this property is the URL of an image file or the keyword `none`. The default value is `none`.

The image is the preferred list marker. If it is available, the browser displays it in place of any other defined marker. If the image is unavailable, or if the user has disabled image loading, the browser uses the marker defined by the `list-style-type` property (see [Section 8.4.8.3](#)).

HTML/XHTML authors use the `list-style-image` property to define custom bullets for their unordered lists. While any image could conceivably be used as a bullet, we recommend that you keep your marker GIF or JPEG images small, to ensure attractively rendered lists.

For example, by placing the desired bullet image in the file *mybullet.gif* on your server, you could use that image:

```
li {list-style-image: url(pics/mybullet.gif); list-sty
```

In this case, the browser uses the image if it is able to successfully download *mybullet.gif*. Otherwise, the browser uses a conventional square bullet.

The `list-style-image` property is supported by Internet Explorer and Netscape. However, they differ in where they position the list marker. Netscape and Internet Explorer 6 put it outside, and Internet Explorer 5 puts it inside the item. Read the next section for an explanation.

### 8.4.8.2 The `list-style-position` property

There are two ways to position the marker associated with a list item: inside the block associated with the item or outside the block. Accordingly, the `list-style-position` property accepts one of two values: `inside` or `outside`.

The default value is `outside`, meaning that the item marker hangs to the

left of the item, like this:

- This is a bulleted list with an "outside" marker

The value `inside` causes the marker to be drawn with the list item flowing around it, much like a floating image:

- This is a bulleted list with an "inside" marker

Notice that the second line of text is not indented but instead lines up with the left edge of the marker.

The current versions of the popular browsers fully support the `list-style-position` property.

#### 8.4.8.3 The `list-style-type` property

The `list-style-type` property serves double duty in a sense, determining how a styles-conscious browser renders both ordered and unordered list items. The property has the same effect as the `type` attribute on a list item. [[Section 7.3.1.1](#)]

When applied to items within an unordered list, the `list-style-type` property uses one of four values `disc`, `circle`, `square`, or `none` and marks the unordered list items with a corresponding dingbat. The default value of a level-1 list item is `disc`, although browsers change that default depending on the nesting level of the list.

When applied to items within an ordered list, the `list-style-type` property uses one of six values `decimal`, `lower-roman`, `upper-roman`, `lower-alpha`, `upper-alpha`, or `none` corresponding to the item numbers expressed as decimal values, lowercase Roman numerals, uppercase Roman numerals, lowercase letters, or uppercase letters, respectively. Most browsers use decimal numbering as the default.

The popular browsers support `list-style-type` as well as the `list-`

`style` property described in the next section.

#### 8.4.8.4 The `list-style` property

The `list-style` property is the shorthand version for all the other `list-style` properties. It accepts any or all of the values allowed for the `list-style-type`, `list-style-position`, and `list-style-image` properties, in any order and with values appropriate for the type of list they are to affect. These are valid `list-style` properties:

```
li {list-style: disc}
```

```
li {list-style: lower-roman inside}
```

```
li {list-style: url(http://www.kumquat.com/images/tiny
```

The first example creates list items that use a disc as the bullet image. The second causes numbered list items to use lowercase Roman numerals, drawn inside the list item's block. In the last example, the styles-conscious browser uses a square as the bullet image if the referenced image is unavailable.

#### 8.4.8.5 Using list properties effectively

Although you can apply list properties to any element, they affect only the appearance of elements whose `display` property is set to `list-item`. Normally, the only tag with this property is the `<li>` tag. [Section 8.4.10.1]

However, this shouldn't deter you from using these properties elsewhere, particularly with the `<ul>` and `<ol>` tags. Because these properties are inherited by elements whose parents have them set, modifying a list property for the `<ul>` and `<ol>` tags subsequently modifies it for all the `<li>` tags contained within that list. This makes it much easier to define lists with a particular appearance.

For example, suppose you want to create a list style that uses lowercase Roman numerals. One way is to define a class of the `<li>` tag with the appropriate `list-style-type` defined:

```
li.roman {list-style-type: lower-roman}
```

Within your list, you'll need to specify each list element using that class:

```
<ol>

  <li class=roman>Item one

  <li class=roman>Item two

  <li class=roman>And so forth

</ol>
```

Having to repeat the class name is tedious and error-prone. A better solution is to define a class of the `<ol>` tag:

```
ol.roman {list-style-type: lower-roman}
```

Any `<li>` tag within the list inherits the property and uses lowercase Roman numerals:

```
<ol class=roman>

  <li>Item one

  <li>Item two

  <li>And so forth

</ol>
```

This is much easier to understand and manage. If you want to change the numbering style later, you need only change the `<ol>` tag properties,

rather than finding and changing each instance of the `<li>` tag in the list.

You can use these properties in a much more global sense, too. Setting a list property on the `<body>` tag changes the appearance of all lists in the document; setting it on a `<div>` tag changes all the lists within that division.

## 8.4.9 Table Properties

For the most part, HTML/XHTML browsers render table content using the same properties that control the rendering of conventional document content. However, there are a few special circumstances that occur only within tables. To give authors greater control over these items, CSS2 has added a few table-specific properties. None are yet supported by the popular browsers.

### 8.4.9.1 The `border-collapse`, `border-spacing`, and `empty-cells` properties

There are two divergent views regarding cell borders within tables. The first view holds that each cell is an independent entity with unique borders. The other view holds that adjacent cells share the border side and that changing a border in one cell should affect the neighboring cell.

To give the most control to authors, CSS2 provides the `border-collapse` property, which lets you choose the model that suits your style. By default, the value of this property is `collapse`, meaning adjacent cells share their border style. Alternatively, you can set the `border-collapse` property to `separate`, which enlarges the table so that borders are rendered separately and distinctly around each cell.

If you choose the `separate` model, you can also use the `border-spacing` property to set the spacing between adjacent borders. The default border spacing is 0, meaning that adjacent cell borders touch each other, although some browsers may use a different default. By

increasing this value, you cause the browser to insert additional space between borders, allowing the background color or image of the table to show through. If you specify just one value for `border-spacing`, it sets the spacing for both horizontal and vertical borders. If you provide two values, the first sets the horizontal spacing and the second determines the vertical spacing.

Within the `separate` model, you can also control how borders are drawn around empty cells. By default, borders are drawn around every cell in a table, even if it has no content. You can change this by switching the `empty-cells` property from its default value of `show` to the value `hide`. When this property is set, empty cells simply show the table background. If a whole row of cells is empty, the browser removes the row from the table entirely.

#### 8.4.9.2 The `caption-side` property

Use the `caption-side` property only with the `<caption>` element. It accepts values of `top`, `bottom`, `left`, or `right`, and tells the browser where to place the caption adjacent to its associated table. The `caption-side` property provides a more consistent method of placing the caption than the browser-dependent `align` attribute of the `<caption>` tag. None of the popular browsers support `caption-side` yet, but you might want to include it anyway, for future versions.

#### 8.4.9.3 The `speak-header` property

An audio-capable browser might offer a number of ways for users to navigate by hearing the contents of a table. A simplistic approach would have the browser read the table contents in order, from top to bottom and right to left. A more sophisticated audio-browser organizes the table contents according to their respective headers and reads the information in a more comprehensible manner. To avoid confusion in any case, the browser must provide some way to tell the user which cell it is reading.



The `speak-header` property provides two ways for a browser to identify a cell or collection of cells in the table. If `once` (the default) is specified, the browser reads the contents of a header cell only once before proceeding to read the contents of each of its associated data cells. This way, a user moving across a row of cells would hear the row header and column header of the first cell in the row, but would hear the changing column headers only as she moved to subsequent cells in the row.

If you set the `speak-header` property to `always`, the browser prefaces the reading of each cell's contents with a reading of its associated header. This may prove more useful with complex tables or where the header values make it easier to understand the table contents especially when a table contains only numbers.

Note that headers are spoken only when the browser knows which header cells are associated with a data cell. Conscientious authors always use the `header` attribute with their table cells, to specify the header cells related to each data cell in their tables.

#### 8.4.9.4 The `table-layout` property

Table layout is a tough task for any browser. To create an attractive table, the browser must find the widest cell in each column, adjust that column to accommodate the width, and then adjust the overall table to accommodate all of its columns. For large tables, document rendering can be noticeably slowed as the browser makes several passes over the table, trying to get things just right.

To help in this process, use the `table-layout` property. If you set the property to `fixed`, the browser determines column widths based on the widths of cells in the first row of the table. If you explicitly set the column widths, setting the table's `table-layout` property to `fixed` makes the table-rendering process even faster, enhancing the readers' experience as they view your document.

By default, the `table-layout` property is set to `auto`, which forces the

browser to use the more time-consuming, multiple-pass layout algorithm, even if you specify the widths of your columns in the table. If your table content is variable and you cannot explicitly set the widths, leave the `table-layout` property set to `auto`. If you can fix your column widths and your table content is amenable, set `table-layout` to `fixed`.

## 8.4.10 Classification Properties

Classification properties are the most fundamental of the CSS2 style properties. They do not directly control how a styles-conscious browser renders HTML or XHTML elements. Instead, they tell the browser how to classify and handle various tags and their contents as they are encountered.

For the most part, you should not set these properties on an element unless you are trying to achieve a specific effect.

### 8.4.10.1 The `display` property

Every element in an HTML or XHTML document can be classified, for display purposes, as a block item, an inline item, or a list item. Block elements, like headings, paragraphs, tables, and lists, are formatted as separate blocks of text, separate from their previous and following block items. Inline items, like the physical and content-based style tags and hyperlink anchors, are rendered within the current line of text within a containing block. List items, specifically `<li>`-tagged content, are rendered like block items, with a preceding bullet or number known as a *marker*.

The `display` property lets you change an element's display type to `block`, `inline`, `list-item`, or `none`. The first three values change the element's classification accordingly; the value `none` turns off the element, preventing it or its children from being displayed in the document.

Conceivably, you could wreak all sorts of havoc by switching element classifications, forcing paragraphs to be displayed as list items and

converting hyperlinks to block elements. In practice, this is just puerile monkey business, and we don't recommend that you change element classifications without a very good reason to do so.

Netscape fully supports this property; Internet Explorer supports only the `block` and `none` values.

#### 8.4.10.2 The white-space property

The `white-space` property defines how the styles-conscious browser treats whitespace (tabs, spaces, and carriage returns) within a block tag. The keyword value `normal` the default collapses whitespace so that one or more spaces, tabs, and carriage returns are treated as a single space between words. The value `pre` emulates the `<pre>` tag, in that the browser retains and displays all spaces, tabs, and carriage returns. Finally, the `nowrap` value tells the browser to ignore carriage returns and not insert automatic line breaks; all line-breaking must be done with explicit `<br>` tags.

Like the `display` property, the `white-space` property is rarely used for good purposes. Don't change how elements handle whitespace without a compelling reason for doing so.

Internet Explorer 6 supports the `nowrap` value, while Netscape 6 supports all values for the `white-space` property.

#### 8.4.11 Generated Content Properties

The idea of generated content is not new to HTML. Even the earliest browsers automatically appended appropriate bullets or numbers to enhance the readability of your unordered and ordered list items. Such features are hardly enough, though, and authors have wished for better content-generation tools in HTML. CSS2 finally comes through, giving authors the ability to create arbitrary content, numbered lists, and all sorts of element-based content.

The foundation of the CSS2 generated-content model is the `content` and `quotes` properties, along with the `:before` and `:after` pseudoelements. You use the former to define the content you need, and use the latter to position that content with respect to the elements in your document.

#### 8.4.11.1 The `:before` and `:after` pseudoelements

You were introduced to pseudoelements earlier in this chapter, and you even saw one (`:first-letter`) in action (see [Figure 8-2](#)). The `:before` and `:after` pseudoelements operate similarly. Append either to a style-element selector to select and specify the content and properties of generated content in your document. In general, any content created within these pseudoelements inherits the display attributes of the parent element, such that fonts, sizes, and colors applied to an element are also applied to its generated content. For example:

```
p.note { color : blue }  
  
p.note:before { content : "Note: " }
```

This style example inserts the word "Note:" before every `<p class=note>` element. The inserted text is rendered in blue, like the rest of the paragraph. Replacing it with this style would color the inserted text red, while the remainder of the note would be blue:

```
p.note:before {content : "Note: "; color : red}
```

Any generated content, before or after an element, is included in the box of an element and affects its formatting, flow, size, and layout.

#### 8.4.11.2 The `content` property

The `content` property accepts a wide variety of values, ranging from simple strings to automatic counter references. Any number of these

values, separated by spaces, can be included in a single `content` property. The browser concatenates the values to form a single value that it then inserts into the document.

The simplest of `content` values is a quote-enclosed string. You may not include HTML or XHTML markup in the string. Rather, use escape sequences to generate special text (e.g., `"\A"`, which generates a line break).

CSS2 escape sequences are like HTML/XHTML character entities. Whereas character entities begin with the ampersand (&), followed by the name or decimal value of a character (# suffix for the latter), you create the same characters for CSS2 string-content property values by preceding the hexadecimal equivalent of the character with a backslash (\). The escape sequence `\A` is the same as the character entity `&#010`, which, if you consult [Appendix F](#), you'll find is the line-feed character.

The `content` property also accepts URL values. Expressed in styles, not HTML-like fashion, the URL may point to any object acceptable to the browser, including text, images, or sound files. For example, to place a decorative symbol next to each equation in a document, you might use:

```
p.equation:before { content : url("http://www.kumquat...") }
```

Keep in mind that the object shouldn't contain HTML/XHTML markup, because the browser inserts its contents verbatim into the document.

The `content` property also supports automatic generation of contextually correct, locale-specific quotation marks. You insert them using the `open-quote` and `close-quote` keywords. These keywords insert the appropriate quotation mark and increment or decrement, respectively, the browser's nested quotation counter. You can control the appearance of the quotation marks using the `quotes` property, described below. You may also use the `no-open-quote` and `no-close-quote` keywords, which increment or decrement the nesting depth without inserting a quotation mark.

A clever feature of the `content` property is its ability to have the browser render the value of any attribute of its associated element. The `attr` value has a single parameter, corresponding to the name of an attribute. If that attribute is defined for the element, its value is inserted into the document. To display the URL of an image after the image, for instance, you might write:

```
img::after { content : "attr(src) " }
```

If the attribute is not defined for the element, no content gets inserted, although the other values for the `content` property (like the parentheses we included in the example above) would still be inserted.

One of the most powerful features of the `content` property is its ability to create numbered lists. We cover this in detail in [Section 8.4.11.4](#).

### 8.4.11.3 Specifying quotation marks

While you insert quotation marks using the `open-quote` and `close-quote` values with the `content` property, you control the actual characters used for quotation marks with the `quotes` property.

The value of this property is one or more pairs of strings. The first pair defines the open and close quotation marks for the outermost level of quotations in your document. The next pair specifies the next level, and so forth. If the quotation level exceeds the supplied pairs of characters, the browser starts over with the outermost pair. Note that while most languages use single characters as quotation marks, you can specify strings of any length to be used as quotation marks.

You may also want to specify alternative quotation marks based on the language used. You can use the `:lang` pseudoelement to associate different `quotes` properties with different languages. For example:

```
q:lang(en) { quotes : `"` `"' "`` "''" }
```

```
q:lang(no) { quotes : "«" "»" "<" ">" }
```

ensures that English and Norwegian documents use their respective quotation marks.

#### 8.4.11.4 Creating counters

You can create simple numbered lists easily in HTML and XHTML with the `<ol>` element. More complex numbered lists, especially nested numbered lists, are impossible with the markup languages, though. Instead, CSS2 provides the notion of a counter whose value can be set and changed as the browser renders your document. Insert the value of the counter using special functions recognized by the `content` property, and alter the appearance and format of the counter with other CSS2 properties.

Every CSS2 counter has a name. To create a counter, simply mention its name in the `counter-reset` or `counter-increment` properties associated with any element. If an instance of that named counter does not already exist in the current document nesting level, the CSS2-conscious browser automatically creates it. Thereafter, set or reset the value of the counter as needed. For example, suppose we want to use `<h1>` elements as chapter headings, with `<h2>` elements as section headings. Both chapters and sections are numbered, with section headings being reset with each new chapter. You can achieve this with:

```
h1:before { counter-increment : chapter; counter-reset  
  
h2:before { counter-increment : section }
```

When the CSS2-conscious browser encounters the first `<h1>` element in the document, it creates both the `chapter` and `section` counters and resets their values to 0. At the same time, and for every encounter thereafter, the CSS2-conscious browser enacts the `counter-increment` property to set the `chapter` counter to 1, representing Chapter 1, then 2, and so on. As `<h2>` elements are encountered within

a chapter, the `section` counter gets incremented according to the h2 style rule, numbering each section in order. Notice, too, that the `section` counter gets reset by the h1 rule, so that the section counter restarts for each chapter.<sup>[12]</sup>

[12] Note here that the browser doesn't display counters unless you explicitly tell it to. See [Section 8.4.11.5](#).

Both the `counter-reset` and `counter-increment` properties accept lists of counter names, letting you reset or increment groups of counters in one property. You can also supply a numeric value after a counter name, so that with `counter-reset`, the counter gets initialized to that specified value, and `counter-increment` adds the value to the current counter value. Negative numbers are allowed, too, so that you may count down, if desired.

For example, if we want our document to begin with Chapter 7 and we want section numbers to increase by 2, we might rewrite the previous example as follows:

```
body { counter-reset : chapter 6 }

h1:before { counter-increment : chapter; counter-reset

h2:before { counter-increment : section 2 }
```

Notice how we created the `chapter` counter in the earliest possible element in our document, using a value one less than the desired first value? When the browser encounters the first `<h1>` element, it creates, sets to 6, and then increments the `chapter` counter.

The scope of a counter name is the nesting level in which it is defined, not necessarily document-wide. If you use the same counter name in a child element, the browser creates a new instance of the counter at that level. In our example, all the `<h1>` and `<h2>` elements exist at the same nesting level, so one instance of the `chapter` and `section` counters serves that whole level. If you nested a `<div>` tag in that element, which



in turn contained `<h1>` and `<h2>` elements, new instances of both counters would be created at that new level.

This nesting behavior is critical for nested numbered lists to work. If you associate a counter with the `<li>` element and then nest several ordered lists, each list level has its own instance of the counter, with separate number sequences at each level.

#### 8.4.11.5 Using counters in your documents

Creating counters is of little use if you don't display their values in your documents. The display is not automatic. To show a counter, use the special `counter()` and `counters()` values in the `content` property.

The `counter()` value requires the name of a counter inside its parentheses, with an optional format specification. The browser then displays the value of the specified counter within the generated content in the format desired. The format can be any list format accepted by the `list-style-type` property, as described in [Section 8.4.8.3](#).

For example, to actually display the numbers of our numbered chapters and sections, we expand our style rules for the `<h1>` and `<h2>` elements:

```
h1:before { counter-increment : chapter;

    counter-reset : section;

    content : "Chapter " counter(chapter) ":" }

h2:before { counter-increment : section;

    content : "Section " counter(section) ":" }
```

Then, when the CSS2-conscious browser encounters this in the document:

```
<h1>The Harvest Commences!</h1>
```

it renders it as:<sup>[13]</sup>

<sup>[13]</sup> We, of course, show you how it should appear, as none of the popular browsers yet support these CSS2 properties.

Chapter 1: The Harvest Commences!

To number our chapters using Roman numerals, we would change the properties to:

```
h1:before { counter-increment : chapter;

            counter-reset : section;

            content : "Chapter " counter(chapter, upper-roman)

h2:before { counter-increment : section;

            content : "Section " counter(section, lower-roman)
```

The `counter()` value is the value of the counter at the current nesting level. To access all the values of the same-named counter at all nesting levels, use the plural `counters()` value instead. Include the counter name in the parentheses and a separator string. The browser puts the separator string between each of the list of values for the counter in the display. You may also supply a format type to switch from the default decimal numbering.

The `counters()` value is most useful when creating nested numbered lists. Consider these properties:

```
ol { counter-reset: item }

li:before { counter-increment: item ;

            content: counters(item, ".") }
```

If you nest several `<ol>` elements in your document, each `<li>` includes all the nested values, separated by periods. This creates the familiar numbering pattern<sup>[14]</sup> of 1, 1.1, 1.1.1, etc., as the nesting increases.

[14] Surely you've noticed it in this book!

#### 8.4.11.6 Creating markers

According to the CSS2 standard, the browser should place styles-generated content before or after conventional HTML/XHTML content of the affected element, and it should therefore become part of the element's flow. This is not acceptable for numbered lists, where the number should be displayed separate from the content of each numbered item. To do this, add the `display` property to your generated content, with the special value of `marker`. To make our nested numbered list example completely correct, for instance, we use the rules:

```
ol { counter-reset: item }

li:before { display : marker;

            counter-increment: item ;

            content: counters(item, ".") }
```

This way, the generated counter number gets rendered to the left of the element's actual content. In a similar fashion, you can place markers after an element. For example, use the following properties to create numbered equations within chapters (the `<blockquote>` element delineates the equation):

```
h1:before { counter-increment : chapter;

            counter-reset : equation }

blockquote:after { counter-increment : equation;
```

```
display : marker;
```

```
content : "("counter(chapter, upper-roman) "-" coun
```

When rendering a marker, the browser determines where to place the marker content in relation to the element's actual content. You modify this behavior with the `marker-offset` property. It accepts a numerical (length) value equal to the distance between the edge of the marker and the edge of the associated element. For example, to ensure that our equation numbers get shifted 0.5 inches away from the related equation, we could use:

```
h1:before { counter-increment : chapter;
```

```
counter-reset : equation }
```

```
blockquote:after { counter-increment : equation;
```

```
display : marker;
```

```
content : "("counter(chapter, upper-roman) "-" coun
```

```
marker-offset : 0.5in }
```

Currently, none of the generated-content and marker-control properties and values are supported by any browser.

## 8.4.12 Audio Properties

From its humble beginnings, HTML has been a visual medium for computer display devices. Although increasing attention has been paid to other media as the standard evolved, CSS2 is the first real effort to comprehensively address using HTML/XHTML documents for non-visual media.

For example, CSS2 forecasts that someday some browsers will be able

to speak the textual content of a document, using some sort of text-to-speech technology. Such a browser would be of enormous help for the visually impaired and would also allow web browsing via the phone and other devices where a visual display is not readily available or usable. Imagine the excitement of driving down the road while your favorite web pages are read to you!<sup>[15]</sup>

<sup>[15]</sup> Conversely, imagine the annoyance of someone having web pages read to them while you try to enjoy a quiet meal or watch a movie. We are constantly reminded that every advance in technology has a dark side.

CSS2 attempts to standardize these alternative renderings by defining a number of properties that control the aural experience of a web listener. None of them are currently supported in any popular browser, but we envision a time in the near future when you may be able to take advantage of some or all of these properties.

#### 8.4.12.1 The volume property

The most basic aural property is `volume`. It accepts numeric length or percentage values along with a few keywords corresponding to preset volume levels.

Numeric values range from 0 to 100, with 0 corresponding to the minimum audible level and 100 being the maximum comfortable level. Note that 0 is not the same as silent, as the minimum audible level in an environment with loud background noise (like a factory floor) may be quite high.

Percentage values compute an element's volume as a percentage of the containing element's volume. Computed values less than 0 are set to 0; values greater than 100 are set to 100. Thus, to make an element twice as loud as its parent element, set the `volume` property to `200%`. If the volume of the parent element is 75, the child element's volume gets set to the limit of 100.

You also may specify a keyword value for the `volume` property. Here,

`silent` actually turns the sound off. The `x-soft` value corresponds to a value of 0; `soft` is the same as the numeric volume of 25; `medium` is 50, `loud` is 75, and `x-loud` corresponds to 100.

### 8.4.12.2 Speaking properties

Three properties control if and how text is converted to speech. The first is `speak`, which turns speech on and off. By default, the value of `speak` is `normal`, meaning that text is converted to speech using standard, locale-specific rules for pronunciation, grammar, and inflection. If you set `speak` to `none`, speech is turned off. You might use this feature to suppress speaking of secondary content or content that does not readily translate to audio, such as a table.

Finally, you can set the `speak` property to `spell-out`, which spells out each word. This is useful for acronyms and abbreviations. For example, using:

```
acronym { speak : spell-out }
```

ensures that acronyms such as URL get translated aurally as "you-are-ell" and not as "earl."

By default, the `speak-punctuation` property is set to `none`, causing punctuation to be expressed as pauses and inflection in the generated speech. If you give this property the `code` value, punctuation is spoken literally. This might be useful for aurally reproducing programming code fragments or literal transcriptions of some content.<sup>[16]</sup>

[16] Regrettably, there is no `victor-borge` mode for this property. Perhaps CSS3 will address this egregious oversight.

The `speak-numeral` property defaults to the value `continuous`, meaning that numerals are pronounced as a single number. Accordingly, the number "1234" would be reproduced as "one thousand two hundred thirty-four." When set to `digits`, the numbers are pronounced digit by

digit, such as "one, two, three, four."

### 8.4.12.3 Voice characteristics

To create a richer listening experience, CSS2 defines a number of properties that alter the spoken content. This lets you use different voices for different content, speed up the speech, and change the pitch and stress levels in the speech.

The `speech-rate` property accepts a numeric length value that defines the number of words spoken per minute. The default value is locale-dependent, since different cultures have different notions of a "normal" rate of speech. Instead of a specific value, you may use any of the keywords `x-slow`, `slow`, `medium`, `fast`, and `x-fast`, corresponding to 80, 120, 180, 300, and 500 words per minute, respectively. The `faster` keyword sets the rate to 40 words per minute faster than the containing element, while `slower` sets the rate to 40 words per minute slower than the containing element.

The `voice-family` property is the aural analog of the `font-family` property. A voice family defines a style and type of speech. Such definitions are browser- and platform-specific, much like fonts. It is assumed that browsers will define generic voice families, such as "male," "female," and "child," and may also offer specific voice families like "television announcer" or "book author." The value of the `voice-family` property is a comma-separated list of these voice family names; the browser goes down the list until it finds a voice family that it can use to speak the element's text.

The `pitch` property controls the average pitch, with units in Hertz (`hz`), of the spoken content. The basic pitch of a voice is defined by the voice family. Altering the pitch lets you create a variation of the basic voice, much like changing the point size of a font. For example, with a change in pitch, the "book author" might be made to sound like a chipmunk.<sup>[17]</sup>

[17] Assuming, of course, that he or she doesn't already sound like a chipmunk.

You can set the `pitch` property to a numeric value such as `120hz` or `210hz` (the average pitches of typical male and female voices) or to one of the keywords `x-low`, `low`, `medium`, `high`, or `x-high`. Unlike other speech property keywords, these do not correspond to specific pitch frequencies but instead are dependent on the base pitch of the voice family. The only requirement is that these keywords correspond to increasingly lower or higher pitches.

While the `pitch` property sets the average pitch, the `pitch-range` property defines how far the pitch can change as the browser reproduces text aurally. The value of this property is a numeric value ranging from 0 to 100, with a default value of 50. Setting the `pitch-range` to 0 produces a flat, monotonic voice; values over 50 produce increasingly animated and excited-sounding voices.

The `stress` property controls the amount of inflection that is placed on elements in the spoken text. Various languages have differing rules for stressing syllables and adding inflection based on grammar and pronunciation rules. The `stress` property accepts a value in the range of 0 to 100, with the default value of 50 corresponding to "normal" stress. Using a value of 0 eliminates inflection in the spoken content. Values over 50 increasingly exaggerate the inflection of certain spoken elements.

The `richness` property controls the quality or fullness of the voice. A richer voice tends to fill a room and carries further than a less rich, or smoother, voice. Like `pitch` and `stress`, the `richness` property accepts a numeric value in the range of 0 to 100, with a default value of 50. Values approaching 0 make the voice softer. Values over 50 make the voice fuller and more booming.

#### 8.4.12.4 Pause properties

Like whitespace in a printed document, insert pauses in spoken content to offset and thereby draw attention to content as well as to create a better-paced, more understandable spoken presentation.



The `pause-before` and `pause-after` properties generate pauses just before or just after an element's spoken content. These properties accept either an absolute time value (using the `s` or `ms` units) or a percentage value. With a percentage value, the pause is relative to the length of time required to speak a single word. For example, if the speech rate is 120 words per minute, one word, on average, is spoken every 0.5 seconds. A pause of 100%, therefore, would be 0.5 seconds long; a 20% pause would be 0.1 seconds long, and so on.

The `pause` property sets both the `pause-before` and `pause-after` properties at once. Use one value for `pause` to set both properties; the first of two values sets `pause-before`, and the second sets the `pause-after` property value.

#### 8.4.12.5 Cue properties

Cue properties let you insert audible cues before or after an element. For example, you might precede each chapter in a book with a musical cue, or denote the end of quoted text with an audible tone.

The `cue-before` and `cue-after` properties take as their value the URL of a sound file, which the browser loads and plays before or after the styled document element, respectively. Technically, the sound can be of any duration, but the presumption is that audible cues are short and nonintrusive, enhancing the audio experience instead of overwhelming it.

Use the `cue` property to set both the `cue-before` and `cue-after` properties at once. If you provide one URL value, it sets both cue sounds; with two values, the first sets the `cue-before` sound and the second sets the `cue-after` sound.

#### 8.4.12.6 Audio mixing

To create a more pleasant listening experience, you may want to play background music during a spoken passage. The `play-during`

property meets this need. Its values are the URL of the sound file and several keywords that control playback.

The `repeat` keyword repeats the background audio until the spoken content is complete. If you don't use this keyword, the background sound plays once, even if it is shorter than the spoken content. A background sound that is longer than the spoken content ends when the content ends.

The `mix` keyword tells the CSS2-conscious browser to meld the background sound with any other background sounds that may be playing as defined by some parent element. If you don't use this keyword, child-element background sounds replace parent-element background sounds, which resume when the current element has finished.

In lieu of a URL representing the background sound, you can use the value `none`. This lets you silence all background sounds, such as one or more playing from parent elements, while the current element is being spoken.

#### 8.4.12.7 Spatial positioning

While a rendered document exists on a two-dimensional page, spoken content can be placed anywhere in the three-dimensional space surrounding the listener. The CSS2 standard defines the `azimuth` and `elevation` properties so that you can place spoken content from elements in different places around the listener. `azimuth` relates to where around and `elevation` tells how far above or below the sound appears to the listener.

The `azimuth` property accepts either an angle value or keywords indicating a position around the listener. The position directly in front of the listener is defined to be 0 degrees. The listener's right is at 90 degrees, while directly behind is 180 degrees. The listener's left is at 270 degrees or, equivalently, -90 degrees.

Position keywords include a base position, possibly modified by the `behind` keyword. These keywords correspond to the angular positions listed in [Table 8-2](#).

**Table 8-2. Angular equivalents for azimuth keywords**

Keyword	Angular position	Angular position when used with behind
<code>left-side</code>	270	270
<code>far-left</code>	300	240
<code>left</code>	320	220
<code>center-left</code>	340	200
<code>center</code>	0	180
<code>center-right</code>	20	160
<code>right</code>	40	140
<code>far-right</code>	60	120
<code>right-side</code>	90	90

The `leftwards` keyword subtracts 20 degrees from the parent element's `azimuth`. Similarly, `rightwards` adds 20 degrees to the parent element's `azimuth`. Note that this process can continue until you work your way around the listener; these values add or subtract 20 degrees no matter what the `azimuth` of the parent is.

The `elevation` property accepts an angular value ranging from -90 degrees to 90 degrees, corresponding to from directly below the listener to directly above the listener. Zero degrees is considered to be level with the listener's ears. You can also use the `below`, `level`, and `above` keywords for -90, 0, or 90 degrees, respectively.

Use the `higher` keyword to increase the elevation by 10 degrees over the parent element's `elevation`; `lower` changes the elevation of the sound to 10 degrees below the parent element's `elevation`.

### 8.4.13 Paged Media

Printing has never been HTML's strong suit. In fact, printing has been intentionally ignored by the HTML and XHTML standards, because printing assumes page layout, and HTML and XHTML are not layout tools.

Authors use cascading style sheets to format and lay out their HTML/XHTML document contents, so it is not surprising that the CSS2 standard introduces some basic pagination control features that let authors help the browser figure out how to best print their documents. These features fall into two groups: those that define a particular page layout and those that control the pagination of a document.

#### 8.4.13.1 Defining pages

As an extension to the box model, CSS2 defines a "page box," a box of finite dimensions in which content is rendered. The page box does not necessarily correspond to a physical sheet of paper; the user agent maps one or more page boxes to sheets of paper during the printing process. Many small page boxes may fit on a single sheet; large page boxes may be scaled to fit on a sheet or may be broken across several sheets at the discretion of the browser.

During the printing process, content flows into the page box, is paginated appropriately, and is transferred to a target sheet on a hard-copy output

device. The dimensions of the page box may differ from the browser's display window, so the flow and rendering of a printed document may be completely different from its onscreen representation. As always, obtaining a specific rendered appearance for your documents is generally impossible. However, you can use the CSS2 pagination features to help the browser print your document in an attractive, useful manner.

You define a page box using the special `@page` at-rule. Immediately following the `@page` keyword is an optional name for the page, followed by a list of properties separated by semicolons and enclosed in curly braces. These properties define the size, margins, and appearance of the page box.

Use the `size` property to specify the size of the page box. The value of this property is either one or two length values, or one of the special keywords `portrait`, `landscape`, or `auto`. If you provide a single length value, it creates a square, setting both the width and height of the page to that value. Two length values set the width and the height of the page, respectively. The `portrait` keyword specifies the locally accepted page size that is taller than it is wide (typically 8 1/2 by 11 inches), while `landscape` uses a locally accepted page size that is wider than it is tall (typically 11 by 8 1/2 inches). Finally, `auto` creates a page box that is the same size as the target sheet of paper on which the document is printed.

In general, you should use the special page size keywords to ensure that your document prints well in the local environment. Using:

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

works fine in the U.S. but may fail in European locales. Instead, use:

```
@page normal { size : portrait }
```

which should select an 8.5" x 11" page in the U.S. and an A4 sheet in Europe.<sup>[18]</sup>

[18] The word "normal" in the rule is the page name, of course.

Use the `margin`, `margin-top`, `margin-bottom`, `margin-left`, and `margin-right` properties within the `@page` rule to set margins for your page. Keep in mind that the browser may define margins for rendering the page box within the target sheet, so your margins are in addition to those margins. The default margins for the page box are not defined and are browser-dependent.

Finally, the `marks` property is used within the `@page` rule to create crop and registration marks outside the page box on the target sheet. By default, no marks are printed. You may use one or both of the `crop` and `cross` keywords to create crop marks and registration marks, respectively, on the target print page.

#### 8.4.13.2 Left, right, and first pages

In many printing applications, authors want different page layouts for the first page of their document as well as differing formats for right and left pages in double-sided documents. CSS2 accommodates all of these cases using three pseudoclasses attached to the name of a page.

The `:first` pseudoclass applies the page format to the first page in a document. Page-layout attributes specified in the `:first` page override corresponding attributes in the general page layout. You can use the `:first` pseudoclass in conjunction with a named page layout; the appropriate first-page layout is applied if the first page of the document is rendered using the named page.

In a similar fashion, the `:left` and `:right` pseudoclasses define left and right page layouts for your document. Again, named pages can have left and right variations. The browser automatically applies appropriate left and right layouts to every page in the document, if such layouts exist.

You need not specify named pages to use any of these pseudoclasses. Indeed, most documents do not do so. For example, if you use these settings:

```
@page :first { margin-top : 3in }
```

```
@page :left { margin-left : 2in; margin-right : 1in }
```

```
@page :right { margin-left : 1in; margin-right : 2in }
```

without further intervention, the first page of your document will have a three-inch top margin (and an appropriate right and left margin, depending on how your locale defines whether the first page of a document is on the right or the left). Subsequent pages will alternate wide and narrow inner and outer margins.

### 8.4.13.3 Using named pages

Once you create a named page layout, you can use it in your document by adding the `page` property to a style that is later applied to an element in your document. If an element has a page layout that is different from that of the preceding or containing element, a page break is inserted into the document, and formatting resumes using the new page layout. When the scope of the element ends, the page layout reverts to the previous layout, with appropriate page breaks as needed.

For example, this style renders all the tables in your document on landscape pages:

```
@page { size : portrait }
```

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

```
table { page : rotated }
```

While printing, if the browser encounters a `<table>` element in your document and the current page layout is the default portrait layout, it starts a new page and prints the table on a landscape page. If non-tabular content follows the table, the browser inserts another page break, and the flow resumes on the default portrait-sized page. Several tables in a row would be rendered on a single landscape sheet, if they all fit.

#### 8.4.13.4 Controlling pagination

Unless you specify otherwise, page breaks occur only when the page format changes or when the content overflows the current page box. To otherwise force or suppress page breaks, use the `page-break-before`, `page-break-after`, and `page-break-inside` properties.

Both the `page-break-before` and `page-break-after` properties accept the `auto`, `always`, `avoid`, `left`, and `right` keywords. `auto` is the default; it lets the browser generate page breaks as needed. The keyword `always` forces a page break before or after the element, while `avoid` suppresses a page break immediately before or after the element. The `left` and `right` keywords force one or two page breaks, so that the element is rendered on a left-hand or right-hand page.

Using pagination properties is straightforward. Suppose your document has level-1 headers start new chapters, with sections denoted by level-2 headers. You'd like each chapter to start on a new, right-hand page, but you don't want section headers to be split across a page break from the subsequent content. Accordingly, you might write your CSS2 print rule:

```
h1 { page-break-before : right }
```

```
h2 { page-break-after : avoid }
```

Use only the `auto` and `avoid` values with the `page-break-inside` property. `auto` allows page breaks within the element (the default behavior), while `avoid` suppresses them. Even so, elements that are larger than the printed page get broken up; that is why the keyword is `avoid` and not `prevent`.

If you prefer that your tables not be broken across pages if possible, you would write the rule:

```
table { page-break-inside : avoid }
```



#### 8.4.13.5 Controlling widows and orphans

In typographic lingo, *orphans* are those lines of a paragraph stranded at the bottom of a page due to a page break, while *widows* are those lines remaining at the top of a page following a page break. Generally, printed pages do not look attractive with single lines of text stranded at the top or bottom. Most printers try to leave at least two or more lines of text at the top or bottom of each page.

If you want to take control of this behavior, you can apply the `widows` and `orphans` properties to an element. The value of each property is the minimum number of lines of text that can be left at the top or bottom of the page, respectively. The default is 2, meaning that the browser generates page breaks as needed to ensure that at least two lines of text from the element appear at the top or bottom of each page. You generally want to apply this property to all of the elements in your document, to ensure consistent pagination throughout.

## 8.2 Style Syntax

The syntax of a style — its "rule," as you may have gleaned from our previous examples — is very straightforward.

### 8.2.1 The Basics

A style rule is made up of at least two basic parts: a *selector*, which is the name of the HTML or XHTML markup element (tag name) that the style rule affects, followed by a curly brace (`{}`)-enclosed, semicolon-separated list of one or more style `property:value` pairs:

```
selector {property1:value1; property2:value1 value2 va
```

For instance, we might define the `color` property for the contents of all the level-1 header elements of our document to be the value `green`:

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

In this example, `h1` is the selector, which is also the name of the level-1 header element, `color` is the style property, and `green` is the value. Neat and clean.

Properties require at least one value but may have two or more values. Separate multiple values with a space, as is done for the three values that define `property2` in our first example. Some properties require that multiple values be separated with commas.

Current styles-conscious browsers ignore letter case in any element of a style rule. Hence, `H1` and `h1` are the same selector, and `COLOR`, `color`, `CoLoR`, and `cOLoR` are equivalent properties. At one time, convention dictated that HTML authors write selector names in uppercase characters, such as `H1`, `P`, and `STRONG`. This convention is still common and is used in the W3C's own CSS2 document.

However, current standards dictate, particularly for XML-compliant documents, that element names be identical to their respective DTD definitions. With XHTML, for instance, all element names (e.g., `h1`, `p`, or `strong`) are lowercase, so their respective CSS2 selectors must be in lowercase. We'll abide by these latter conventions.

Any valid element name (a tag name minus its enclosing `<` and `>` characters and attributes) can be a selector. You may include more than one tag name in the list of selectors, as we explain in the following sections.

## 8.2.2 Multiple Selectors

When separated by commas, all the elements named in the selector list are affected by the property values in the style rule. This makes life easy for authors. For instance:

```
h1, h2, h3, h4, h5, h6 {text-align: center}
```

does exactly the same thing as:

```
h1 {text-align: center}
```

```
h2 {text-align: center}
```

```
h3 {text-align: center}
```

```
h4 {text-align: center}
```

```
h5 {text-align: center}
```

```
h6 {text-align: center}
```

Both styles tell the browser to center the contents of header levels 1-6. Clearly, the first version is easier to type, understand, and modify. It also takes less time and fewer resources to transmit across a network.

### 8.2.3 Contextual Selectors

Normally, the styles-conscious browser applies document-level or imported styles to a tag's contents wherever they appear in your document, without regard to context. However, the CSS2 standard defines a way to have a style applied only when a tag occurs within a certain context within a document, such as when it is nested within other tags.

To create a contextual selector, list the tags in the order in which they should be nested in your document, outermost tag first. When that nesting order is encountered by the browser, the style properties are applied to the last tag in the list.

For example, here's how you might use contextual styles to create a classic outline, complete with uppercase Roman numerals for the outer level, capital letters for the next level, Arabic numerals for the next, and lowercase letters for the innermost level:

```
ol li {list-style: upper-roman}


ol ol li {list-style: upper-alpha}

ol ol ol li {list-style: decimal}

ol ol ol ol li {list-style: lower-alpha}
```

According to the example style sheet, when the styles-conscious browser encounters the `<li>` tag nested within one `<ol>` tag, it uses the `upper-roman` value for the `list-style` property of the `<li>` tag. When it sees an `<li>` tag nested within two `<ol>` tags, the browser uses the `upper-alpha` list style. Nest an `<li>` tag within three and four `<ol>` tags, and you'll see the `decimal` and `lower-alpha` list styles, respectively. That's exactly what Netscape does, as shown in [Figure 8-1](#) (Internet Explorer does the same thing). Compare [Figure 8-1](#) with using the `<ol>` tag's `type` attribute to achieve similar effects, as shown in [Figure 7-7](#).

**Figure 8-1. Nested ordered list styles**



figs/htm5\_0801.gif

Similarly, you may impose a specific style on tags related only by context. For instance, this contextual style definition colors the emphasis (`<em>`) tag's contents red only when it appears inside a level-1 header tag (`<h1>`), not elsewhere in the document:

```
h1 em {color: red}
```

If there is potential ambiguity between two contextual styles, the more specific context prevails.

Like individual tags, you may have several contextual selectors mixed with individual selectors, all separated by commas, sharing the same list of style declarations. For example:

```
h1 em, p strong, address {color: red}
```

means you'll see red whenever the `<em>` tag appears within an `<h1>` tag, when the `<strong>` tag appears within a `<p>` tag, and for the contents of the `<address>` tag.

The nesting need not be exact to match the rule. For example, if you nest the `<strong>` tag within a `<ul>` tag within a `<p>` tag, you'll still match the rule for `p strong` that we defined above. If a particular nesting matches several style rules, the most specific rule is used. For example, if you defined two contextual selectors:

```
p strong {color: red}
```

```
p ul strong {color: blue}
```

and use the sequence `<p><ul><strong>` in your document, the second, more specific rule applies, coloring the contents of the `<strong>` tag blue.

## 8.2.4 Universal, Child, and Adjacent Selectors

The CSS2 standard defines additional patterns for selectors besides commas and spaces, as illustrated in the following examples:

```
* {color: purple; font: ZapfDingBats}
```

```
ol > li {font-size: 200%; font-style: italic}
```

```
h1 + h2 {margin-top: +4mm}
```

In the first example, the universal asterisk selector applies the style to all elements of your document, so that any text gets displayed in Zapf Dingbat characters.<sup>[4]</sup> The second example selects a particular child/parent relationship; in this case, items in an ordered list. The third example illustrates the adjacent selector type, which selects for one tag immediately following another in your document. In this case, the special selector adds vertical space to instances in which your document has a level-2 header immediately following a level-1 header.

<sup>[4]</sup> Assuming, of course, that the style is not overridden by a subsequent rule.

## 8.2.5 Attribute Selectors

It is possible to attach a style to only those HTML/XHTML elements that have specific attributes. You do this by listing the desired attributes in square brackets ( `[]` ) next to the element name, before the style definition:

```
div[align] { font-style: italic }

div[align=left] {font-style: italic }

div[title~="bibliography"] { font-size: smaller }

div[lang|="en"] {color: green }
```

The first example is the simplest: it italicizes the subsequent text contents of only those `<div>` tags that contain the `align` attribute, regardless of the value assigned to the attribute. The second example is a bit pickier; it matches only `<div>` tags whose `align` attributes are set to `left`.

The third example matches any `<div>` tag whose `title` attribute contains the word "bibliography," specifically delimited by one or more spaces. Partial word matches do not count; if you used `div[title~="a"]`, you would match only `<div>` tags whose `title` attributes contained a single "a" delimited by spaces (or at the beginning or end of the title).

The final example is used almost exclusively for matching groups of languages specified in the `lang` attribute. It matches any `<div>` tag whose `lang` attribute is set to a hyphen-separated list of words, beginning with "en." This example would match attributes such as `lang=en`, `lang=en-us`, or `lang=en-uk`.

You can combine the universal selector with attribute selectors to match any element with a specific attribute. For example:

```
*[class=comment] { display: none }
```

would hide all the elements in your document whose `class` attributes are set to `comment`.

Netscape Version 6 supports attribute selectors; Internet Explorer does not.

## 8.2.6 Pseudoelements

There are elemental relationships in your documents that you cannot explicitly tag. The drop-cap is a common print style, but how do you select the first letter in a paragraph? There are ways, but you have to identify each instance separately. There is no tag for the first line in a paragraph. And there are occasions where you might want the browser to automatically generate content, such as to add the prefix "Item #" and automatically number each item in an ordered list.

CSS2 introduces four new pseudoelements that let you define special relationships and styles for their display (`:first-line`, `:first-letter`, `:before`, and `:after`). Declare each as a colon-separated suffix of a standard markup element. For example:

```
p:first-line {font-size: 200%; font-style: italic}
```

means that the browser should display the first line of each paragraph italicized and twice as large as the rest of the text. Similarly:

```
p:first-letter {font-size: 200%; float: left}
```

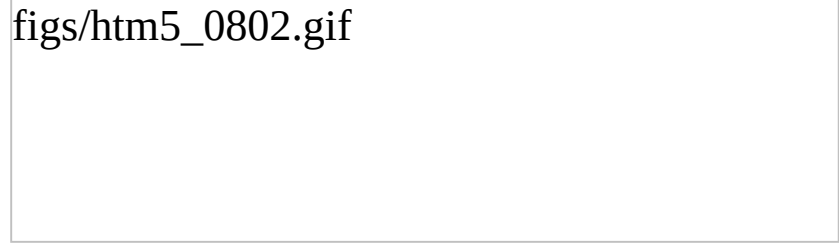
tells the browser to make the first letter of a paragraph twice as large as the remaining text and to float the letter to the left, allowing the first two lines of the paragraph to float around the larger initial letter (see [Figure 8-2](#)).<sup>[5]</sup>

[5] The properties that can be specified for the `:first-letter` and `:first-line` pseudoelements are the `font`, `color`, and `background` properties, `text-decoration`, `vertical-align`, `text-transform`, `line-height`, and `clear`. In addition, the `:first-letter` pseudoelement accepts the `margin` properties, `padding` properties, `border` properties, and `float`. The `:first-line` pseudoelement also accepts the `word-spacing` and `letter-spacing` properties.

**Figure 8-2. Use the first-letter pseudoelement to select the first letter of text within a tag's content**



figs/htm5\_0802.gif



The `:before` and `:after` pseudoelements let you identify where in your document you insert generated content such as list numbers and special lead-in headers. Hence, these pseudoelements go hand in hand with the CSS2 `content` and `counter` properties. To whet your appetite, consider this example:

```
ol {counter-reset: item}

ol li:before {content: "Item #" counter(item) " ";
               counter-increment: item}
```

Internet Explorer 5 and later and Netscape 6 support the `:first-line` and `:first-letter` pseudoelements, but only Netscape 6 supports the `:before` and `:after` ones.

## 3.7 The Document Header

The document header describes the various properties of the document, including its title, position within the Web, and relationship with other documents. Most of the data contained within the document header is never actually rendered as content visible to the user.

### 3.7.1 The `<head>` Tag

The `<head>` tag serves to encapsulate other header tags. Place it at the beginning of your document, just after the `<html>` tag and before the `<body>` or `<frameset>` tag. Both the `<head>` tag and its corresponding ending `</head>` tag can be unambiguously inferred by the browser and so can be safely omitted from an HTML, but not an XHTML, document. We encourage you to include them in all your documents, since they promote readability and support document automation.

## <head>

### *Function*

Defines the document header

### *Attributes*

`dirlangprofile`

### *End tag*

`</head>`; rarely omitted in HTML

### *Contains*

*head\_content*

### *Used in*

*html\_tag*

The `<head>` tag may contain a number of other tags that help define and manage the document's content. These include, in any order of appearance: `<base>`, `<isindex>`, `<link>`, `<meta>`, `<nextid>`, `<object>`, `<script>`, `<style>`, and `<title>`.

### **3.7.1.1 The `dir` and `lang` attributes**

As we discussed in the sections about the `<html>` tag attributes, `dir` and `lang` help extend HTML and XHTML to an international audience. [\[Section 3.6.1.1\]](#) [\[Section 3.6.1.2\]](#)

### **3.7.1.2 The `profile` attribute**

Often, the header of a document contains a number of `<meta>` tags used

to convey additional information about the document to the browser. In the future, authors may use predefined profiles of standard document metadata to better describe their documents. The `profile` attribute supplies the URL of the profile associated with the current document.

The format of a profile and how it might be used by a browser are not yet defined; this attribute is primarily a placeholder for future development.

### 3.7.2 The `<title>` Tag

The `<title>` tag does exactly what you might expect: the words you place inside its start and end tags define the title for your document. (This stuff is pretty much self-explanatory and easier than you might think at first glance.) The title is used by the browser in some special manner, most often placed in the browser window's title bar or on a status line. Usually, too, the title becomes the default name for a link to the document if the document is added to a link collection or to a user's favorites or bookmarks list.

## **<title>**

### *Function*

Defines the document title

### *Attributes*

`dirlang`

### *End tag*

`</title>`; never omitted

### *Contains*

*plain\_text*

### *Used in*

*head\_content*

The `<title>` tag is the only thing required within the `<head>` tag. Since the `<head>` tag itself and even the `<html>` tag can safely be omitted, the `<title>` tag could be the first line within a valid HTML document. Beyond that, most browsers will even supply a generic title for documents lacking a `<title>` tag, such as the document's filename, so you don't even have to supply a title. That goes a bit too far even for our down-and-dirty tastes, though. No respectable author should serve up a document missing the `<title>` tag and a title.

Browsers do not specially format title text, and they ignore anything other than text inside the title start and end tags. For instance, they will ignore any images or links to other documents.

Here's an even barer barebones example of a valid HTML document, to highlight the header and title tags; watch what happens when Netscape displays it in [Figure 3-2](#):

```
<html>

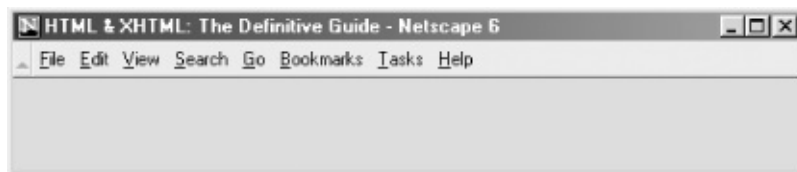
<head>

<title>HTML and XHTML: The Definitive Guide</title>

</head>

</html>
```

**Figure 3-2. What's in a <title>?**



### **3.7.2.1 What's in a title?**

Selecting the right title is crucial to defining a document and ensuring that it can be effectively used on the Web.

Keep in mind that users can access each of the documents in your collection in nearly any order and independently of one another. Each document's title should therefore define the document both within the context of your other documents and on its own merits.

Titles that include references to document sequencing are usually inappropriate. Simple titles, like "Chapter 2" or "Part VI," do little to help a user understand what the document might contain. More descriptive titles, such as "Chapter 2: Advanced Square Dancing" or "Part VI: Churchill's Youth and Adulthood," convey both a sense of place within a larger set of documents and specific content that invites the reader to read on.

Self-referential titles also aren't very useful. A title like "Home Page" is

completely content-free, as are titles like "Feedback Page" or "Popular Links." You want a title to convey a sense of content and purpose so that users can decide, based upon the title alone, whether to visit that page or not. "The Kumquat Lover's Home Page" is descriptive and likely to draw in lovers of the bitter fruit, as are "Kumquat Lover's Feedback Page" and "Popular Links Frequented by Kumquat Lovers."

People spend a great deal of time creating documents for the Web, often only to squander that effort with an uninviting, ineffective title. As special software that automatically collects links for users becomes more prevalent on the Web, the only descriptive phrases associated with your pages when they are inserted into some vast link database will be the titles you choose for them. We can't emphasize this enough: take care to select descriptive, useful, context-independent titles for each of your documents.

### 3.7.2.2 The `dir` and `lang` attributes

The `dir` and `lang` attributes help extend HTML and XHTML to an international audience. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)]

### 3.7.3 Related Header Tags

Other tags you may include within the `<head>` tag deal with specific aspects of document creation, management, linking, automation, or layout. That's why we only mention them here and describe them in greater detail in other, more appropriate sections and chapters of this book. Briefly, the special header tags are:

`<base>` and `<link>`

Define the current document's base location and relationship to other documents. [[<base>](#)] [[<link>](#)]

`<isindex>`

Deprecated in HTML 4, the `<isindex>` tag at one time could be used to create automatic document indexing forms, allowing users to search databases of information using the current document as a querying tool. [[<isindex>](#)]

`<nextid>`

Not supported in HTML 4 or XHTML, the `<nextid>` tag tried to make creation of unique labels easier when using document automation tools. [[<nextid>](#)]

`<meta>`

Provides additional document data not supplied by any of the other `<head>` tags. [[<meta>](#)]

`<object>`

Defines methods by which nonstandard objects can be rendered by the browser. [[Section 12.2.1](#)]

`<script>`

Defines one or more scripts that can be invoked by elements within the document. [[Section 12.3.1](#)]

`<style>`

Lets you create CSS properties to control body-content display characteristics for the entire document. [[<style>](#)]



## 11.1 An Overview of Frames

Figure 11-1 (shown in [Section 11.3.1.2](#)) is a simple example of a frame display. It shows how the document window may be divided into columns and rows of individual frames separated by rules and scrollbars. Although it is not immediately apparent in the example, each frame in the window contains an independent document. Frames may contain any valid content the browser is capable of displaying, including XHTML documents and multimedia. If the frame's contents include a hyperlink that the user selects, the new document's contents even another frame document may replace that same frame, another frame's content, or the entire browser window.

Frames are enabled with a special frame document. Its contents do not get displayed. Rather, the frame document contains extension tags that tell the browser how to divide its main display window into discrete frames and what documents go inside the frames.

The individual documents referenced and displayed in the frame document window act independently, to a degree; the frame document controls the entire window. You can, however, direct one frame's document to load new content into another frame. That's done by attaching a name to a frame and targeting the named frame with a special attribute for the hyperlink `<a>` tag.

## 5.2 Inserting Images in Your Documents

One of the most compelling features of HTML and XHTML is their ability to include images with your document text, either as an intrinsic components of the document (online images), as separate documents specially selected for download via hyperlinks, or as a background for your document. When judiciously added to the body content, images static and animated icons, pictures, illustrations, drawings, and so on can make your documents more attractive, inviting, and professional looking, as well as informative and easy to browse. You may also specially enable an image so that it becomes a visual map of hyperlinks. When used to excess, however, images make your document cluttered, confusing, and inaccessible, as well as unnecessarily lengthening the time it takes for users to download and view your pages.

### 5.2.1 Understanding Image Formats

Neither HTML nor XHTML prescribes an official format for images. However, the popular browsers specifically accommodate certain image formats: GIF and JPEG, in particular (see the following sections for explanations). Most other multimedia formats require special accessory applications that each browser owner must obtain, install, and successfully operate to view the special files. So it's not too surprising that GIF and JPEG are the *de facto* image standards on the Web.

Both image formats were already in widespread use before the Web came into being, so there's lots of supporting software out there to help you prepare your graphics for either format. However, each has its own advantages and drawbacks, including features that some browsers exploit for special display effects.

#### 5.2.1.1 GIF

The Graphics Interchange Format (GIF) was first developed for image

transfer among users of the CompuServe online service. The format has several features that make it popular for use in HTML and XHTML documents. Its encoding is cross-platform, so that with appropriate GIF-decoding software (included with most browsers), the graphics you create and make into a GIF file on a Macintosh, for example, can be loaded into a Windows-based PC, decoded, and viewed without a lot of fuss. The second main feature is that GIF uses special compression technology that can significantly reduce the size of the image file for faster transfer over a network. GIF compression is "lossless," too; none of an image's original data is altered or deleted, so the uncompressed and decoded image exactly matches its original. Also, GIF images can be easily animated.

Even though GIF image files invariably have the *.gif* (or *.GIF*) filename suffix, there actually are two GIF versions: the original GIF87 and an expanded GIF89a, which supports several new features including transparent backgrounds, interlaced storage, and animation that are popular with web authors (see [Section 5.2.1.2](#)). The currently popular browsers support both GIF versions, which use the same encoding scheme that maps 8-bit pixel values to a color table, for a maximum of 256 colors per image. Most GIF images have even fewer colors; there are special tools to simplify the colors in more elaborate graphics. By simplifying the GIF images, you create a smaller color map and enhance pixel redundancy for better file compression and, consequently, faster downloading.

However, because of the limited number of colors, a GIF-encoded image is not always appropriate, particularly for photorealistic pictures (see the discussion of JPEG in [Section 5.2.1.3](#)). GIFs make excellent icons, reduced-color images, and drawings.

Because most graphical browsers explicitly support the GIF format, it is currently the most widely accepted image-encoding format on the Web. It is acceptable for both inline images and externally linked ones. When in doubt as to which image format to use, choose GIF.<sup>[2]</sup> It will work in almost any situation.

[2] We cannot resist the temptation to point out that choosy authors choose GIF.

### 5.2.1.2 Interlacing, transparency, and animation

GIF images can be made to perform three special tricks: interlacing, transparency, and animation. With interlacing, a GIF image seemingly materializes on the display, rather than progressively flowing onto it from top to bottom. Normally, a GIF-encoded image is a sequence of pixel data in order, row by row, from the top to the bottom of the image. While the common GIF image renders onscreen like pulling down a window shade, interlaced GIFs open like a venetian blind. That's because interlacing sequences every fourth row of the image. Users get to see a full image top to bottom, albeit fuzzy in a quarter of the time it takes to download and display the remainder of the image. The resulting quarter-done image usually is clear enough so that users with slow network connections can evaluate whether to take the time to download the remainder of the image file.

Not all graphical browsers, although able to display an interlaced GIF, are actually able to display the materializing effects of interlacing. With those that do, users still can defeat the effect by choosing to delay image display until after download and decoding. Older browsers, on the other hand, always download and decode images before display and don't support the effect at all.

Another popular effect available with GIF images GIF89a-formatted images, that is is the ability to make a portion of them transparent, so that what's underneath (usually, the browser window's background) shows through. The transparent GIF image has one color in its color map designated as the background color. The browser simply ignores any pixel in the image that uses that background color, thereby letting the display window's background show through. By carefully cropping its dimensions and by using a solid, contiguous background color, a transparent image can be made to seamlessly meld into or float above a page's surrounding content.

Transparent GIF images are great for any graphic that you want to meld

into the document and not stand out as a rectangular block. Transparent GIF logos are very popular, as are transparent icons and dingbats any graphic that should appear to have an arbitrary, natural shape. You may also insert a transparent image inline with conventional text to act as a special character glyph within conventional text.

The down side to transparency is that the GIF image will look lousy if you don't remove its border when it is included in a hyperlink anchor (`<a>`) tag or is otherwise specially framed. And content flow happens around the image's rectangular dimensions, not adjacent to its apparent shape. That can lead to unnecessarily isolated images or odd-looking sections in your web pages.

The third unique trick available with GIF89a-formatted images is the ability to do simple frame-by-frame animation. Using special GIF-animation software utilities, you may prepare a single GIF89a file that contains a series of GIF images. The browser displays each image in the file, one after the other, something like the page-flipping animation booklets we had and perhaps drew as kids. Special control segments between each image in the GIF file let you set the number of times the browser runs through the complete sequence (looping), how long to pause between each image, whether the image space gets wiped to background before the browser displays the next image, and so on. By combining these control features with those normally available for GIF images, including individual color tables, transparency, and interlacing, you can create some very appealing and elaborate animations.<sup>[3]</sup>

[3] Songline Studios has published an entire book dedicated to GIF animation: *GIF Animation Studio*, by Richard Koman.

Simple GIF animation is powerful for one other important reason: you don't need to specially program your HTML documents to achieve animation. But there is one major down side that limits their use for anything other than small, icon-sized, or thin bands of space in the browser window: GIF animation files get large fast, even if you are careful not to repeat static portions of the image in successive animation cells. And if you have several animations in one document, download delays may and usually will annoy the user. If there is any feature that deserves

close scrutiny for excess, it's GIF animation.

Any and all GIF tricks interlacing, transparency, and animation don't just happen; you need special software to prepare the GIF file. Many image tools now save your creations or acquired images in GIF format, and most now let you enable transparency and make interlaced GIF files. There also are a slew of shareware and freeware programs specialized for these tasks, as well as for creating GIF animations. Look into your favorite Internet software archives for GIF graphics and conversion tools, and see [Chapter 17](#) for details on creating transparent images.

### 5.2.1.3 JPEG

The Joint Photographic Experts Group ( JPEG) is a standards body that developed what is now known as the JPEG image-encoding format. Like GIFs, JPEG images are platform-independent and specially compressed for high-speed transfer via digital communication technologies. Unlike GIF, JPEG supports tens of thousands of colors for more detailed, photorealistic digital images. And JPEG uses special algorithms that yield much higher data-compression ratios. It is not uncommon, for example, for a 200-KB GIF image to be reduced to a 30-KB JPEG image. To achieve that amazing compression, JPEG does lose some image data. However, you can adjust the degree of "lossiness" with special JPEG tools, so that although the uncompressed image may not exactly match the original, it will be close enough that most people cannot tell the difference.

Although JPEG is an excellent choice for photographs, it's not a particularly good choice for illustrations. The algorithms used for compressing and uncompressing the image leave noticeable artifacts when dealing with large areas of one color. Therefore, if you're trying to display a drawing, the GIF format may be preferable.

The JPEG format, usually designated by the *.jpg* (or *.JPG* ) filename suffix, is nearly universally understood by today's graphical browsers. On rare occasions, you'll come across an older browser that cannot directly display JPEG images.

## 5.2.2 When to Use Images

Most pictures are worth a thousand words. But don't forget that no one pays attention to a blabbermouth. First and foremost, think of your document images as visual tools, not gratuitous trappings. They should support your text content and help readers navigate your documents. Use images to clarify, illustrate, or exemplify the contents. Content-supporting photographs, charts, graphs, maps, and drawings are all natural and appropriate candidates. Product photographs are essential components in online catalogs and shopping guides, for example. And link-enabled icons and dingbats, including animated images, can be effective visual guides to internal and external resources. If an image doesn't do any of these valuable services for your document, throw it out already!

One of the most important considerations when adding images to a document is the additional delay they add to the retrieval time for a document over the network, particularly for modem connections. While a common text document might run, at most, 10 or 15 thousand bytes, images can easily extend to hundreds of thousands of bytes each. And the total retrieval time for a document is not only equal to the sum of all its component parts, but also to compounded networking overhead delays.

Depending on the speed of the connection (*bandwidth*, usually expressed as bits or bytes per second) as well as network congestion that can delay connections, a single document containing one 100-KB image may take anywhere from around 15 seconds through a 57.6-Kbps modem connection in the wee hours of the morning, when most everyone else is asleep, to well over *10 minutes* with a 9600-bps modem at noon. You get the picture?

That said, of course, pictures and other multimedia are driving Internet providers to come up with faster, better, more robust ways to deliver web content. Soon, 57.6-Kbps modem connections will go the way of the horse and carriage (as 9600-bps modems already have), to be replaced by technologies like cable modems and ADSL. Indeed, soon most



connections will attain data rates approaching or exceeding what used to be available only to the biggest users (besides costing an arm and a leg), over 1 Mb per second.

Still, as the price lowers, use goes up, so there is the issue of congestion. If you are competing for access to an overburdened server, it doesn't matter how fast your connection may be.

### **5.2.3 When to Use Text**

Text hasn't gone out of style. For some users, it is the only accessible portion of your document. We argue that, in most circumstances, your documents should be usable by readers who cannot view images or have disabled their automatic download in their browsers to improve their connections. While the urge to add images to all of your documents may be strong, there are times when pure text documents make more sense.

Documents being converted to the Web from other formats rarely have embedded images. Reference materials and other serious content often are completely usable in a text-only form.

You should create text-only documents when access speed is critical. If you know that many users will be vying for your pages, you should accommodate them by avoiding the use of images within your documents. In some extreme cases, you might provide a home (leading) page that lets readers decide between duplicate collections of your work: one containing the images and another stripped of them. (The popular browsers include special picture icons as placeholders for yet-to-be downloaded images, which can trash and muddle your document's layout into an unreadable mess.)

Text is most appropriate supporting images only, without frills or nonessential graphics if your documents are to be readily searchable by any of the many web indexing services. Images are almost always ignored by these search engines. If the major content of your pages is provided with images, very little information about your documents will find its way into the online web directories.



## 5.2.4 Speeding Image Downloads

There are several ways to reduce the overhead and delays inherent with images, besides being very choosy about which to include in your documents:

### *Keep it simple*

A full-screen, 24-bit color graphic, even when reduced in size by digital compression with one of the standard formats, such as GIF or JPEG, is still going to be a network-bandwidth hog. Acquire and use the various image-management tools to optimize image dimensions and number of colors into the fewest number of pixels. Simplify your drawings. Stay away from panoramic photographs. Avoid large, empty backgrounds in your images, as well as gratuitous borders and other space-consuming elements. Also avoid dithering (blending two colors among adjacent pixels to achieve a third color); this technique can significantly reduce the compressibility of your images. Strive for large areas of uniform colors, which compress readily in both GIF and JPEG formats.

### *Reuse images*

This is particularly true for icons and GIF animations. Most browsers cache incoming document components in local storage for the very purpose of quick, network-connectionless retrieval of data. For smaller GIF animation files, try to prepare each successive image to update only portions that change in the animation, rather than redrawing the entire image (this speeds up the animation, too).

### *Divide up large documents*

This is a general rule that includes images. Many small document segments, organized through hyperlinks (of course!) and effective tables of contents, tend to be better accepted by users than a few large documents. In general, people would rather "flip" several pages than dawdle waiting for a large one to download. (It's related to the TV channel-surfing syndrome.) One accepted rule of thumb is to

keep your documents under 50 KB each, so even the slowest connections won't overly frustrate your readers.

### *Isolate necessarily large graphics*

Provide a special link to large images, perhaps one that includes a thumbnail of the graphic, thereby letting readers decide if and when they want to spend the time downloading the full image. Since the downloaded image isn't mixed with other document components like inline images, it's also much easier for the reader to identify and save the image on her system's local storage for later study. (For details on non-inline image downloads, see [Section 5.6.2](#).)

### *Specify image dimensions*

Finally, another way to improve performance is by including the image's rectangular height and width information in its tag. By supplying those dimensions, you eliminate the extra steps the extended browsers must take to download, examine, and calculate an image's space in the document. There is a down side to this approach, however, that we explore in [[Section 5.2.6.12](#)].

## **5.2.5 JPEG or GIF?**

You may choose to use only JPEG or GIF images in your HTML documents if your sources for images or your software toolset prefers one over the other format. Both are nearly universally supported by today's browsers, so there shouldn't be any user-viewing problems.

Nevertheless, we recommend that you acquire the facilities to create and convert to both formats to take advantage of their unique capabilities. For instance, use GIF's transparency feature for icons and dingbats. Alternatively, use JPEG for large and colorful images for faster downloading.

## **5.2.6 The <img> Tag**

The `<img>` tag lets you reference and insert a graphic image into the current text flow of your document. There is no implied line or paragraph break before or after the `<img>` tag, so images can be truly "in line" with text and other content.

## <img>

### *Function*

Inserts an image into a document

### *Attributes*

`align, alt, border, class, controls (□), dir, dynsrc (□), height, hspace, id, ismap, lang, longdesc, loop (□), lowsrc (□), name (□), onAbort, onClick, onDblClick, onError, onKeyDown, onKeyPress, onKeyUp, onLoad, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, src, start (□), style, title, usemap, vspace, width`

### *End tag*

None in HTML; `</img>` or `<img ... />` in XHTML

### *Contains*

Nothing

### *Used in*

*text*

The format of the image itself is not defined by the HTML or XHTML standard, although the popular graphical browsers support GIF and JPEG images. The standards don't specify or restrict the size or dimensions of the image, either. Images may have any number of colors, but how those colors are rendered is highly browser-dependent.

Image presentation in general is very browser-specific. Images may be ignored by nongraphical browsers. Browsers operating in a constrained environment may modify the image size or complexity. And users, particularly those with slow network connections, may choose to defer image loading altogether. Accordingly, you should make sure your

documents make sense and are useful even if the images are completely removed.

The HTML version of the `<img>` tag has no end tag. With XHTML, either use `</img>` immediately following the `<img>` tag and its attributes or make the last character in the tag the end-tag slash mark: `<img src=kumquat.gif />`, for example.

#### 5.2.6.1 The `src` attribute

The `src` attribute for the `<img>` tag is required (unless you use `dynsrc` with Internet Explorer-based movies; see [Figure 5.2.7.1](#)). Its value is the image file's URL, either absolute or relative to the document referencing the image. To unclutter their document storage, authors typically collect image files into a separate folder, which they often name something like "pics" or "images." [[Section 6.2](#)]

For example, this HTML fragment places an image of a famous kumquat packing plant into the narrative text (see [Figure 5-8](#)):

```
Here we are, on day 17 of the tour, in front of the kumquat  
packing plant:
```

```
<p>
```

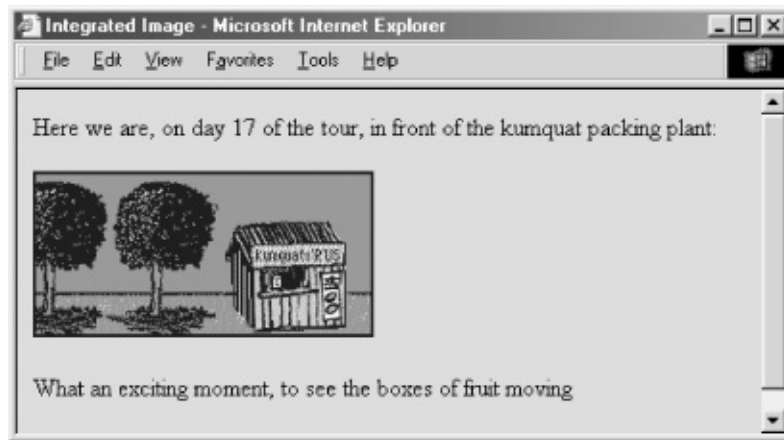
```

```

```
<p>
```

```
What an exciting moment, to see the boxes of fruit moving
```

**Figure 5-8. Image integrated with text**



In the example, the paragraph (`<p>`) tags surrounding the `<img>` tag cause the browser to render the image by itself, with some vertical space after the preceding text and before the trailing text. Text may also about the image, as we describe in [Figure 5.2.6.4](#).

### 5.2.6.2 The `lowsrc` attribute

To the benefit of users, particularly those with slow network connections, Netscape provides the `lowsrc` companion to the `src` attribute in the `<img>` tag as a way to speed up document rendering. The `lowsrc` attribute's value, like `src`, is the URL of an image file. Earlier versions of Netscape (before Version 6) would load and display the `lowsrc` image when they first encountered the `<img>` tag. Then, when the document had been completely loaded and could be read by the user, Netscape would retrieve the image specified by the `src` attribute.

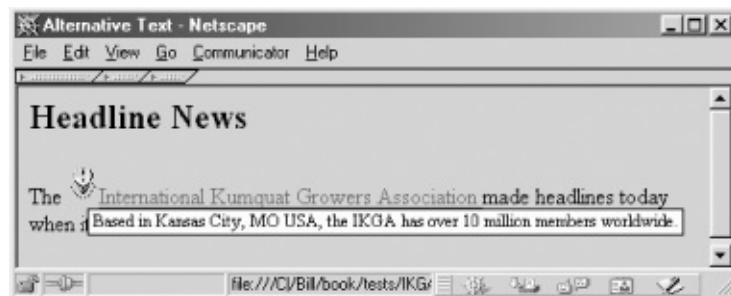
Netscape Version 6 simply uses the dimensions of the `lowsrc` image to temporarily allocate display space for the image as it renders the document. The earlier versions of Netscape also used the `lowsrc` dimensions to resize the final image, which you could exploit for some special effects. This no longer works. Instead, we recommend that you eschew the Netscape extension and explicitly allocate image space with the `height` and `width` attributes described later in this chapter.

### 5.2.6.3 The `alt` and `longdesc` attributes

The `alt` attribute specifies alternative text the browser may show if image display is not possible or is disabled by the user. It's an option, but one we highly recommend you exercise for most images in your document. This way, if the image is not available, the user still has some indication of what's missing. And for users with certain disabilities, `alt` often is the only way they can appreciate your images.

In addition, the latest browsers display the alternative description in a text box when users pass the mouse over the image. Accordingly, you might embed short, parenthetical information that pops up when users pass over a small, inline icon, such as that shown in [Figure 5-9](#).

**Figure 5-9. Contemporary graphical browsers display alt in a temporary pop-up window**



The value for the `alt` attribute is a text string of up to 1,024 characters, including spaces and punctuation. The string must be enclosed in quotation marks. The `alt` text may contain entity references to special characters, but it may not contain any other sort of markup; in particular, style tags aren't allowed.

Graphical browsers don't normally display the `alt` attribute if the image is available and the user has enabled picture downloading. Otherwise, they insert the `alt` attribute's text as a label next to an image-placeholder icon. Well-chosen `alt` labels thereby additionally support those users with graphical browsers who have disabled automatic image download because of a slow connection to the Web.

Nongraphical, text-only browsers like Lynx put the `alt` text directly into

the content flow, just like any other text element. So, when used effectively, the `alt` tag sometimes can transparently substitute for missing images. (Your text-only browser users will appreciate not being constantly reminded of their second-class web citizenship.) For example, consider using an asterisk as the `alt` attribute alternative to a special bullet icon:

```
<h3>Introduct
```

A graphical browser displays the bullet image, while in a nongraphical browser the `alt` asterisk takes the place of the missing bullet. Similarly, use `alt` text to replace special image bullets for list items. For example, the following code:

```
<ul>

  <li> Kumquat recipes 

  <li> Annual harvest dates

</ul>
```

displays the *new.gif* image with graphical browsers and the text "(New!)" with text-only browsers. The `alt` attribute uses even more complex text (see [Figure 5-10](#)):

```
Here we are, on day 17 of the tour, in front of the ku
packing plant:
```

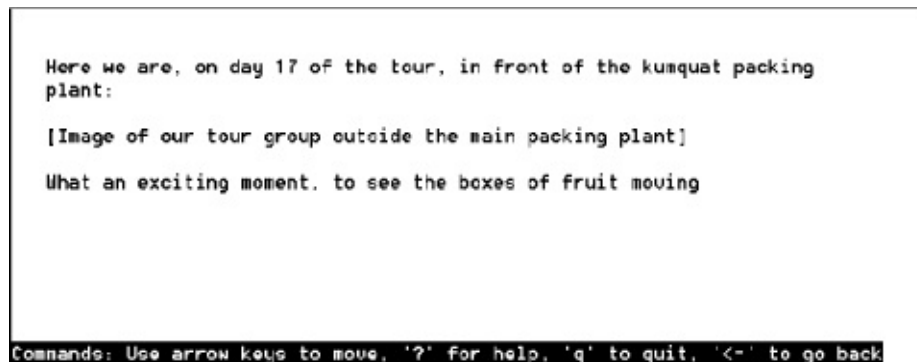
```
<p>


```



What an exciting moment, to see the boxes of fruit mov

**Figure 5-10. Text-only browsers like Lynx display an image's alt attribute text**



The `longdesc` attribute is similar to the `alt` attribute but allows for longer descriptions. The value of `longdesc` is the URL of a document containing a description of the image. If you have a description longer than 1,024 characters, use the `longdesc` attribute to link to it. Neither HTML 4 nor XHTML specifies what the content of the description must be, and no browsers currently implement `longdesc`; all bets are off when deciding how to create those long descriptions.

While `alt` is useful for users with disabilities, `longdesc` is the better option, for obvious reasons.

#### 5.2.6.4 The align attribute

The standards don't define a default alignment for images with respect to other text and images in the same line of text: you can't always predict how the text and images will look.<sup>[4]</sup> HTML images normally appear in line with a single line of text. Common print media like magazines wrap text around images, with several lines next to and abutting the image, not just a single line.

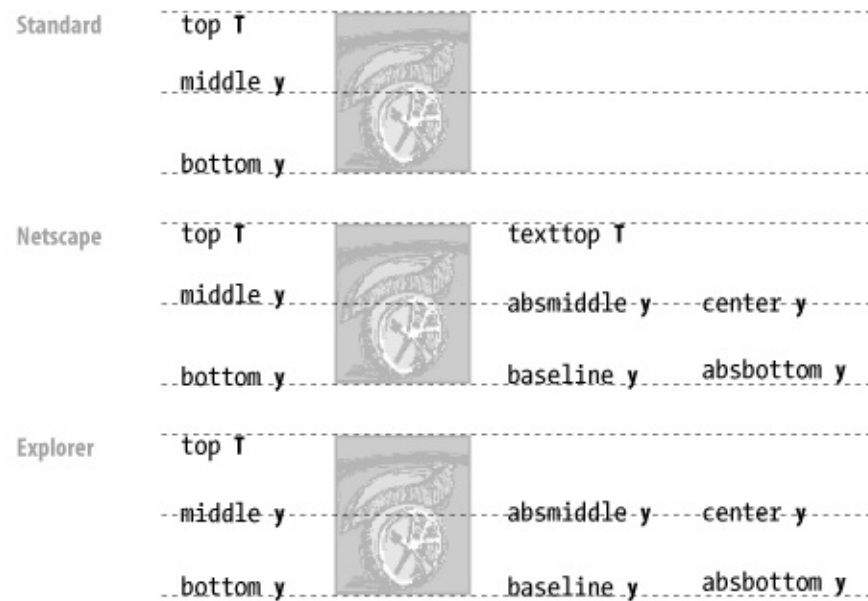
[4] Most of the popular graphical browsers insert an image so its base aligns with the

baseline of the text the same alignment specified by the attribute value of `bottom`. But document designers should assume that alignment varies between browsers and always include the desired type of image alignment.

Fortunately, document designers can exert some control over the alignment of images with the surrounding text through the `align` attribute for the `<img>` tag. The HTML and XHTML standards specify five image-alignment attribute values: `left`, `right`, `top`, `middle`, and `bottom`. The `left` and `right` values flow any subsequent text around the image, which is moved to the corresponding margin; the remaining three align the image vertically with respect to the surrounding text. Netscape adds four more vertical alignment attributes to that list `texttop`, `absmiddle`, `baseline`, and `absbottom` while Internet Explorer adds `center`.

[Figure 5-11](#) illustrates the various image-alignment options.

**Figure 5-11. Standard and browser-extended inline image alignments with text**



Alignment	Standard	Netscape	Explorer
top	●	●	●
texttop		●	
middle	●	●	●
absmiddle		●	●
center		●	●
bottom	●	●	●
baseline		●	●
absbottom		●	●

The inline image-alignment options are:

`top`

The top of the image is aligned with the top edge of the tallest item in the current line of text. If there are no other images in the current line, the top of the image is aligned with the top of the text.

`texttop`

The `align=texttop` attribute and value tells Netscape to align the top of the image with the top of the tallest text item in the current line. It is different from the `top` option, which aligns the top of the image with the top of the tallest item, image or text, in the current line. If the line contains no other images that extend above the top of the text, `texttop` and `top` have the same effect.

middle

Netscape and Internet Explorer treat the `middle` image-alignment value differently: Netscape aligns the middle of the image with the baseline of the text, regardless of other inline elements, such as another inline image (Figure 5-12); Internet Explorer aligns the middle of the image with the middle of the tallest item in the current line, text or image (Figure 5-13). Notice the alignments and differences in Figure 5-12 and Figure 5-13, particularly when only one image contains the `align` attribute. Both figures display the following HTML fragment:

```
Line of text
```

```

```

```

```

```
goes on ...
```

```
<br clear=left>
```

```
<p>
```

```
Line of text
```

```

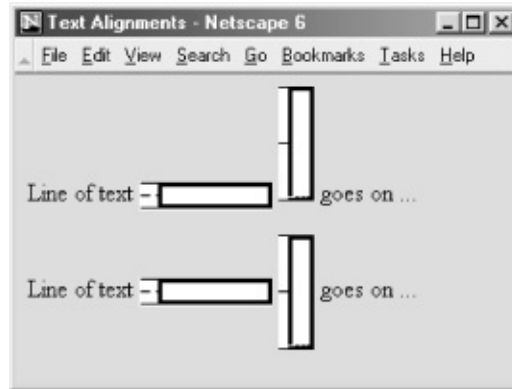
```

```

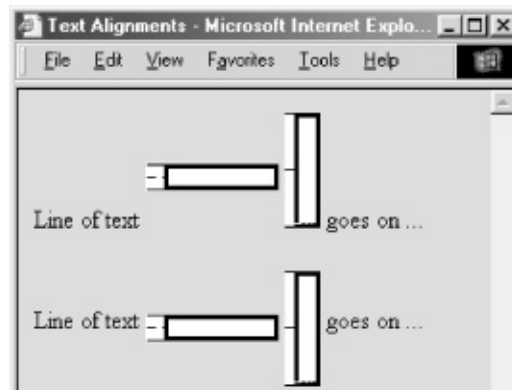
```

```
goes on ...
```

**Figure 5-12. Netscape aligns middle of image to baseline of text**



**Figure 5-13. Internet Explorer aligns middle of image to middle of tallest line element**



Also note that Internet Explorer Versions 3 and later treat `middle`, `absmiddle`, and `center` the same, whereas earlier Internet Explorer versions and Netscape distinguish between `middle` and `absmiddle` alignments. (If you are confused as to exactly what each alignment value means, please raise your hand.)

### `absmiddle`

If you set the `align` attribute of the `<img>` tag to `absmiddle`, the browser will fit the absolute middle of the image to the absolute middle of the current line. For Netscape and early versions of Internet Explorer, this is different from the common `middle` option, which aligns the middle of the image with the baseline of the current line of text (the bottom of the characters). Version 3 and later of

Internet Explorer, on the other hand, treat `absmiddle` the same as `middle` and `center`.

#### `center`

The `center` image alignment value gets treated the same as `absmiddle` by both Internet Explorer and Netscape, but note that the browsers treat `absmiddle` and `middle` differently.

#### `bottom` and `baseline` (default)

With Netscape and early versions of Internet Explorer, the `bottom` and `baseline` image-alignment values have the same effect as if you didn't include any alignment attribute at all: the browsers align the bottom of the image in the same horizontal plane as the baseline of the text. This is not to be confused with `absbottom`, which takes into account letter "descenders" like the tail on the lowercase "y." Internet Explorer Versions 3 and later, on the other hand, treat `bottom` the same as `absbottom`. (Did we see a hand up in the audience?)

#### `absbottom`

The `align=absbottom` attribute tells the browsers to align the bottom of the image with the true bottom of the current line of text. The true bottom is the lowest point in the text, taking into account descenders, even if there are no descenders in the line. A descender is the tail on a "y," for example; the baseline of the text is the bottom of the "v" in the "y" character.

Use the `top` or `middle` alignment values for best integration of icons, dingbats, or other special inline effects with the text content. Otherwise, `align=bottom` (the default) usually gives the best appearance. When aligning one or more images on a single line, select the alignment that gives the best overall appearance to your document.

### 5.2.6.5 Wrapping text around images

The `left` and `right` image-alignment values tell the browser to place an image against the left or right margin, respectively, of the current text flow. The browser then renders subsequent document content in the remaining portion of the flow adjacent to the image. The net result is that the document content following the image gets wrapped around the image.

```

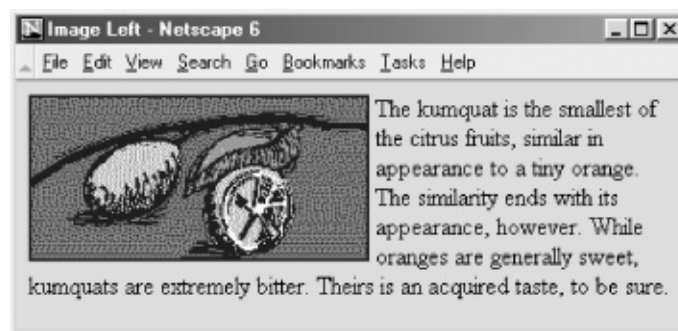
```

The kumquat is the smallest of the citrus fruits, similar to a tiny orange. The similarity ends with its appearance, however. While oranges are generally sweet,

kumquats are extremely bitter. There is an acquired taste, to be sure.

Figure 5-14 shows text flow around a left-aligned image.

**Figure 5-14. Text flow around a left-aligned image**



You can place images against both margins simultaneously (Figure 5-15), and the text will run down the middle of the page between them:

```

```

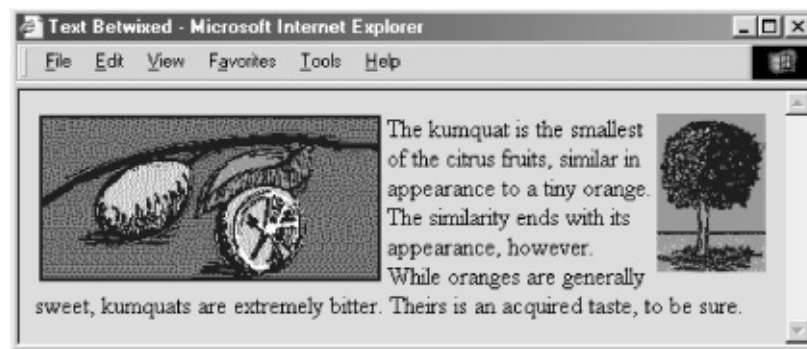
```

```

The kumquat is the smallest of the citrus fruits, similar to a tiny orange. The similarity ends with its appearance, however. While oranges are generally sweet,

tiny orange. The similarity ends with its appearance, oranges are generally sweet, kumquats are extremely bitter. That's all to be sure.

**Figure 5-15. Running text between left- and right-aligned images**



While text is flowing around an image, the left (or right) margin of the page is temporarily redefined to be adjacent to the image as opposed to the edge of the page. Subsequent images with the same alignment will stack up against each other. The following source fragment achieves that staggered image effect:

```

```

Marcia!

```
<br>
```

```

```

Jan!

```
<br>
```

```

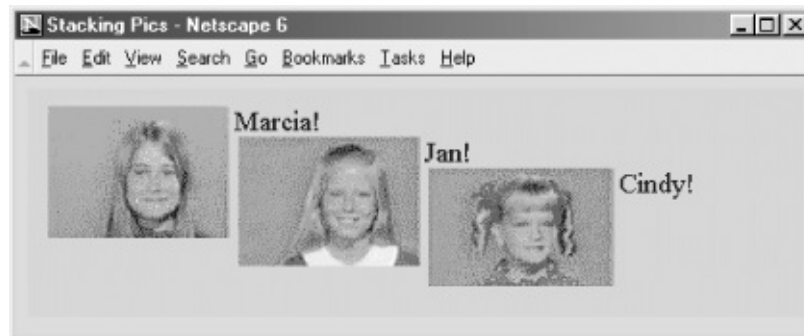
```



Cindy!

The results of this example are shown in [Figure 5-16](#).

**Figure 5-16. Three very lovely girls**



When the text flows beyond the bottom of the image, the margin returns to its former position, typically at the edge of the browser window.

#### 5.2.6.6 Centering an image

Have you noticed that you can't horizontally center an image in the browser window with the `align` attribute? The `middle` and `absmiddle` values center the image vertically with the current line, but the image is horizontally justified depending on what content comes before it in the current flow and the dimensions of the browser window.

You can horizontally center an inline image in the browser window, but only if it's isolated from surrounding content, such as by paragraph, division, or line-break tags. Then, either use the `<center>` tag or use the `align=center` attribute or center-justified style in the paragraph or division tag to center the image. For example:

Kumquats are tasty treats

`<br>`

```
<center>
```

```

```

```
</center>
```

```
that everyone should strive to eat!
```

Use the paragraph tag with its `align=center` attribute if you want some extra space above and below the centered image:

```
Kumquats are tasty treats
```

```
<p align=center>
```

```

```

```
</p>
```

```
that everyone should strive to eat!
```

### 5.2.6.7 Align and `<center>` are deprecated

The HTML 4 and XHTML standards have deprecated the `align` attribute for all tags, including `<img>`, in deference to style sheets. They've deprecated `<center>`, too. Nonetheless, the attribute and tag are very popular among HTML authors and remain well supported by the popular browsers. So, while we do expect that someday both `align` and `<center>` will disappear, it won't be anytime soon. Just don't say we didn't warn you.

What if you don't want to use `align` or `<center>`? Some authors and many of the WYSIWYG editors use HTML/XHTML tables to align content. That's one way, albeit involved (see [Chapter 10](#)). The W3C wants you to use styles. For example, use the `margin-left` style to indent the image from the left side of the display. You can read lots more about CSS in

## Chapter 8.

### 5.2.6.8 The border attribute

Browsers normally render images that also are hyperlinks (i.e., images included in an `<a>` tag) with a 2-pixel-wide colored border, indicating to the reader that the image can be selected to visit the associated document. Use the `border` attribute and a pixel-width thickness value to remove (`border=0`) or widen that image border. Be aware that this attribute, too, is deprecated in HTML 4 and XHTML, in deference to style sheets, but continues to be well supported by the popular browsers.

Figure 5-17 shows you the thick and thin of image borders, as rendered by Internet Explorer from the following XHTML source:

```
<a href="test.html">

</a>

<a href="test.html">

</a>

<a href="test.html">

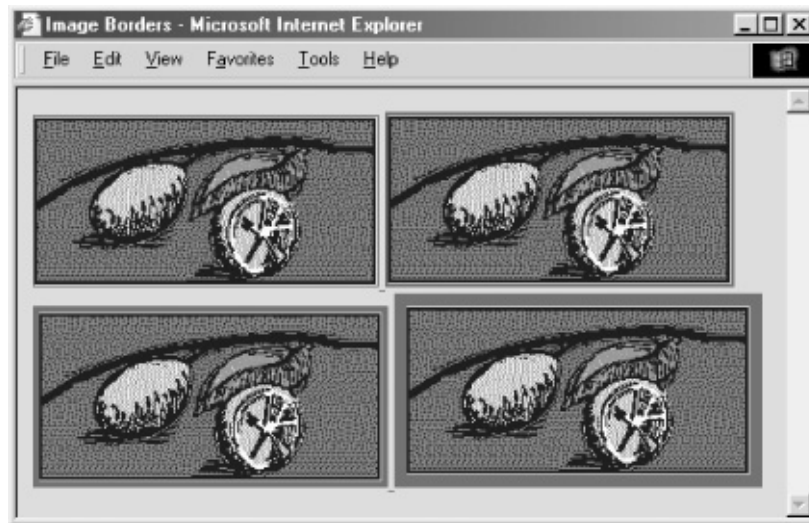
</a>

<a href="test.html">

    
```

</a>

**Figure 5-17. The thick and thin of image borders**



#### **5.2.6.9 Removing the image border**

You can eliminate the border around an image hyperlink altogether with the `border=0` attribute within the `<img>` tag. For some images, particularly image maps, the absence of a border can improve the appearance of your pages. Images that are clearly link buttons to other pages may also look best without borders.

Be careful, though, that by removing the border, you don't diminish your page's usability. No border means you've removed a common visual indicator of a link, making it less easy for your readers to find the links on the page. Browsers will change the mouse cursor as the reader passes it over an image that is a hyperlink, but you should not assume they will, nor should you make readers test your borderless images to find hidden links.

We strongly recommend that with borderless images you use some additional way to let your readers know to click the images. Even

including simple text instructions will go a long way toward making your pages more accessible to readers.

#### 5.2.6.10 The height and width attributes

Ever watch the display of a page's contents shift around erratically while the document is loading? That happens because the browser readjusts the page layout to accommodate each loaded image. The browser determines the size of an image and, hence, the rectangular space to reserve for it in the display window by retrieving the image file and extracting its embedded height and width specifications. The browser then adjusts the page's display layout to insert that picture in the display.

[5] This is not the most efficient way to render a document, since the browser must sequentially examine each image file and calculate its screen space before rendering adjacent and subsequent document content. That can significantly increase the amount of time it takes to render the document and disrupt reading by the user.

[5] Another reminder that images are separate files, which are loaded individually and in addition to the source document.

A more efficient way for authors to specify an image's dimensions is with the `height` and `width` `<img>` attributes. That way, the browser can reserve space before actually downloading an image, speeding document rendering and eliminating the content shifting. Both attributes require an integer value that indicates the image size in pixels; the order in which they appear in the `<img>` tag is not important.

#### 5.2.6.11 Resizing and flood-filling images

A hidden feature of the `height` and `width` attributes is that you don't need to specify the actual image dimensions; the attribute values can be larger or smaller than the actual size of the image. The browser automatically scales the image to fit the predefined space. This gives you a down-and-dirty way of creating thumbnail versions of large images and

a way to enlarge very small pictures. Be careful, though: the browser still must download the entire file, no matter what its final rendered size is, and you will distort an image if you don't retain its original height versus width proportions.

Another trick with `height` and `width` provides an easy way to flood-fill areas of your page and can also improve document performance.

Suppose you want to insert a colored bar across your document.<sup>[6]</sup> Rather than creating an image to fill the full dimensions, create one that is just 1 pixel high and wide and set it to the desired color. Then use the `height` and `width` attributes to scale it to the larger size:

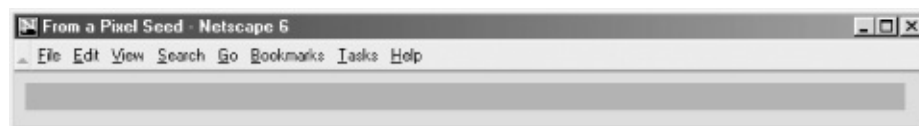
[6] This is one way to create colored horizontal rules, since Netscape doesn't support the `color` attribute for the `<hr>` tag.

```

```

The smaller image downloads much faster than a full-scale one, and the `width` and `height` attributes have Netscape create the desired bright-red colored bar after the tiny image arrives at the browser (see [Figure 5-18](#)).

**Figure 5-18. This colored horizontal bar was made from a one-pixel image**



One last trick with the `width` attribute is to use a percentage value instead of an absolute pixel value. This causes the browser to scale the image to a percentage of the document window width. Thus, to create a colored bar 20 pixels high and the width of the window, you could use:

```

```

As the document window changes size, the image will change size as

well.

If you provide a percentage `width` and omit the `height`, the browser will retain the image's aspect ratio as it grows and shrinks. This means that the height will always be in the correct proportion to the width, and the image will display without distortion.

#### 5.2.6.12 Problems with height and width

Although the `height` and `width` attributes for the `<img>` tag can improve performance and let you perform neat tricks, there is a knotty down side to using them. The browser sets aside the specified rectangle of space to the prescribed dimensions in the display window, even if the user has turned off automatic download of images. What the user often is left with is a page full of semi-empty frames with meaningless picture-placeholder icons inside. The page looks terribly unfinished and is mostly useless. Without accompanying dimensions, on the other hand, the browser simply inserts a placeholder icon inline with the surrounding text, so at least there's something there to read in the display.

We don't have a solution for this dilemma, other than to insist that you use the `alt` attribute with some descriptive text so that users at least know what they are missing (see [Section 5.2.6.3](#)). We do recommend that you include these size attributes, because we encourage any practice that improves network performance.

#### 5.2.6.13 The `hspace` and `vspace` attributes

Graphical browsers usually don't give you much space between an image and the text around it. And unless you create a transparent image border that expands the space between them, the typical 2-pixel buffer between an image and adjacent text is just too close for most designers' comfort. Add the image into a hyperlink, and the special colored border will negate any transparent buffer space you labored to create, as well as drawing even more attention to how close the adjacent text butts up against the image.

The `hspace` and `vspace` attributes can give your images breathing room. With `hspace`, you specify the number of pixels of extra space to leave between the image and text on the left and right sides of the image; the `vspace` value is the number of pixels on the top and bottom:

```

```

The kumquat is the smallest of the citrus fruits, similar in appearance to a tiny orange. The similarity ends with appearance, however. While oranges are generally sweet, kumquats are extremely bitter. There is an acquired taste to be sure. Most folks, at first taste, wonder how you ever eat another, let alone enjoy it!

```
<p>
```

```

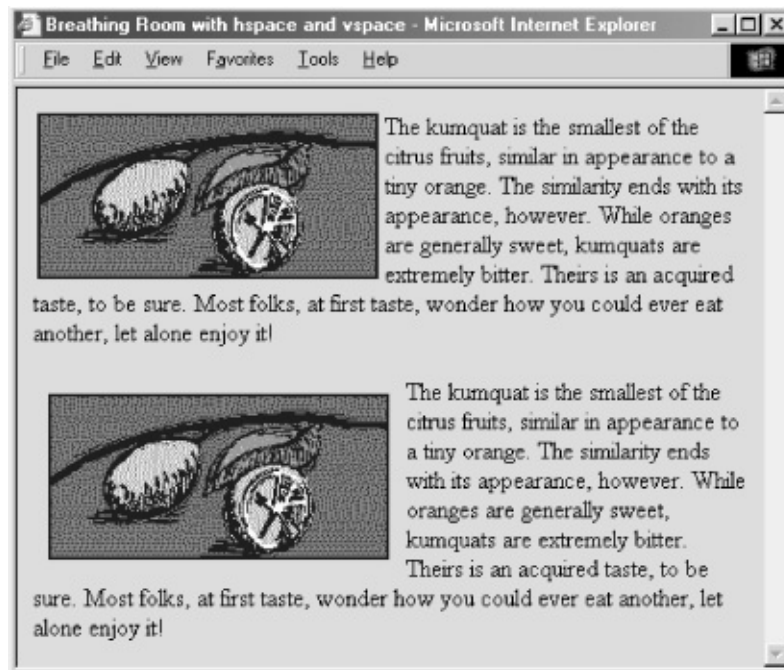
```

The kumquat is the smallest of the citrus fruits, similar in appearance to a tiny orange. The similarity ends with appearance, however. While oranges are generally sweet, kumquats are extremely bitter. There is an acquired taste to be sure. Most folks, at first taste, wonder how you ever eat another, let alone enjoy it!

**Figure 5-19** shows the difference between two wrapped images.



**Figure 5-19. Improve image/text interfaces with vspace and hspace**



We're sure you'll agree that the additional space around the image makes the text easier to read and the overall page more attractive.

#### 5.2.6.14 The ismap and usemap attributes

The `ismap` and `usemap` attributes for the `<img>` tag tell the browser that the image is a special mouse-selectable visual map of one or more hyperlinks, commonly known as an *image map*. The `ismap` style of image maps, known as a *server-side* image map, may be specified only within an `<a>` tag hyperlink. [[<a>](#)]

For example (notice the redundant attribute and value, as well as the trailing end-tag slash mark in the `<img>` tag, which are telltale signs of XHTML):

```
<a href="/cgi-bin/images/map2">
```

```
  
```

</a>

The browser automatically sends the x,y position of the mouse (relative to the upper-left corner of the image) to the server when the user clicks somewhere on the `ismap` image. Special server software (the `/cgi-bin/images/map2` program, in this example) may then use those coordinates to determine a response.

The `usemap` attribute provides a *client-side* image-map mechanism that effectively eliminates server-side processing of the mouse coordinates and its incumbent network delays and problems. Using special `<map>` and `<area>` tags, HTML authors provide a map of coordinates for the hyperlink-sensitive regions in the `usemap` image, along with related hyperlink URLs. The value of the `usemap` attribute is a URL that points to that special `<map>` section. The browser on the user's computer translates the coordinates of a click of the mouse on the image into some action, including loading and displaying another document. [`<map>`] [`<area>`]

For example, the following source specially encodes the 100 x 100-pixel `map2.gif` image into four segments, each of which, if clicked by the user, links to a different document. Notice that we've included, validly, the `ismap` image-map processing capability in the example `<img>` tag so that users of other, `usemap`-incapable browsers have access to the alternative, server-side mechanism to process the image map:

```
<a href="/cgi-bin/images/map2">
```

```
    
```

```
</a>
```

```
...
```

```
<map name="map2">
```

```
<area coords="0,0,49,49" href="link1.html">

<area coords="50,0,99,49" href="link2.html">

<area coords="0,50,49,99" href="link3.html">

<area coords="50,50,99,99" href="link4.html">

</map>
```

Geographical maps make excellent `ismap` and `usemap` examples: browsing a nationwide company's pages, for instance, the users might click on their home towns on a map to get the addresses and phone numbers for nearby retail outlets. The advantage of the `usemap` client-side image-map processing is that it does not require a server or special server software and so, unlike the `ismap` mechanism, can be used in non-web (networkless) environments, such as local files or CD-ROMs.

Please read our more complete discussion of anchors and links, including image maps within links, in [Section 6.5](#).

#### **5.2.6.15 The class, dir, event, id, lang, style, and title attributes**

Several nearly universal attributes give you a common way to identify (`title`) and label (`id`) the image tag's contents for later reference or automated treatment, to change the contents' display characteristics (`class`, `style`), to reference the language (`lang`) used, and to specify the direction in which the text should flow (`dir`). And, of course, there are all the user events that may happen in and around the tagged contents that the browser senses and that you may react to via an on-event attribute and some programming. [[Section 8.1.1](#)] [[Section 8.3](#)]

Of these many HTML 4 and XHTML attributes, `id` is the most important. It lets you label the image for later access by a program or browser operation (see [Chapter 12](#)). [[Section 4.1.1.4](#)]

The remaining attributes have questionable meaning in context with `<img>`. Granted, there are a few style-sheet options available that may influence an image's display, and it's good to include a title (although `alt` is better). However, it's hard to imagine what the influence of language (`lang`) or its presentation direction (`dir`) might have on an image. [\[Section 3.6.1.1\]](#) [\[Section 3.6.1.2\]](#) [\[Section 4.1.1.4\]](#)

#### 5.2.6.16 The `name`, `onAbort`, `onError`, `onLoad` and other event attributes

There are four `<img>` attributes originally supported by Netscape and now by Internet Explorer 6 that enable you to use JavaScript to manipulate images. The first is the `name` attribute.<sup>[7]</sup> Now redundant with the `id` attribute, `name` lets you label the image so that it can be referenced by a JavaScript applet. For example:

[7] HTML Version 4.01 and XHTML have adopted the `name` attribute, too.

```

```

lets you later refer to that picture of a kumquat as simply "kumquat" in a JavaScript applet, perhaps to erase or otherwise modify it. You cannot individually manipulate an image with JavaScript if it is not named or doesn't have an associated `id`.

The other three attributes let you provide some special JavaScript event handlers. The value of each attribute is a chunk of JavaScript code, enclosed in quotation marks; it may consist of one or more JavaScript expressions, separated by semicolons.

The popular browsers invoke the `onAbort` event handler if the user stops loading an image, usually by clicking the browser's Stop button. You might, for instance, use an `onAbort` message to warn users if they stop loading some essential image, such as an image map (see [Section 6.5](#)):

```

```

The `onError` attribute is invoked if some error occurs during the loading of the image, but not for a missing image or one that the user chose to stop loading. Presumably, the applet could attempt to recover from the error or load a different image in its place.

The currently popular browsers execute the JavaScript code associated with the `<img>` tag's `onLoad` attribute right after the browser successfully loads and displays the image.

See [Section 13.3.3](#) for more information about JavaScript and event handlers.

#### 5.2.6.17 Combining `<img>` attributes

You may combine any of the various standard and extension attributes for images where and when they make sense. The order for inclusion of multiple attributes in the `<img>` tag is not important, either. Just be careful not to use redundant attributes, or you won't be able to predict the outcome.

### 5.2.7 Video Extensions

The special `controls`, `dynsrc`, `loop`, and `start` attribute extensions for the `<img>` tag are unique to Internet Explorer and are not HTML 4 or XHTML standard attributes. They let you embed an inline movie into the body content, just like an image.

Equivalent behavior is available in Netscape via an extension program known as a *plug-in*. Plug-ins place an additional burden on the user, in that each user must find and install the appropriate plug-in before being

able to view the inline video. The Internet Explorer `<img>` tag extensions, on the other hand, make video display an intrinsic part of the browser. [\[Section 12.2\]](#)

However, the Internet Explorer movie extensions currently are very limited. They are not supported by any other browser and can be used only with Audio Video Interleave (AVI)-formatted movie files, since that's the player format built into Internet Explorer and enabled through Microsoft Windows operating systems. Moreover, recent innovations in browser technology, objects, and applets in particular may make Internet Explorer's approach of extending the already overloaded `<img>` tag obsolete.

#### 5.2.7.1 The `dynsrc` attribute

Use the `dynsrc` attribute extension in the `<img>` tag to reference an AVI movie for inline display by Internet Explorer. Its required value is the URL of the movie file, enclosed in quotation marks. For example, this text displays the tag and attribute for an AVI movie file entitled *intro.avi*:

```

```

The browser sets aside a video viewport in the HTML display window and plays the movie, with audio if it's included in the clip and if your computer is able to play audio. Internet Explorer treats `dynsrc` movies similar to inline images: in line with current body content and according to the dimension of the video frame. And, like common images, the `dynsrc`-referenced movie file gets displayed immediately after download from the server. You may change those defaults and add some user controls with other attributes, as described later.

Because all other browsers currently ignore the special Internet Explorer attributes for movies, they may become confused by an `<img>` tag that does not contain the otherwise required `src` attribute and an image URL. We recommend that you include the `src` attribute and a valid image file URL in all `<img>` tags, including those that reference a movie for Internet

Explorer users. The other browsers display the still image in place of the movie; Internet Explorer does the reverse and plays the movie, but does not display the image. Note that the order of attributes does not matter. For example:

```

```

Internet Explorer loads and plays the AVI movie *intro.avi*; other graphical browsers will load and display the *mvstill.gif* image instead.

### 5.2.7.2 The controls attribute

Normally, Internet Explorer plays a movie inside a framed viewport once, without any visible user controls. Although no longer supported in Internet Explorer Version 5 or later, with older versions of the browser the user may restart, stop, and continue the movie by clicking inside that viewport with the mouse. Use the `controls` attribute (no value) to add visible controls to the movie viewport so that the user may, with the mouse, play, fast-forward, reverse, stop, and pause the movie, like on a VCR. If the movie clip includes a soundtrack, Internet Explorer provides an audio volume control as well. For example:

```
` tag lets you have the clip play repeatedly for an integer number of times set by the attribute's value, or forever if the value is `infinite`. The user may still cut the loop short by clicking on the movie image, by pressing the stop button if given controls (see [Section 5.2.7.2](#)), or by moving on to another document.

The following *intro.avi* movie clip will play from beginning to end, then restart at the beginning and play through to the end nine more times:

```
<img dynsrc="movies/intro.avi" loop=10 src="pics/mvsti
```

Whereas the following movie will play over and over again, incessantly:

```
<img dynsrc="movies/intro.avi" loop=infinite src="pics
```

Looping movies aren't necessarily meant to annoy. Some special-effects animations, for instance, are a sequence of repeated frames or segments. Rather than stringing the redundant segments into one long movie, which extends its download time, simply loop the single, compact segment.



#### 5.2.7.4 The start attribute

Normally, an Internet Explorer movie clip starts playing as soon as it's downloaded. You can modify that behavior with the `start` attribute in the movie's `<img>` tag. By setting its value to `mouseover`, you delay playback until the user passes the mouse pointer over the movie viewport. The other valid `start` attribute value, `fileopen`, is the default: start playback just after download. It is included because both values may be combined in the `start` attribute, to cause the movie to play back automatically once after download and then whenever the user passes the mouse over its viewport. When combining the `start` attribute values, add a value-separating comma, with no intervening spaces, or else enclose them in quotes.

For example, our by-now-infamous *intro.avi* movie will play once when its host HTML document is loaded by the user and again whenever he passes the mouse over the movie's viewport:

```

```

#### 5.2.7.5 Combining movie <img> attributes

Treat Internet Explorer inline movies as you would any image, mixing and matching the various movie-specific as well as the standard and extended `<img>` tag attributes and values supported by the browser. For example, you might align the movie (or its image alternative, if displayed by another browser) to the right of the browser window:

```

```

Combining attributes to achieve a special effect is good. We also recommend that you combine attributes to give control to the user, when appropriate.

As we stated in [Section 5.2.7.4](#), by combining attributes you can also delay playback until the user passes the mouse over its viewport.

Magically, the movie comes alive and plays continuously:

```
` tag.<sup>[2]</sup> Once named, the frame may become the destination display window for a hypertext-linked document selected within a document displayed in some other frame. You accomplish this redirection by adding the special `target` attribute to the anchor that references the document.

[2] The `id` attribute provides the same unique labeling but cannot be used for frame content redirection.

### 11.7.1 The `target` Attribute for the `<a>` Tag

If you include a `target` attribute within an `<a>` tag, the browser loads and displays the document named in the tag's `href` attribute in a frame or window whose `name` matches the target. If the named frame or window doesn't exist, the browser opens a new window, gives it the specified label, and loads the new document into that window. Once this process has been completed, hypertext-linked documents can target the new window.

Targeted hypertext links make it easy to create effective navigational tools. A simple table of contents document, for example, might redirect documents into a separate window:

```
<h3>Table of Contents</h3>
```

```
<ul>
```

```
<li><a href="pref.html" target="view_window">Preface
```

```
<li><a href="chap1.html" target="view_window">Chapte
```

```
<li><a href="chap2.html" target="view_window">Chapte
```

```
<li><a href="chap3.html" target="view_window">Chapte
</ul>
```

The first time the user selects one of the table of contents hypertext links, the browser opens a new window, labels it "view\_window," and displays the desired document's contents inside it. If the user selects another link from the table of contents and the "view\_window" is still open, the browser again loads the selected document into that window, replacing the previous document.

Throughout the whole process, the window containing the table of contents is accessible to the user. By clicking on a link in one window, the user causes the contents of the other window to change.

Rather than opening an entirely new browser window, a more common use of `target` is to direct hyperlink contents to one or more frames in a `<frameset>` display or to an inline `<iframe>` window. You might place the table of contents into one frame of a two-frame document and use the adjacent frame for display of the selected documents:

```
<frameset cols="150,*">

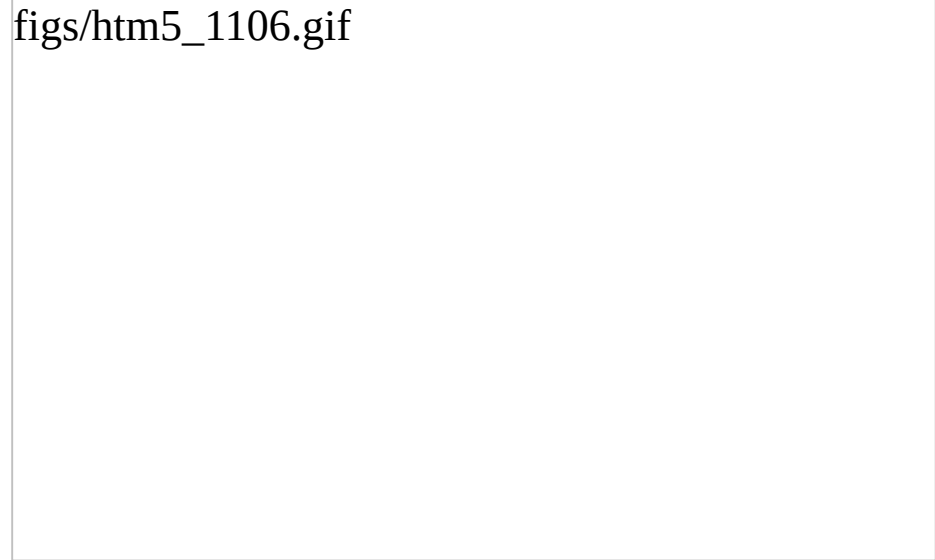
  <frame src="toc.html">

  <frame src="pref.html" name="view_frame">

</frameset>
```

When the browser initially displays the two frames, the left frame contains the table of contents, and the right frame contains the Preface (see [Figure 11-6](#)).


**Figure 11-6. Table of contents frame controls content of adjacent frame**



figs/htm5\_1106.gif

When a user selects a link from the table of contents in the left frame (for example, Chapter 1), the browser loads and displays the associated document into the "view\_frame" frame on the right side ([Figure 11-7](#)). As other links are selected, the right frame's contents change, while the left frame continuously makes the table of contents available to the user.

**Figure 11-7. The contents of Chapter 1 are displayed in the adjacent frame**



figs/htm5\_1107.gif

## 11.7.2 Special Targets

There are four reserved target names for special document-redirection actions:

`_blank`

The browser always loads a `target="_blank"` linked document into a newly opened, unnamed window.

`_self`

This target value is the default for all `<a>` tags that do not specify a target, causing the target document to be loaded and displayed in the same frame or window as the source document. This target is redundant and unnecessary unless used in combination with the `target` attribute in the `<base>` tag in a document's head (see [Section 11.7.3](#)).

`_parent`

This target causes the document to be loaded into the parent window or frameset containing the frame containing the hypertext reference. If the reference is in a window or top-level frame, then it is equivalent to the target `_self`.

A brief example may help clarify how this link works. Consider a link in a frame that is part of a three-column frameset. This frameset, in turn, is a row in the top-level frameset being displayed in the browser window. This arrangement is shown in [Figure 11-8](#).

If no target is specified for the hypertext link, it is loaded into the containing frame. If a target of `_parent` is specified, the document is loaded into the area occupied by the three-column frameset containing the frame that contains the link.


`_top`

This target causes the document to be loaded into the window containing the hypertext link, replacing any frames currently

displayed in the window.

Continuing with the frame hierarchy, as shown in [Figure 11-8](#), using a target of `_top` would remove all the contained frames and load the document into the entire browser window.

**Figure 11-8. Using special hypertext targets in nested frames and framesets**



figs/htm5\_1108.gif

All four of these `target` values begin with the underscore character. Any other window or target beginning with an underscore is ignored by the browser, so don't use the underscore as the first character of any frame `name` or `id` you define in your documents.

### 11.7.3 The `<base>` Default Target

It can be tedious to specify a target for every hypertext link in your documents, especially when most are targeted at the same window or frame. To alleviate this problem, you can add a `target` attribute to the

`<base>` tag. [[<base>](#)]

The `target` attribute in the `<base>` tag sets the default target for every hypertext link in the current document that does not contain an explicit `target` attribute. For example, in our example table of contents document, almost every link causes the document to be displayed in another window named "view\_frame." Rather than including that target in each hypertext link, you should place the common target in the table of contents's `<base>` tag within its `<head>`:

```
<head>

<title>Table of Contents</title>

<base target="view_frame">

</head>

<body>

<h3>Table of Contents</h3>

<ul>

  <li><a href="pref.html">Preface</a></li>

  <li><a href="chap1.html">Chapter 1</a></li>

  <li><a href="chap2.html" >Chapter 2</a></li>

  <li><a href="chap3.html">Chapter 3</a></li>

</ul>

</body>
```

Notice that we don't include any other target references in the list of



hyperlinks, because the browser loads and displays all the respective documents in the base target "view\_frame."

### 11.7.4 Traditional Link Behavior

Before the onset of frames, each time you selected a hyperlink, the corresponding document replaced the contents of the browser window. With frames, this behavior is modified so that the corresponding document replaces the content of the referencing frame. This is often not the desired behavior, and it can be disconcerting to people browsing your documents.

For example, suppose you have arranged all of the documents on your site to present themselves in three frames: a navigational frame at the top of the browser window, a scrolling content frame in the middle, and a feedback form at the bottom. You named the content frame with the `name` attribute of the `<frame>` tag in the top-level document for your collection and used the `target` attribute of the `<base>` tag in every document on your site to ensure that all links are loaded into the center content frame.

This arrangement works perfectly for all the documents on your site, but what happens when a user selects a link that takes him to a different site? The referenced document is still loaded into the center content frame. Now the user is confronted by a document from some other site, surrounded by your navigation and feedback frames!<sup>[3]</sup> Very impolite.

<sup>[3]</sup> Check out [Chapter 17](#) for how to step out into the forefront when your pages happen to be on the other end of a targetless hyperlink.

The solution is to make sure that every hypertext link that references a remote document has a target of `_top`. This way, when the user selects a link that takes him away from your site, the remote document replaces the contents of the entire browser window, including your navigation and feedback frames. If the majority of the links in your documents are to other sites, you might consider adding `target="_top"` to a `<base>` tag in your document and using explicit `target` attributes in the links to your local documents.

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R]  
[S] [T] [U] [V] [W] [X] [Z]



## **HTML & XHTML: The Definitive Guide, 5th Edition**

By [Bill Kennedy](#), [Chuck Musciano](#)

Publisher : O'Reilly

Pub Date : August 2002

ISBN : 0-596-00382-X

Pages : 670

Slots : 1

- [Table of Contents](#)
- [Index](#)
- [Reviews](#)
- [Examples](#)
- [Reader Reviews](#)
- [Errata](#)

*HTML & XHTML: The Definitive Guide*, 5th Edition is the most comprehensive, up-to-date book available on HTML and XHTML. The authors cover every element of HTML/XHTML in detail, explaining how each element works and how it interacts with other elements. With hundreds of examples, the book gives you models for writing your own effective web pages and for mastering advanced features like style sheets and frames.

## Chapter 15. XML

HTML is a maverick. It follows the rules of formal electronic document-markup design and implementation only loosely. HTML was born out of the need to assemble text, graphics, and other digital content into electronic documents that could be sent over the global Internet. In the early days of the Web's boom, the demand for better browsers and document servers driven by hordes of new users with insatiable appetites for more and cooler web pages left little time for worrying about things like standards and practices.

Of course, without guiding standards, HTML would eventually have devolved into Babel. That almost happened during the browser wars in the mid- to late-1990s. Chaos is not an acceptable foundation for an industry whose value is already measured in the trillions of dollars. Although the standards people at the W3C managed to rein in the maverick HTML with standard Version 4, it is still too wild for the royal herd of markup languages.

The HTML 4.01 standard is defined using the Standardized Generalized Markup Language (SGML). While more than adequate for formalizing HTML, SGML is far too complex to use as a general tool for extending and enhancing HTML. Instead, the W3C has devised a standard known as the Extensible Markup Language, or XML. Based on the simpler features of SGML, XML is kinder, gentler, and more flexible, well suited to guiding the birth and orderly development of new markup languages. With XML, HTML itself is being reborn as XHTML.

In this chapter, we cover the basics of XML, including how to read it, how to create simple XML Document Type Definitions (DTDs), and the ways you might use XML to enhance your use of the Internet. In the next chapter, we explore the depths of XHTML.

You don't have to understand all about XML to write XHTML. We think it's helpful, but if you want to cut to the chase, feel free to skip to the next chapter. However, you may want to take a look at some of the uses of

XML covered at the end of this chapter, starting in [Section 15.8](#).

This chapter provides only an overview of XML. Our goal is to whet your appetite and make you conversant in XML. It is only an overview. For full fluency, consult *Learning XML* by Erik T. Ray or *XML in a Nutshell* by W. Scott Means and Elliotte Rusty Harold, both from O'Reilly.

## Chapter 11. Frames

Beginning with Netscape 2.0, HTML authors have been able to divide the browser's main display window into independent window *frames*, each simultaneously displaying a different document something like a wall of monitors in a TV control room. Instantly popular, frames were adopted (and extended) by Microsoft for Internet Explorer and are standard features for HTML 4 and XHTML.

## 10.2 Basic Table Tags

A wide variety of tables can be created with only five tags: the `<table>` tag, which encapsulates a table and its elements in the document's body content; the `<tr>` tag, which defines a table row; the `<th>` and `<td>` tags, which define the table's headers and data cells; and the `<caption>` tag, which defines a title or caption for the table. Beyond these core tags, you may also define and control whole sections of tables, including adding running headers and footers, with the `<colgroup>`, `<col>`, `<tbody>`, `<thead>`, and `<tfoot>` tags. Each tag has one or more required and optional attributes, some of which affect not only the tag itself but also related tags.

### 10.2.1 The `<table>` Tag

The `<table>` tag and its `</table>` end tag define and encapsulate a table within the body of your document. Unless otherwise placed within the browser window by style sheet, paragraph, division-level, or other alignment options, the browser stops the current text flow, breaks the line, inserts the table beginning on a new line, and then restarts the text flow on a new line below the table.

## **<table>**

### **Function:**

Defines a table

### **Attributes:**

`align`, `background` (`[]`, `[]`), `bgcolor`, `border`, `bordercolor` (`[]`, `[]`), `bordercolordark` (`[]`), `bordercolorlight` (`[]`), `cellpadding`, `cellspacing`, `class`, `cols` (`[]`, `[]`), `dir`, `frame`, `height` (`[]`, `[]`), `hspace` (`[]`), `id`, `lang`, `nowrap` (`[]`), `onClick`, `onDblClick`, `onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`, `rules`, `style`, `summary`, `title`, `valign` (`[]`), `vspace` (`[]`), `width`

### **End tag:**

`</table>`; never omitted

### **Contains:**

*table\_content*

### **Used in:**

*block*

The only content allowed within the `<table>` is one or more `<tr>` tags, which define each row of table contents, along with the various table sectioning tags: `<thead>`, `<tfoot>`, `<tbody>`, `<col>`, and `<colgroup>`.

### **10.2.1.1 The align attribute (deprecated)**

The HTML 4 and XHTML standards have deprecated this attribute in



favor of the `align` property provided by Cascading Style Sheets, yet it remains popular and is currently well supported by the popular browsers.

Like images, tables are rectangular objects that float in the browser display, aligned according to the current text flow. Normally, the browser left-justifies a table, abutting its left edge to the left margin of the display window. Or the table may be centered if under the influence of the `<center>` tag, a centered paragraph, or a centered division. Unlike images, however, tables are not inline objects. Text content normally flows above and below a table, not beside it. You can change that display behavior with the `align` attribute or a cascading style definition for the `<table>` tag.

The `align` attribute accepts a value of either `left`, `right`, or `center`, indicating that the table should be placed flush against the left or right margin of the text flow, with the text flowing around the table, or in the middle with text flowing above and below.

Note that the `align` attribute within the `<table>` tag is different from those used within a table's element tags, `<tr>`, `<td>`, and `<th>`. In those tags, the attribute controls text alignment within the table's cells, not alignment of the table within the containing body-text flow.

### 10.2.1.2 The `bgcolor` and `background` attributes

You can make the background of a table a different color than the document's background with the `bgcolor` attribute for the `<table>` tag. The color value for the `bgcolor` attribute must be set to either an RGB color value or a standard color name. Both the syntax of color values and the acceptable color names are provided in [Appendix G](#).

The popular browsers give every cell in the table (but not the caption) this background color. You may also set individual row and cell colors by providing the `bgcolor` attribute or a style attribute for those rows or cells.

The `background` attribute, a nonstandard extension supported by the popular browsers, supplies the URL of an image that is tiled to fill the background of the table. The image is clipped if the table is smaller than the image. By using this attribute with a borderless table, you can put text over an image contained within a document.

### 10.2.1.3 The border attribute

The optional `border` attribute for the `<table>` tag tells the browser to draw lines around the table and the rows and cells within it. The default is no borders at all. You may specify a value for `border`, but you don't have to with HTML. Alone, the attribute simply enables borders and a set of default characteristics different for each of the popular browsers (reexamine [Figure 10-1](#); it has borders). With XHTML, use `border="border"` to achieve the same default results. Otherwise, in HTML or with XHTML, supply an integer value for `border` equal to the pixel width of the 3D chiseled-edge lines that surround the outside of the table and make it appear to be embossed onto the page.

### 10.2.1.4 The frame and rules attributes

With Netscape 4, the `border` attribute is all or nothing, affecting the appearance and spacing both of the frame around the table and of the rule lines between data cells. Internet Explorer Versions 4 and later and Netscape 6 let you individually modify the various line segments that make up the borders around the table (`frame`) as well as around the data cells (`rules`).

The standard `frame` attribute modifies `border`'s effects for the lines that surround the table. The default value what you get if you don't use `frame` at all is `box`, which tells the browser to draw all four lines around the table. The value `border` does the same thing as `box`. The value `void` removes all four of the `frame` segments. The `frame` values `above`, `below`, `lhs`, and `rhs` draw the various border segments on the top, bottom, left, and right side, respectively, of the table. The value

`hsides` draws borders on the top and bottom (horizontal) sides of the table; `vsides` draws borders on the left and right (vertical) sides of the table.

With standard tables (supported in Internet Explorer 4 and later and in Netscape 6), you also may control the thickness of a table's internal cell borders via the `rules` attribute. The default behavior, represented by the value of `all`, is to draw borders around all cells. Specifying `groups` places thicker borders between row and column groups defined by the `<thead>`, `<tbody>`, `<tfoot>`, `<col>`, and `<colgroup>` tags. Using `rows` or `cols` places borders only between every row or column, respectively, while using `none` removes borders from every cell in the table.

### **10.2.1.5 The `bordercolor`, `bordercolorlight`, and `bordercolordark` attributes**

The popular browsers normally draw a table border in three colors, using light and dark variations on the document's background color to achieve a 3D effect. The nonstandard `bordercolor` attribute lets you set the color of the table borders and rules to something other than the background (if borders are enabled, of course). The `bordercolor` attribute's value can be either an RGB hexadecimal color value or a standard color name, both of which are described fully in [Appendix G](#).

Internet Explorer also lets you set the border edge colors individually with special extension attributes: the `bordercolorlight` and `bordercolordark` colors shade the lighter and darker edges of the border.

The effectiveness of the 3D beveled-border effect is tied to the relationship between these two colors. In general, the light color should be about 25% brighter than the border color, and the dark color should be about 25% darker.

### 10.2.1.6 The cellpadding attribute

The `cellspacing` attribute controls the amount of space placed between adjacent cells in a table and along the outer edges of cells along the edges of a table.

Browsers normally put 2 pixels of space between cells and along the outer edges of the table. If you include a `border` attribute in the `<table>` tag, the cell spacing between interior cells grows by 2 more pixels (4 total) to make space for the chiseled edge on the interior border. The outer edges of edge cells grow by the value of the `border` attribute.

By including the `cellspacing` attribute, you can widen or reduce the interior cell borders. For instance, to make the thinnest possible interior cell borders, include the `border` and `cellspacing=0` attributes in the table's tag.

### 10.2.1.7 The cellpadding attribute

The `cellpadding` attribute controls the amount of space between the edge of a cell and its contents, which by default is 1 pixel. You may make all the cell contents in a table touch their respective cell borders by including `cellpadding=0` in the table tag. You may also increase the `cellpadding` space by making its value greater than 1.

### 10.2.1.8 Combining the border, cellpadding, and cellspacing attributes

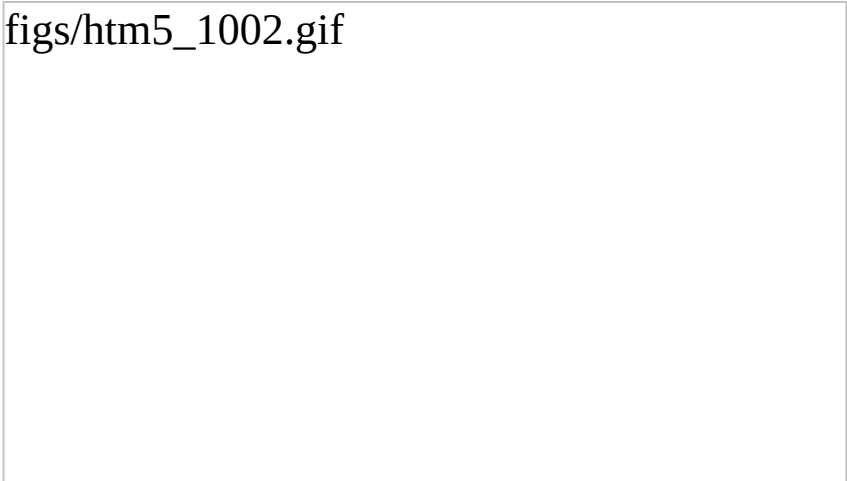
The interactions between the `border`, `cellpadding`, and `cellspacing` attributes of the `<table>` tag combine in ways that can be confusing. [Figure 10-2](#) summarizes how the attributes create interior and exterior borders of various widths.

While all sorts of combinations of the `border` and `cellspacing` attributes are possible, these are the most common:

- `border=1` and `cellspacing=0` produces the narrowest possible interior and exterior borders: 2 pixels wide.
- `border=n` and `cellspacing=0` makes the narrowest possible interior borders (2 pixels wide), with an external border that is  $n + 1$  pixels wide.
- `border=1` and `cellspacing=n` tables have equal-width exterior and interior borders, all with chiseled edges just 1 pixel wide. All borders will be  $n + 2$  pixels wide.

**Figure 10-2. The border, cellspacing, and cellpadding attributes of a table**

figs/htm5\_1002.gif

The figure is a placeholder for a visual representation of a table with various border and spacing attributes. It is represented by a large, empty rectangular box with a thin black border.

### 10.2.1.9 The cols attribute

To format a table, the browser must first read the entire table contents, determining the number and width of each column in the table. This can be a lengthy process for long tables, forcing users to wait to see your pages. The nonstandard `cols` attribute tells the browser, in advance, how many columns to expect in the table. The value of this attribute is an integer value defining the number of columns in the table.

The `cols` attribute only advises the browser. If you define a different

number of columns, the browser is free to ignore the `cols` attribute in order to render the table correctly. In general, it is good form to include this attribute with your `<table>` tag, if only to help the browser do a faster job of formatting your tables.

#### 10.2.1.10 The `valign` and `nowrap` attributes

The `valign` attribute sets the default vertical alignment of data in cells for the entire table. Acceptable values for the `valign` attribute in `<table>` are `top`, `bottom`, `middle`, or `baseline`; the default vertical position is the center of the cell.

Browsers treat each table cell as though it's a browser window unto itself, flowing contents inside the cell as they would common body contents (although they are subject to special table-cell alignment properties). Accordingly, the browsers automatically wrap text lines to fill the allotted table cell space. The `nowrap` attribute, when included in the `<table>` tag, stops that normal word wrapping in all rows in the table. With `nowrap`, the browser assembles the contents of the cell onto a single line, unless you insert a `<br>` or `<p>` tag, which then forces a break so that the contents continue on a new line inside the table cell.

The `valign` and `nowrap` attributes for the `<table>` tag currently are supported by Internet Explorer Version 5 but not Version 6. You achieve similar effects in Netscape and Internet Explorer 6 by including a `valign` or `nowrap` attribute within the individual `<tr>`, `<td>`, and `<th>` tags.

#### 10.2.1.11 The `width` and `height` attributes

Browsers automatically make a table only as wide as needed to correctly display all of the cell contents. If necessary, you can make a table wider with the `width` attribute.

The value of the `width` attribute is either an integer number of pixels or a relative percentage of the screen width, including values greater than

100%. For example:

```
<table width=400>
```

tells the extended browser to make the table 400 pixels wide, including any borders and cell spacing that extend into the outer edge of the table. If the table is wider than 400 pixels, the browser ignores the attribute.

Alternatively:

```
<table width="50%">
```

tells the browser to make the table half as wide as the display window. Again, this width includes any borders or cell spacing that extend into the outer edge of the table and has no effect if the table normally is more than half the user's current screen width.

Use relative widths for tables you want to resize automatically to the user's window; for instance, tables you always want to extend across the entire window (`<table width="100%">`). Use an absolute width value for carefully formatted tables whose contents become hard to read in wide display windows.

For Netscape and Internet Explorer, you can use the nonstandard `height` attribute to suggest a recommended height for the table. The browser makes the table no shorter than this height but may make the table taller if needed to contain the table's contents. This attribute is useful when trying to stretch tables to fit in a frame or some specific area of a document but is of little use otherwise, particularly since it is not a standard attribute.

### 10.2.1.12 The **summary** attribute

The `summary` attribute was introduced to HTML in the 4.0 standard. Its value is a quote-enclosed string that describes the purpose and summarizes the contents of the table. Its intended use, according to the standard, is to provide extended access to nonvisual browsers, particularly for users with disabilities.

### 10.2.1.13 The `hspace` and `vspace` attributes

As with images, Netscape, but not Internet Explorer, lets you give your table some extra space within the body text of your document. Use the nonstandard `hspace` and `vspace` attributes in the `<table>` tag, each with a value equal to the number of pixels of space to offset the table from the left and right or top and bottom, respectively, of the enclosing text.

## 10.2.2 Common Table Attributes

The HTML and XHTML standards, combined with the CSS standard, provide a number of attributes common not only to the `<table>` tag and the other table-creation tags but to most other tags as well. Except for the CSS-related attributes `class` and `style` for controlling the table display, none of the standard attributes are yet fully supported by any of the popular browsers.

### 10.2.2.1 The `id` and `title` attributes

Use the `id` attribute with a quote-enclosed string value to uniquely label a `<table>` tag for later reference by a hyperlink or an applet. Use the `title` attribute with a string value to optionally entitle the table or any of its segments for general reference. A `title`'s value need not be unique, and it may or may not be used by the browser. Internet Explorer, for example, displays the `title` attribute's text value whenever the user passes the mouse pointer over the element's contents. [[Section 4.1.1.4](#)] [[Section 4.1.1.4](#)]

### 10.2.2.2 The `dir` and `lang` attributes

Although its contents are predominantly in English, the Web is worldwide. The HTML 4 and XHTML standards take pains to extend the language to all cultures. We support that effort wholeheartedly. The `dir` and `lang`



attributes are just small parts of that process.

The `dir` attribute advises the browser which direction the text of the contents should flow in, from left to right (`dir=ltr`), as for common Western languages like English and German, or right to left (`dir=rtl`), as for common Eastern language like Hebrew and Chinese.

The `lang` attribute lets you explicitly indicate the language used in the table or even individual cell contents. Its value should be an ISO standard two-letter primary code followed by an optional dialect subcode, with a hyphen (–) between the two.

Currently, `dir` and `lang` are supported by Internet Explorer Versions 5 and later and by Netscape 6. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)]

### 10.2.2.3 The class and style attributes

The CSS standard is the sanctioned way to define display attributes for HTML/XHTML elements, and it is rapidly becoming the only way. Use the `style` attribute to define display characteristics for the table and its elements that take immediate effect and override the display styles that may be currently in effect for the whole document. Use the `class` attribute to reference a style sheet that defines the unique display characteristics for the table and its elements.

We discuss the `class` and `style` attributes and the CSS standard in detail in [Chapter 8](#). [[Section 8.1.1](#)] [[Section 8.3](#)]

### 10.2.2.4 The event attributes

The popular browsers have internal mechanisms that detect the various user-initiated mouse and keyboard events that can happen in and around your tables and their elements. For instance, the user might click the mouse pointer in one of the table cells or highlight the caption and then press the Enter key.

With the various event attributes, such as `onClick` and `onKeyDown`, you can react to these events by having the browser execute one or more JavaScript commands or applets that you reference as the value to the respective event attribute. See [Chapter 12](#) for details.

### 10.2.3 The `<tr>` Tag

Make a new row in a table with the `<tr>` tag. Place within the `<tr>` tag one or more cells containing headers, defined with the `<th>` tag, or data, defined with the `<td>` tag (see [Section 10.2.4](#)). The `<tr>` tag accepts a number of special attributes that control its behavior, along with the common table attributes described in [Section 10.2.2](#).

## <tr>

### *Function:*

Defines a row within a table

### *Attributes:*

`align`, `background` (☐) , `bgcolor`, `bordercolor` (☐) ,  
`bordercolordark` (☐) , `bordercolorlight` (☐) , `char`,  
`charoff`, `class`, `dir`, `id`, `lang`, `nowrap` (☐, ☐) , `onClick`,  
`onDblClick`, `onKeyDown`, `onKeyPress`, `onKeyUp`,  
`onMouseDown`, `onMouseMove`, `onMouseOut`, `onMouseOver`,  
`onMouseUp`, `style`, `title`, `valign`

### *End tag:*

`</tr>`; may be omitted in HTML

### *Contains:*

*tr\_content*

### *Used in:*

*table\_content*

Every row in a table has the same number of cells as the longest row; the browser automatically creates empty cells to pad rows with fewer defined cells.

### 10.2.3.1 The align and valign attributes

The `align` attribute for the `<table>` tag may be deprecated in the HTML and XHTML standards, but it is alive and kicking for `<tr>` and other table elements. The `align` attribute for the `<tr>` tag lets you change the default horizontal alignment of all the contents of the cells in a

row. The attribute affects all the cells within the current row, but not subsequent rows.

An `align` attribute value of `left`, `right`, `center`, `justify`, or `char` causes the browser to align the contents of each cell in the row against the left or right edge, in the center of the cell, spread across the cell, or to a specified character in the cell, respectively.

Similarly, you can change the default vertical alignment for the contents of data cells contained within a table row with the `valign` attribute. Normally, browsers render cell contents centered vertically. By including the `valign` attribute in the `<tr>` tag with a value of `top`, `bottom`, `center`, `middle`, or `baseline` (Internet Explorer only), you tell the browser to place the table row's contents flush against the top or bottom of their cells, centered, or aligned to the baseline of the top line of text in other cells in the row, respectively (see [Figure 10-3](#)). Netscape ignores the `baseline` value.

```
<table border="border">

  <tr>

    <th>Alignment</th>

    <th>Top</th>

    <th>Baseline</th>

    <th>Center</th>

    <th>Middle<></th>

    <th>Bottom</th>

  </tr>

  <tr align="center">
```

```
<th><h1>Baseline_ _<br />Another line</h1></th>

<td valign="top">AAyy</td>

<td valign="baseline">_AAyy_</td>

<td valign="center">AAyy</td>

<td valign="middle">AAyy</td>


<td valign="bottom">AAyy</td>

</tr>

</table>
```

**Figure 10-3. Effects of the valign attribute on Internet Explorer's (top) and Netscape's (bottom) table cell-content alignment**

figs/htm5\_1003.gif



You also can specify the horizontal and vertical alignments for individual cells within a row (see [Section 10.2.4.1](#)). Use the alignment attributes in the `<tr>` tag to specify the most common cell-content justifications for the row (if not the default), and use a different `align` or `valign` attribute for those individual cells that deviate from the common alignment.

**Table 10-1** contains the horizontal (`align`) and vertical (`valign`) table cell-content attribute values and options. Values in parentheses are the defaults for the popular browsers.

**Table 10-1. Table cell-content alignment attribute values and options**

	Netscape and Internet Explorer	
Attribute	Headers (<th>)	Data (<td>)
<code>align</code>	Left	(Left)
	(Center)	Center
	Right	Right
	Justify	Justify
	Char <sup>[1]</sup>	Char <sup>[1]</sup>
<code>valign</code>	Top	Top
	(Center)	(Center)
	(Middle)	(Middle)

	Bottom	Bottom
	Baseline	Baseline

[1] Value not yet supported.

### 10.2.3.2 The `char` and `charoff` attributes

Even simple word processors let you line up decimal points for numbers in a table. Until the advent of the HTML 4.0 standard, however, the language was deficient in this feature. Now you may include the `char` attribute to indicate which letter in each of the table row's cells should be the axis for that alignment. You need not include a value with `char`. If you don't, the default character is language-based: it's a period in English, for example, and a comma in French. Include the `char` attribute and a single letter as its value to specify a different alignment character.

Use the `charoff` attribute and an integer value to specify the offset to the first occurrence of the alignment character on each line. If a line doesn't include the alignment character, it should be horizontally shifted to end at the alignment position.

The `char` and `charoff` attributes are defined in HTML 4 and XHTML but are not yet supported by any of the popular browsers.

### 10.2.3.3 The `bgcolor` and `background` attributes

Like its relative for the `<table>` tag, the `bgcolor` attribute for the `<tr>` tag sets the background color of the entire row. Its value is either an RGB color value or a standard color name. Both the syntax of color values and the acceptable color names are provided in [Appendix G](#).

Every cell in the row is given this background color. Individual cell colors can be changed by providing the `bgcolor` attribute for those cells.

Unique to Netscape for the `<tr>` tag, the nonstandard `background` attribute with its image-file URL value places a graphic tiled into and behind the text of the entire table row. For example, this tag fills the table row with bricks:

```
<tr background="bricks.gif">
```

#### 10.2.3.4 The `bordercolor`, `bordercolorlight`, and `bordercolordark` attributes

Like their nonstandard brethren for the `<table>` tag, Internet Explorer lets you use these attributes to set the color of the borders within the current row.

Their values override any values set by the corresponding attributes in the containing `<table>` tag. See the corresponding descriptions of these extensions in [Section 10.2.1.5](#) for details. Color values can be either RGB color values or standard color names, both of which are described fully in [Appendix G](#).

#### 10.2.3.5 The `nowrap` attribute

Browsers treat each table cell as though it were a browser window unto itself, flowing contents inside the cell as they would common body contents (although subject to special table cell-alignment properties). Accordingly, the browsers automatically wrap text lines to fill the allotted table cell space. The `nowrap` attribute, when included in a table row, stops that normal word wrapping in all cells in that row. With `nowrap`, the browser assembles the contents of the cell onto a single line, unless you insert a `<br>` or `<p>` tag, which forces a break so that the contents continue on a new line inside the table cell.

### 10.2.4 The `<th>` and `<td>` Tags

The `<th>` and `<td>` tags go inside the `<tr>` tags of a table to create the



header and data cells, respectively, and to define the cell contents within the rows. The tags operate similarly; the only real differences are that the browsers render header text meant to entitle or otherwise describe table data in boldface font style and that the default alignment of their respective contents may be different than for data. Data typically gets left-justified by default; headers get centered (see [Table 10-1](#)).

## <th> and <td>

### Function:

Define table data and header cells

### Attributes:

abbr, align, background (□, □), bgcolor, bordercolor (□), bordercolordark (□), bordercolorlight (□), char, charoff, class, colspan, dir, headers, height, id, lang, nowrap, onClick, onDbClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, rowspan, scope, style, title, valign, width

### End tag:

</th> or </td>; may be omitted in HTML

### Contains:

*body\_content*

### Used in:

*tr\_content*

Like those available for the table row (<tr>) tag, the table cell tags support a rich set of style and content-alignment attributes that you may apply to a single data or header cell. These attributes override the default values for the current row. There are special attributes that control the number of columns or rows a cell may span in the table. The <th> and <td> tags also accept the common table attributes described in [Section 10.2.2](#).

The contents of the <th> and <td> tags can be anything you might put in the body of a document, including text, images, forms, and so on even

another table. And, as described earlier, the browser automatically creates a table large enough, both vertically and horizontally, to display all the contents of any and all the cells.

If a particular row has fewer header or data items than other rows, the browser adds empty cells at the end to fill the row. If you need to make an empty cell before the end of a row, for instance to indicate a missing data point, create a header or data cell with no content.

Empty cells look different than those containing data or headers if the table has borders: the empty cell does not appear embossed onto the window but instead is simply left blank. If you want to create an empty cell that has incised borders like all the other cells in your table, be sure to place a minimal amount of content in the cell: a single `<br>` tag, for instance.

#### 10.2.4.1 The `align` and `valign` attributes

The `align` and `valign` attributes are identical to those of the same name for the table row tag (`<tr>`; see [Section 10.2.3](#)), except that when used with a `<th>` or `<td>` tag, they control the horizontal or vertical alignment of content in just the current cell. Their value overrides any alignment established by the respective `align` or `valign` attribute of the `<tr>` tag but does not affect the alignment of subsequent cells. See [Table 10-1](#) for alignment details.

You may set the `align` attribute's value to `left`, `right`, or `center`, causing the browsers to align the cell contents against the left or right edge or in the center of the cell, respectively.

In addition, Internet Explorer Version 5, but not Version 6, supports a value of `justify` so that the words spread out to fill the cell, as in a newspaper column. Internet Explorer 6 centers the `align=justify` text. Netscape 6 simply ignores the `justify` attribute value and places the text against the left margin of the table cell.

The `valign` attribute may have a value of `top` (default), `bottom`, `center`, `middle`, or `baseline`, telling the browser to align the cell's contents to the top or bottom edge, in the center or middle of the cell, or (Internet Explorer 6, but not Netscape 6) to the baseline of the first line of text in other cells in the row.

#### 10.2.4.2 The width attribute

Like its twin in the `<table>` tag that lets you widen a table, the `width` attribute for table cell tags lets you widen an individual cell and hence the entire column it occupies. You set the `width` to an integer number of pixels or a percentage indicating the cell's width as a fraction of the table as a whole.

For example:

```
<th width=400>
```

sets the current header cell's width, and hence the entire column of cells, to 400 pixels wide. Alternatively:

```
<td width="40%">
```

creates a data cell with a column occupying 40% of the entire table's width.

Since Netscape and Internet Explorer make all cells in a column the same width, you should place a `width` attribute in only one cell within a column, preferably the first instance of the cell in the first row, for source readability. If two or more cells in the same column happen to have `width` attributes, the widest one is honored. You can't make a column thinner than the minimum needed to display all of the cells in the column. So, if the browser determines that the column of cells needs to be at least 150 pixels wide to accommodate all the cells' contents, it ignores a `width` attribute in one of the column's cell tags that attempts to make the cell only 100 pixels wide.

### 10.2.4.3 The height attribute

The `height` attribute lets you specify a minimum height, in pixels, for the current cell. Since all cells in a row have the same height, this attribute need only be specified on one cell in the row, preferably the first. If some other cell in the row needs to be taller to accommodate its contents, the browser ignores the `width` attribute, and all the cells in the row are set to the larger size.

By default, all the cells in a row are the height of the largest cell in the row that just accommodates its contents.

### 10.2.4.4 The colspan attribute

It's common to have a table header that describes several columns beneath it, like the headers we used in [Figure 10-1](#). Use the `colspan` attribute in a table header or data tag to extend a table cell across two or more columns in its row. Set the value of the `colspan` attribute to an integer value equal to the number of columns you want the header or data cell to span. For example:

```
<td colspan="3">
```

tells the browser to make the cell occupy the same horizontal space as three cells in rows above or below it. The browser flows the contents of the cell to occupy the entire space.

What happens if there aren't enough extra cells on the right? The browser just extends the cell over as many columns as exist to the right; it doesn't add extra empty cells to each row to accommodate an overextended `colspan` value. You may defeat that limitation by adding the needed extra but contentless cells to a single row. (Give them a single `<br>` tag as their contents if you want Netscape's embossed border around them.)

### 10.2.4.5 The rowspan attribute

Just as the `colspan` attribute layers a table cell across several columns, the `rowspan` attribute stretches a cell down two or more rows in the table.

Include the `rowspan` attribute in the `<th>` or `<td>` tag of the uppermost row of the table where you want the cell to begin and set its value equal to the number of rows you want it to span. The cell then occupies the same space as the current row and an appropriate number of cells below that row. The browser flows the contents of the cell to occupy the entire extended space. For example:

```
<td rowspan="3">
```

creates a cell that occupies the current row plus the two rows below it.

Like the `colspan` attribute, the browser ignores overextended `rowspan` attributes and extends the current cell only down rows you've explicitly defined by other `<tr>` tags following the current row. The browsers do not add empty rows to a table to fill a `rowspan` below the last defined row in a table.

#### 10.2.4.6 Combining colspan and rowspan

You may extend a single cell both across several columns and down several rows by including both the `colspan` and `rowspan` attributes in its table header or data tag. For example:

```
<th colspan="3" rowspan="4">
```

creates a header cell that, as you might expect, spans across three columns and down four rows, including the current cell and extending two more cells to the right and three more cells down. The browser flows the contents of the cell to occupy the entire space, aligned inside according to the current row's alignment specifications or to those you explicitly include in the same tag, as described earlier.

#### 10.2.4.7 The nowrap attribute

Browsers treat each table cell as though it were a browser window unto itself, flowing contents inside the cell as they would common body contents (although subject to special table cell-alignment properties). Accordingly, the browsers automatically wrap text lines to fill the allotted table cell space. The `nowrap` attribute, when included in a table header or data tag, stops that normal word wrapping. With `nowrap`, the browser assembles the contents of the cell onto a single line, unless you insert a `<br>` or `<p>` tag, which forces a break so that the contents continue on a new line inside the table cell.

#### 10.2.4.8 The bgcolor and background attributes

Yet again, you can change the background color this time for an individual data cell. This attribute's value is either an RGB hexadecimal color value or a standard color name. Both the syntax of color values and the acceptable color names are provided in [Appendix G](#).

The `background` attribute, supported by both Internet Explorer and Netscape 6, supplies the URL of an image that is tiled to fill the background of the cell. The image is clipped if the cell is smaller than the image.

Neither `background` nor `bgcolor` overrides a related style-sheet property.

#### 10.2.4.9 The bordercolor, bordercolorlight, and bordercolordark attributes

Internet Explorer lets you alter the colors that make up an individual cell's border if table borders are turned on with the `border` attribute, of course. See the respective attributes' descriptions under the `<table>` tag in [Section 10.2.1.5](#) for details.

The values for these three attributes override any values set for the containing `<table>` or `<tr>` tag. Their values can be either RGB color values or standard color names, both of which are described fully in [Appendix G](#).

#### 10.2.4.10 The `char` and `charoff` attributes

Just as for the `<tr>` tag, you may use the `char` attribute with `<th>` or `<td>` to indicate which letter in the table cell should be the axis for alignment, such as for decimal numbers. You need not include a value with `char` in HTML. If you don't, the default character is language-based: it's a period in English, for example, and a comma in French. Include the `char` attribute and a single letter as its value to specify a different alignment character.

Use the `charoff` attribute and an integer value to specify the offset to the first occurrence of the alignment character in the cell. If a cell doesn't include the alignment character, it should be shifted horizontally to end at the alignment position.

The `char` and `charoff` attributes are standard in HTML 4 and XHTML but are not yet supported by any of the popular browsers.

#### 10.2.4.11 The `headers` and `scope` attributes

The `headers` attribute associates header cells with a data cell in the table. The value of this attribute is a quote-enclosed list of names that have been defined for various header cells using the `id` attribute. The `headers` attribute is especially useful for nonvisual browsers, which might speak the contents of a header cell before presenting the associated data cell contents.

Use the `scope` attribute to associate data cells with a header cell. With a value of `row`, all cells in the header's row are associated with the header cell. Specifying `col` binds all the cells in the current column to the cell.



Using `rowgroup` or `colgroup` binds all the cells in the cell's row group (defined by a `<thead>`, `<tbody>`, or `<tfoot>` tag) or column group (defined by a `<col>` or `<colgroup>` tag) with the header cell.

#### 10.2.4.12 The `abbr` attribute

The value of this attribute should be an abbreviated description of the cell's contents. When short on space, browsers might choose to render the abbreviation instead, or they might use it in nonvisual contexts.

#### 10.2.4.13 The `axis` attribute

Tables are usually chock full of data, prompting the reader to ask questions. A tabular expense report, for example, naturally leads to queries like "How much did I spend on meals?" or "What did my cab fares total?" In the future, browsers may support such queries with the help of the `axis` attribute.

The value of this attribute is a quote-enclosed list of category names that might be used to form a query. As a result, if you used `axis=meals` on the cells containing meal purchases, the browser could locate those cells, extract their values, and produce a sum.

### 10.2.5 The `<caption>` Tag

A table commonly needs a caption to explain its contents, so the popular browsers provide a table-caption tag. Authors typically place the `<caption>` tag and its contents immediately after the `<table>` tag, but it can be placed nearly anywhere inside the table and between the row tags. The caption may contain any body content, much like a cell within a table.

## <caption>

### *Function:*

Defines a table caption

### *Attributes:*

`align`, `class`, `dir`, `id`, `lang`, `onClick`, `onDbClick`,  
`onKeyDown`, `onKeyPress`, `onKeyUp`, `onMouseDown`,  
`onMouseMove`, `onMouseOut`, `onMouseOver`, `onMouseUp`,  
`style`, `title`, `valign` (□)

### *End tag:*

`</caption>`; never omitted

### *Contains:*

*body\_content*

### *Used in:*

*table\_content*

### 10.2.5.1 The align and valign attributes

By default, browsers place the caption's contents centered above the table. You may place it below the table with the `align` attribute set to the value `bottom` (the value `top`, of course, is equivalent to the default).

With Internet Explorer, you may alternatively use the `align` attribute to control the horizontal position of the caption and use the `valign` attribute to change the caption's vertical position. Set the `align` attribute to `left`, `center` (the default), or `right` to position the caption at the respective horizontal location relative to the table, and use the `valign` attribute to place a caption at the `top` or `bottom` of the table. The other

browsers ignore Internet Explorer's different caption-alignment values and attributes.

### **10.2.5.2 The many other attributes**

Like the other table tags, `<caption>` supports the many and various language-, event-, and styles-related attributes, which are described in [Section 10.2.2](#). Use them in good health. Just be sure to use the contextual selector `TABLE CAPTION` when referring to caption styles at the document level or in external style sheets.

## 4.5 Physical Style Tags

Nine physical styles are provided by the current HTML and XHTML standards: bold, italic, monospaced, underlined, strikethrough, larger, smaller, superscripted, and subscripted text. Much to our relief, Netscape 6 has stopped supporting a tenth physical style, "blinking" text. All physical style tags require ending tags.

As we discuss physical tags in detail, keep in mind that they convey an acute styling for the immediate text. For more comprehensive, document-wide control of text display, use style sheets (see [Chapter 8](#)).

## Physical Style Tags

### *Function*

Specify physical styles for text

### *Attributes*

`class, dir, id, lang, onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, style, title`

### *End tags*

Never omitted

### *Contains*

*text*

### *Used in*

*text*

### 4.5.1 The **<b>** Tag

The `<b>` tag is the physical equivalent of the `<strong>` content-based style tag, but without the latter's extended meaning. The `<b>` tag explicitly boldfaces a character or segment of text that is enclosed between it and its corresponding end tag (`</b>`). If a boldface font is not available, the browser may use some other representation, such as reverse video or underlining.

### 4.5.2 The **<big>** Tag

The `<big>` tag makes it easy to increase the size of text. It couldn't be

simpler: the browser renders the text between the `<big>` tag and its matching `</big>` ending tag one font size larger than the surrounding text. If that text is already at the largest size, `<big>` has no effect. `[<font>]`

Even better, you can nest `<big>` tags to enlarge the text. Each `<big>` tag makes the text one size larger, up to a limit of size seven, as defined by the font model.

Be careful with your use of the `<big>` tag, though. Because browsers are quite forgiving and try hard to understand a tag, those that don't support `<big>` often interpret it to mean bold.

### 4.5.3 The `<blink>` Tag (Obsolete Extension)

Text contained between the `<blink>` tag and its end tag `</blink>` does just that: blinks on and off. Netscape for Macintosh, for example, simply and reiteratively reverses the background and foreground colors for the `<blink>`-enclosed text. Neither the HTML nor the XHTML standard includes `<blink>`; it was supported as an extension only by Netscape Navigator versions before Version 6.

We cannot effectively reproduce the animated effect in these static pages, but it is easy to imagine and best left to the imagination, too. Blinking text has two primary effects: it gets your reader's attention and then promptly annoys them to no end. Forget about blinking text.

### 4.5.4 The `<i>` Tag

The `<i>` tag is like the `<em>` content-based style tag. It and its necessary end tag (`</i>`) tell the browser to render the enclosed text in an italic or oblique typeface. If the typeface is not available to the browser, highlighting, reverse video, or underlining might be used.

### 4.5.5 The `<s>` Tag (Deprecated)

The `<s>` tag is an abbreviated form of the `<strike>` tag supported by both Internet Explorer and Netscape. It is now a deprecated tag in HTML 4 and XHTML, meaning don't use it; it will eventually go away.

### 4.5.6 The `<small>` Tag

The `<small>` tag works just like its `<big>` counterpart (see [Section 4.5.2](#)), except it decreases the size of text instead of increasing it. If the enclosed text is already at the smallest size supported by the font model, `<small>` has no effect.

As with `<big>`, you can nest `<small>` tags to sequentially shrink text. Each `<small>` tag makes the text one size smaller than the containing `<small>` tag, to a limit of size 1.

### 4.5.7 The `<strike>` Tag (Deprecated)

The popular browsers put a line through ("strike through") text that appears inside the `<strike>` tag and its `</strike>` end tag. Presumably, it is an editing markup that tells the reader to ignore the text passage, reminiscent of the days before typewriter correction tape. You'll rarely, if ever, see the tag in use today: it is deprecated in HTML 4 and XHTML, just one step away from complete elimination from the standard.

### 4.5.8 The `<sub>` Tag

The text contained between the `<sub>` tag and its `</sub>` end tag gets displayed half a character's height lower, but in the same font and size as the current text flow. Both `<sub>` and its `<sup>` counterpart are useful for math equations and in scientific notation, as well as with chemical formulæ.

### 4.5.9 The `<sup>` Tag

The `<sup>` tag and its `</sup>` end tag superscripts the enclosed text; it gets displayed half a character's height higher, but in the same font and size as the current text flow. This tag is useful for adding footnotes to your documents, along with exponential values in equations. In combination with the `<a>` tag, you can create nice, hyperlinked footnotes:

```
The larval quat
```

```
weevil<a href="footnotes.html#note74"><sup><small>74</small></a>
```

This example assumes that *footnotes.html* contains all your footnotes, appropriately delimited as named document fragments.

#### 4.5.10 The `<tt>` Tag

Like the `<code>` and `<kbd>` tags, the `<tt>` tag and its necessary `</tt>` end tag direct the browser to display the enclosed text in a monospaced typeface. For those browsers that already use a monospaced typeface, this tag may make no discernible change in the presentation of the text.

#### 4.5.11 The `<u>` Tag (Deprecated)

This tag tells the browser to underline the text contained between the `<u>` and the corresponding `</u>` tag. The underlining technique is simplistic, drawing the line under spaces and punctuation as well as the text. This tag is deprecated in HTML 4 and XHTML, but the popular browsers support it.

The same display effects for the `<u>` tag are better achieved by using style sheets, covered in [Chapter 8](#).

#### 4.5.12 The `dir` and `lang` Attributes

The `dir` attribute lets you advise the browser which direction the text within the physical tag should be displayed in, and `lang` lets you specify



the language used within the tag. [\[Section 3.6.1.1\]](#) [\[Section 3.6.1.2\]](#)

### 4.5.13 The class, style, id, and title Attributes

Although each physical tag has a defined style, you can override that style by defining your own look for each tag. This new look can be applied to the physical tags using either the `style` or `class` attributes. [\[Section 8.1.1\]](#) [\[Section 8.3\]](#)

You also may assign a unique id to the physical style tag, as well as a less rigorous title, using the respective attribute and accompanying quote-enclosed string value. [\[Section 4.1.1.4\]](#) [\[Section 4.1.1.4\]](#)

### 4.5.14 Event Attributes

As with content-based style tags, user-initiated mouse and keyboard events can happen in and around a physical style tag's contents. Many of these events are recognized by the browser if it conforms to current standards, and, with the respective "on" attribute and value, you may react to the event by displaying a user dialog box or activating some multimedia event. [\[Section 12.3.3\]](#)

### 4.5.15 Summary of Physical Style Tags

The various graphical browsers render text inside the physical style tags in a similar fashion. [Table 4-2](#) summarizes these browsers' display styles for the native tags. Style-sheet definitions may override these native display styles.

**Table 4-2. Physical style tags**

Tag	Meaning	Display style
<code>&lt;b&gt;</code>	Bold contents	<b>bold</b>

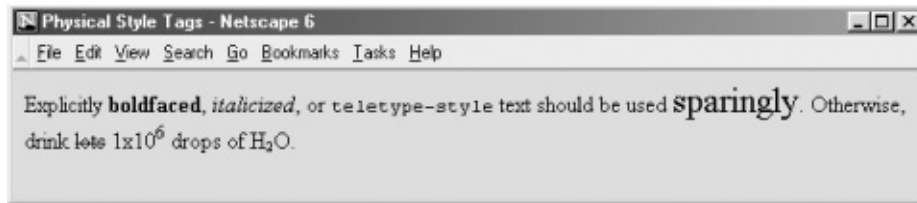
<code>&lt;big&gt;</code>	Increased font size	bigger text
<code>&lt;blink&gt;</code> (obsolete)	Alternating fore- and background colors	blinking text
<code>&lt;i&gt;</code>	Italic contents	<i>italic</i>
<code>&lt;small&gt;</code>	Decreased font size	smaller text
<code>&lt;s&gt;</code> , <code>&lt;strike&gt;</code> (deprecated)	Strikethrough text	<del>strike</del>
<code>&lt;sub&gt;</code>	Subscripted text	sub <sup>script</sup>
<code>&lt;sup&gt;</code>	Superscripted text	sup <sup>erscript</sup>
<code>&lt;tt&gt;</code>	Teletypewriter style	monospaced
<code>&lt;u&gt;</code> (deprecated)	Underlined contents	<u>underlined</u>

The following HTML source example illustrates some of the various physical tags as rendered by Netscape (see [Figure 4-12](#)):

```
Explicitly <b>boldfaced</b>, <i>italicized</i>, or
<tt>teletype-style</tt> text should be used
<big><big>sparingly</big></big>.
```

```
Otherwise, drink <strike>lots</strike> 1x10<sup>6</sup>
drops of H<sub><small><small>2</small></small></sub>O.
```

**Figure 4-12. Use physical text tags with caution**



## 4.5.16 Allowed Content

Any physical style tag may contain any item allowed in text, including conventional text, anchors, images, and line breaks. You can also combine physical style tags with other content-based tags.

## 4.5.17 Allowed Usage

Any physical style tag may be used anywhere an item allowed in text can be used. In general, this means anywhere within a document except in the `<title>`, `<listing>`, and `<xmp>` tags. You can use a physical style tag in a heading, but the browser will probably override and ignore its effect in lieu of the heading tag.

## 4.5.18 Combining Physical Styles

You will probably have better luck combining physical tags than you might have combining content-based tags to achieve multiple effects. For instance, Netscape renders the following in bold and italic typeface:

```
<b><i>Thar she blows!</i></b>
```

In practice, other browsers may elect to ignore such nesting. The HTML 4 standard does require the browser to "do its best" to support every possible combination of styles, but it does not define how the browser should handle such combinations. Although most browsers make a good attempt at doing so, do not assume that all combinations will be available to you.

## 16.4 XHTML 1.1

In May 2001, the W3C released an updated XHTML standard, XHTML 1.1. While most standards expand upon their previous versions, XHTML 1.1 takes the unusual step of defining a more restrictive version of XHTML. If you think of XHTML 1.0 as unwieldy, picky, and time-consuming, you'll find XHTML 1.1 even more so. In our opinion, XHTML 1.1 is an example of the standards process taken to absurd levels, defining a standard that may be academically pure but is essentially unusable.

### 16.4.1 Differences in XHTML 1.1

XHTML 1.1 begins with the XHTML 1.0 strict DTD and makes a few modifications. By supporting only the strict version of XHTML 1.0, Version 1.1 eliminates all deprecated elements and all browser extensions still in common use on the Web. It also makes the following minor changes:

- The `lang` attribute has been removed from every element. Instead, authors should use the `xml:lang` attribute.
- The `name` attribute has been removed from the `<a>` and `<map>` elements. Authors should use the `id` attribute in its place.

Finally, the XHTML 1.1 standard defines a new set of elements that implement a typographic feature known as "ruby" text. Ruby text is short runs of text placed alongside the base text; it is often used to annotate the text or to indicate pronunciation. Ruby text has its roots in East Asian documents, particularly Chinese schoolbooks and Japanese books and magazines. Ruby text is typically displayed in a smaller font<sup>[7]</sup> than the base text and follows certain alignment rules to ensure that it appears adjacent to the appropriate base text element.

[7] The origin of the name "ruby" lies in the name used by printers for the 5.5-point font used by the British press to set this smaller adjacent text.

Ruby text is defined and managed with a set of elements that provides grouping and layout control. We'll be blunt: this new feature is so esoteric and of so little importance to the vast majority of HTML authors even those who would subject themselves to the needless agony of XHTML 1.1 conformance that it does not warrant extensive coverage in this book. For those who are interested, a complete discussion of ruby text can be found at <http://www.w3.org/TR/ruby/>.

For the rest of us, it is sufficient to know that there are a few new elements in XHTML 1.1 that you would be wise not to use in your own DTDs, if only to prevent confusion with the XHTML 1.1 DTD. These new elements are:

`<ruby>`

Defines a segment of ruby text

`<rb>`

Defines the ruby base text

`<rt>`

Defines the ruby text associated with the base text

`<rp>`

Is used as a "ruby parenthesis" to group related ruby elements

`<rbc>`

Serves as a ruby base text container to group several base text elements

`<rtc>`

Serves as a ruby text container to group several ruby elements

Should you encounter any of these elements in a document, refer to the

above-mentioned specification for the details of how they are used. In general, you'll find a single outer `<ruby>` element with at least one `<rb>` and `<rt>` element within it. Multiple `<rb>` and `<rt>` elements may be collected within an `<rp>` element or may be grouped within the `<rbc>` or `<rtc>` container elements.

## Appendix C. Cascading Style Sheet Properties Quick Reference

In the following table, we list, in alphabetical order, all the properties defined in the World Wide Web Consortium's Recommended Specification for Cascading Style Sheets, Level 2 (<http://www.w3.org/pub/WWW/TR/REC-CSS2/>).

As in other sections of this book, we use the Netscape and Internet Explorer icons to the far right of each property to show which browser supports that property. Properties with no icons currently are not supported by any browser. We also include the number of the section in this book that fully defines the property.

We include each property's possible values, defined as either an explicit keyword (shown in `constant width`) or as one of these values:

### *angle*

A numeric value followed by `deg`, `grad`, or `rad`.

### *color*

Either a color name or hexadecimal RGB value, as defined in [Appendix G](#), or an RGB triple of the form:

```
rgb(red, green, blue)
```

where *red*, *green*, and *blue* are either numbers in the range 0 to 255 or percentage values indicating the brightness of that color component. Values of 255 or 100% indicate that the corresponding color component is at its brightest; values of 0 or 0% indicate that the corresponding color component is turned off completely. For example:

```
rgb(27, 119, 207)
```

`rgb(50%, 75%, 0%)`

are both valid color specifications.

### *frequency*

A numeric value followed by `hz` or `khz`, indicating Hertz or kiloHertz.

### *length*

An optional sign (either + or -), immediately followed by a number (with or without a decimal point), immediately followed by a two-character unit identifier. For values of zero, the unit identifier may be omitted.

The unit identifiers `em` and `ex` refer to the overall height of the font and to the height of the letter "x," respectively. The unit identifier `px` is equal to a single pixel on the display device. The unit identifiers `in`, `cm`, `mm`, `pt`, and `pc` refer to inches, centimeters, millimeters, points, and picas, respectively. There are 72.27 points per inch and 12 points in a pica.

### *number*

An optional sign, immediately followed by a number (with or without a decimal point).

### *percent*

An optional sign, immediately followed by a number (with or without a decimal point), immediately followed by a percent sign. The actual value is computed as a percentage of some other element property, usually the element's size.

### *shape*

A shape keyword, followed by a parentheses-enclosed list of comma-separated shape-specific parameters. Currently, the only supported shape keyword is `rect`, which expects four numeric



parameters denoting the offsets of the top, right, bottom, and left edges of the rectangle.

*time*

A numeric value followed by `s` or `ms`, designating a time in seconds or milliseconds.

*url*

The keyword `url`, immediately followed (no spaces) by a left parenthesis, followed by a URL optionally enclosed in single or double quotes, followed by a matching right parenthesis. For example:

```
url("http://www.oreilly.com/catalog")
```

is a valid URL value.

Finally, some values are lists of other values and are described as a "list of" some other value. In these cases, a list consists of one or more of the allowed values, separated by commas.

If there are several different values allowed for a property, these alternative choices are separated by vertical bars (`|`).

If the standard defines a default value for the property, that value is *underlined*.

azimuth	<u>angle</u>   left-side   far-left   left   center-left   center   center-right   right   right   far-right   right-side	Determines the position around the listener at which a sound is played	Section 8.4.12.7
---------	---	--	------------------

background		Composite property for the background-attachment, background-color, background-image, background-position, and background-repeat properties; value is any of these properties' values, in any order	Section 8.4.5.6	<input type="checkbox"/>
background-attachment	scroll   fixed	Determines if the background image is fixed in the window or scrolls as the document scrolls	Section 8.4.5.1	<input type="checkbox"/>
background-color	color   <u>transparent</u>	Sets the background color of an element	Section 8.4.5.2	<input type="checkbox"/>
background-image	url   <u>none</u>	Sets the background image of an element	Section 8.4.5.3	<input type="checkbox"/>
background-position	percent   length   top   center   bottom   left   right	Sets the initial position of the element's background image, if specified; values normally are paired to provide x, y positions; default position is 0% 0%	Section 8.4.5.4	<input type="checkbox"/>
background-repeat	<u>repeat</u>   repeat-x   repeat-y   no-repeat	Determines how the background image is repeated (tiled) across an element	Section 8.4.5.5	<input type="checkbox"/>
		Sets all four of an element's borders; value is one or		

<code>border</code>		more of a <i>color</i> , a value for <code>border-width</code> , and a value for <code>border-style</code>	<a href="#">Section 8.4.7.6</a>	<input type="checkbox"/>
<code>border-bottom</code>		Sets an element's bottom border; value is one or more of a <i>color</i> , a value for <code>border-bottom-width</code> , and a value for <code>border-style</code>	<a href="#">Section 8.4.7.6</a>	<input type="checkbox"/>
<code>border-bottom-width</code>	<i>length</i>   <u>thin</u>   <u>medium</u>   thick	Sets the thickness of an element's bottom border	<a href="#">Section 8.4.7.4</a>	<input type="checkbox"/>
<code>border-collapse</code>	collapse   separate	Sets the table border rendering algorithm	<a href="#">Section 8.4.9.1</a>	<input type="checkbox"/>
<code>border-color</code>	<i>color</i>	Sets the color of all four of an element's borders; default is the color of the element	<a href="#">Section 8.4.7.3</a>	<input type="checkbox"/>
<code>border-left</code>		Sets an element's left border; value is one or more of a <i>color</i> , a value for <code>border-left-width</code> , and a value for <code>border-style</code>	<a href="#">Section 8.4.7.6</a>	<input type="checkbox"/>
<code>border-left-width</code>	<i>length</i>   <u>thin</u>   <u>medium</u>   thick	Sets the thickness of an element's left border	<a href="#">Section 8.4.7.4</a>	<input type="checkbox"/>
<code>border-right</code>		Sets an element's right border; value is one or more of a <i>color</i> , a value for <code>border-right-width</code> ,	<a href="#">Section 8.4.7.6</a>	<input type="checkbox"/>

		and a value for <code>border-style</code>		
<code>border-right-width</code>	<i>length</i>   <code>thin</code>   <u><code>medium</code></u>   <code>thick</code>	Sets the thickness of an element's right border	<a href="#">Section 8.4.7.4</a>	<input type="checkbox"/>
<code>border-spacing</code>		With separate borders, sets the spacing between borders; one value sets vertical and horizontal spacing; two values sets horizontal and vertical spacing, respectively	<a href="#">Section 8.4.9.1</a>	
<code>border-style</code>	<code>dashed</code>   <code>dotted</code>   <code>double</code>   <code>groove</code>   <code>inset</code>   <u><code>none</code></u>   <code>outset</code>   <code>ridge</code>   <code>solid</code>	Sets the style of all four of an element's borders	<a href="#">Section 8.4.7.5</a>	<input type="checkbox"/>
<code>border-top</code>		Sets an element's top border; value is one or more of a <i>color</i> , a value for <code>border-top-width</code> , and a value for <code>border-style</code>	<a href="#">Section 8.4.7.6</a>	<input type="checkbox"/>
<code>border-top-width</code>	<i>length</i>   <code>thin</code>   <u><code>medium</code></u>   <code>thick</code>	Sets the thickness of an element's top border	<a href="#">Section 8.4.7.4</a>	<input type="checkbox"/>
<code>border-width</code>	<i>length</i>   <code>thin</code>   <u><code>medium</code></u>   <code>thick</code>	Sets the thickness of all four of an element's borders	<a href="#">Section 8.4.7.4</a>	<input type="checkbox"/>

bottom	<i>length   percent</i>	Used with the <code>position</code> property to place the bottom edge of an element	<a href="#">Section 8.4.7.14</a>	
caption-side	<code>top   bottom   left   right</code>	Sets the position for a table caption	<a href="#">Section 8.4.9.2</a>	
clear	<code>both   left   none   right</code>	Sets which margins of an element must not be adjacent to a floating element; the element is moved down until that margin is clear	<a href="#">Section 8.4.7.7</a>	<input type="checkbox"/>
clip	<i>shape</i>	Sets the clipping mask for an element	<a href="#">Section 8.4.7.8</a>	
color	<i>color</i>	Sets the color of an element	<a href="#">Section 8.4.5.7</a>	<input type="checkbox"/>
content		Inserts generated content around an element; see text for details	<a href="#">Section 8.4.11.2</a>	
counter-increment		Increments a counter by 1; value is a list of counter names, with each name optionally followed by a value by which it is incremented	<a href="#">Section 8.4.11.4</a>	
counter-reset		Resets a counter to zero; value is a list of counter names, with each name optionally followed by a	<a href="#">Section 8.4.11.4</a>	

		value to which it is reset		
cue-after	<i>url</i>   <u>none</u>	Plays the designated sound after an element is spoken	<a href="#">Section 8.4.12.5</a>	
cue-before	<i>url</i>   <u>none</u>	Plays the designated sound before an element is spoken	<a href="#">Section 8.4.12.5</a>	
display	<u>block</u>   inline   list-item   marker   none	Controls how an element is displayed	<a href="#">Section 8.4.10.1</a>	<input type="checkbox"/>
elevation	<i>angle</i>   below   level   above   higher   lower	Sets the height at which a sound is played	<a href="#">Section 8.4.12.7</a>	
empty-cells	hide   <u>show</u>	With separate borders, hides empty cells in a table	<a href="#">Section 8.4.9.1</a>	
float	left   <u>none</u>   right	Determines if an element floats to the left or right, allowing text to wrap around it or be displayed inline (using <i>none</i> )	<a href="#">Section 8.4.7.9</a>	<input type="checkbox"/>
font		Sets all the font attributes for an element; value is any of the values for <i>font-style</i> , <i>font-variant</i> , <i>font-weight</i> , <i>font-size</i> , <i>line-height</i> , and <i>font-family</i> , in that order	<a href="#">Section 8.4.3.8</a>	<input type="checkbox"/>

<code>font-family</code>	List of font names	Defines the font for an element, either as a specific font or as one of the generic fonts <code>serif</code> , <code>sans-serif</code> , <code>cursive</code> , <code>fantasy</code> , and <code>monospace</code>	<a href="#">Section 8.4.3.1</a>	<input type="checkbox"/>
<code>font-size</code>	<code>xx-small</code>   <code>x-small</code>   <code>small</code>   <u><code>medium</code></u>   <code>large</code>   <code>x-large</code>   <code>xx-large</code>   <code>larger</code>   <code>smaller</code>   <i><code>length</code></i>   <i><code>percent</code></i>	Defines the font size	<a href="#">Section 8.4.3.2</a>	<input type="checkbox"/>
<code>font-size-adjust</code>	<code>none</code>   <i><code>ratio</code></i>	Adjusts the current font's aspect ratio	<a href="#">Section 8.4.3.4</a>	
<code>font-stretch</code>	<code>wider</code>   <code>normal</code>   <code>narrower</code>   <code>ultra-condensed</code>   <code>extra-condensed</code>   <code>condensed</code>   <code>semi-condensed</code>   <code>semi-expanded</code>   <code>expanded</code>   <code>extra-expanded</code>   <code>ultra-expanded</code>	Determines the amount to stretch the current font	<a href="#">Section 8.4.3.3</a>	

font-style	<u>normal</u>   italic   oblique	Defines the style of the face, either normal or some type of slanted style	<a href="#">Section 8.4.3.5</a>	<input type="checkbox"/>
font-variant	<u>normal</u>   small-caps	Defines a font to be in small caps	<a href="#">Section 8.4.3.6</a>	<input type="checkbox"/>
font-weight	<u>normal</u>   bold   bolder   lighter   <i>number</i>	Defines the font weight if a <i>number</i> is used, it must be a multiple of 100 between 100 and 900; 400 is normal, 700 is the same as the keyword <code>bold</code>	<a href="#">Section 8.4.3.7</a>	<input type="checkbox"/>
height	<i>length</i>   <u>auto</u>	Defines the height of an element	<a href="#">Section 8.4.7.10</a>	<input type="checkbox"/>
left	<i>length</i>   <i>percent</i>	Used with the <code>position</code> property to place the left edge of an element	<a href="#">Section 8.4.7.14</a>	<input type="checkbox"/>
letter-spacing	<i>length</i>   <u>normal</u>	Inserts additional space between text characters	<a href="#">Section 8.4.6.1</a>	<input type="checkbox"/>
line-height	<i>length</i>   <i>number</i>   <i>percent</i>   <u>normal</u>	Sets the distance between adjacent text baselines	<a href="#">Section 8.4.6.2</a>	<input type="checkbox"/>
list-style		Defines list-related styles using any of the values for <code>list-style-image</code> , <code>list-style-position</code> , and <code>list-style-type</code>	<a href="#">Section 8.4.8.4</a>	<input type="checkbox"/>



<code>list-style-image</code>	<code>url   <u>none</u></code>	Defines an image to be used as a list item's marker, in lieu of the value for <code>list-style-type</code>	<a href="#">Section 8.4.8.1</a>	<input type="checkbox"/>
<code>list-style-position</code>	<code>inside   <u>outside</u></code>	Indents or extends (default) a list item's marker with respect to the item's content	<a href="#">Section 8.4.8.2</a>	<input type="checkbox"/>
<code>list-style-type</code>	<code>circle   <u>disc</u>   square   decimal   lower-alpha   lower-roman   none   upper-alpha   upper-roman</code>	Defines a list item's marker either for unordered lists ( <code>circle</code> , <code>disc</code> , or <code>square</code> ) or for ordered lists ( <code>decimal</code> , <code>lower-alpha</code> , <code>lower-roman</code> , <code>none</code> , <code>upper-alpha</code> , or <code>upper-roman</code> )	<a href="#">Section 8.4.8.3</a>	<input type="checkbox"/>
<code>margin</code>	<code>length   percent   auto</code>	Defines all four of an element's margins	<a href="#">Section 8.4.7.11</a>	<input type="checkbox"/>
<code>margin-bottom</code>	<code>length   percent   auto</code>	Defines the bottom margin of an element; default value is 0	<a href="#">Section 8.4.7.11</a>	<input type="checkbox"/>
<code>margin-left</code>	<code>length   percent   auto</code>	Defines the left margin of an element; default value is 0	<a href="#">Section 8.4.7.11</a>	<input type="checkbox"/>
<code>margin-right</code>	<code>length   percent   auto</code>	Defines the right margin of an element; default value is 0	<a href="#">Section 8.4.7.11</a>	<input type="checkbox"/>
<code>margin-top</code>	<code>length   percent   auto</code>	Defines the top margin of an element; default value is 0	<a href="#">Section 8.4.7.11</a>	<input type="checkbox"/>

<code>orphans</code>	<i>number</i>	Sets the minimum number of lines allowed in an orphaned paragraph fragment	<a href="#">Section 8.4.13.5</a>	
<code>overflow</code>	<code>auto   hidden   scroll   visible</code>	Determines how overflow content is rendered	<a href="#">Section 8.4.7.13</a>	
<code>padding</code>		Defines all four padding amounts around an element	<a href="#">Section 8.4.7.12</a>	<input type="checkbox"/>
<code>padding-bottom</code>	<i>length   percent</i>	Defines the bottom padding of an element; default value is 0	<a href="#">Section 8.4.7.12</a>	<input type="checkbox"/>
<code>padding-left</code>	<i>length   percent</i>	Defines the left padding of an element; default value is 0	<a href="#">Section 8.4.7.12</a>	<input type="checkbox"/>
<code>padding-right</code>	<i>length   percent</i>	Defines the right padding of an element; default value is 0	<a href="#">Section 8.4.7.12</a>	<input type="checkbox"/>
<code>padding-top</code>	<i>length   percent</i>	Defines the top padding of an element; default value is 0	<a href="#">Section 8.4.7.12</a>	<input type="checkbox"/>
<code>page</code>	<i>name</i>	Associates a named page layout with an element	<a href="#">Section 8.4.13.3</a>	
<code>page-break-after</code>	<code>auto   always   avoid   left   right</code>	Forces or suppresses page breaks after an element	<a href="#">Section 8.4.13.4</a>	

page-break-before	<code>auto</code>   <code>always</code>   <code>avoid</code>   <code>left</code>   <code>right</code>	Forces or suppresses page breaks before an element	<a href="#">Section 8.4.13.4</a>
page-break-inside	<code>auto</code>   <code>avoid</code>	Suppresses page breaks within an element	<a href="#">Section 8.4.13.4</a>
pause-after	<i>percent</i>   <i>time</i>	Pauses after speaking an element	<a href="#">Section 8.4.12.4</a>
pause-before	<i>percent</i>   <i>time</i>	Pauses before speaking an element	<a href="#">Section 8.4.12.4</a>
pitch	<i>frequency</i>   <i>x-low</i>   <i>low</i>   <i>medium</i>   <i>high</i>   <i>x-high</i>	Sets the average pitch of an element's spoken content	<a href="#">Section 8.4.12.3</a>
pitch-range	<i>number</i>	Sets the range of the pitch, from 0 (flat) to 100 (broad); default is 50	<a href="#">Section 8.4.12.3</a>
play-during	<i>url</i>   <i>mix</i>   <i>none</i>   <i>repeat</i>	If a URL is provided, it is played during an element's spoken content specifying <i>repeat</i> loops the audio; <i>mix</i> causes it to mix with, rather than replace, other background audio	<a href="#">Section 8.4.12.6</a>
position	<i>absolute</i>   <i>fixed</i>   <i>relative</i>	Sets the positioning model for an element	<a href="#">Section 8.4.7.14</a>

	<u>static</u>		
quotes	List of strings	Sets the quote symbols used to quote text	<a href="#">Section 8.4.11.3</a>
richness	<i>number</i>	Sets the richness of the voice, from 0 (flat) to 100 (mellifluous); default is 50	<a href="#">Section 8.4.12.3</a>
right	<i>length</i>   <i>percent</i>	Used with the <code>position</code> property to place the right edge of an element	<a href="#">Section 8.4.7.14</a>
speak	<u>normal</u>   <u>none</u>   <u>spell-out</u>	Determines how an element's content is spoken	<a href="#">Section 8.4.12.2</a>
speak-header	<u>always</u>   <u>once</u>	Determines if table headers are spoken once for each row or column or each time a cell is spoken	<a href="#">Section 8.4.9.3</a>
speak-numeral	<u>continuous</u>   <u>digits</u>	Determines how numerals are spoken	<a href="#">Section 8.4.12.2</a>
speak-punctuation	<code>code</code>   <u>none</u>	Determines if punctuation is spoken or used for inflection	<a href="#">Section 8.4.12.2</a>
speech-rate	<i>number</i>   <code>x-slow</code>   <code>slow</code>   <u>medium</u>   <code>fast</code>   <code>x-fast</code>   <code>faster</code>   <code>slower</code>	Sets the rate of speech; a <i>number</i> sets the rate in words per minute	<a href="#">Section 8.4.12.3</a>

<code>stress</code>	<i>number</i>	Sets the stress of the voice, from 0 (catatonic) to 100 (hyperactive); default is 50	<a href="#">Section 8.4.12.3</a>	
<code>table-layout</code>	<u>auto</u>   <code>fixed</code>	Determines the table-rendering algorithm	<a href="#">Section 8.4.9.4</a>	
<code>text-align</code>	<code>center</code>   <code>justify</code>   <code>left</code>   <code>right</code>	Sets the text alignment style for an element	<a href="#">Section 8.4.6.3</a>	<input type="checkbox"/>
<code>text-decoration</code>	<code>blink</code>   <code>line-through</code>   <u><code>none</code></u>   <code>overline</code>   <code>underline</code>	Defines any decoration for the text; values may be combined	<a href="#">Section 8.4.6.4</a>	<input type="checkbox"/>
<code>text-indent</code>	<i>length</i>   <i>percent</i>	Defines the indentation of the first line of text in an element; default is 0	<a href="#">Section 8.4.6.5</a>	<input type="checkbox"/>
<code>text-shadow</code>	See <code>text</code>	Creates text drop shadows of varying colors and offsets	<a href="#">Section 8.4.6.6</a>	
<code>text-transform</code>	<code>capitalize</code>   <code>lowercase</code>   <u><code>none</code></u>   <code>uppercase</code>	Transforms the text in the element accordingly	<a href="#">Section 8.4.6.7</a>	<input type="checkbox"/>
<code>top</code>	<i>length</i>   <i>percent</i>	Used with the <code>position</code> property to place the top edge of an element	<a href="#">Section 8.4.7.14</a>	
	<i>percent</i>			

vertical-align	baseline   bottom   middle   sub   super   text-bottom   text-top   top	Sets the vertical positioning of an element	<a href="#">Section 8.4.6.8</a>	<input type="checkbox"/>
visibility	collapse   hidden   visible	Determines if an element is visible in the document or table	<a href="#">Section 8.4.7.15</a>	
voice-family	List of voices	Selects a named voice family to speak an element's content	<a href="#">Section 8.4.12.3</a>	
volume	<i>number</i>   <i>percent</i>   silent   x- soft   soft   medium   loud   x- loud	Sets the volume of spoken content; numeric values range from 0 to 100	<a href="#">Section 8.4.12.1</a>	
white-space	<u>normal</u>   nowrap   pre	Defines how whitespace within an element is handled	<a href="#">Section 8.4.10.2</a>	<input type="checkbox"/>
widows	<i>number</i>	Sets the minimum number of lines allowed in a widowed paragraph fragment	<a href="#">Section 8.4.13.5</a>	
width	<i>length</i>   <i>percent</i>   <u>auto</u>	Defines the width of an element	<a href="#">Section 8.4.7.16</a>	<input type="checkbox"/>
word-spacing	<i>length</i>   <u>normal</u>	Inserts additional space between words	<a href="#">Section 8.4.6.9</a>	<input type="checkbox"/>

<code>z-index</code>	<i>number</i>	Sets the rendering layer for the current element	<a href="#">Section 8.4.7.17</a>
----------------------	---------------	--	----------------------------------

## **Appendix G. Color Names and Values**

With the popular browsers, and according to the Cascading Style Sheets (CSS) standard, you may prescribe the display color for various elements in your documents. You do so by specifying a color value or a standard name. The user may override these color specifications through her browser preferences.



## 7.3 The `<li>` Tag

It should be quite obvious to you by now that the `<li>` tag defines an item in a list. It's the universal tag for list items in ordered (`<ol>`) and unordered (`<ul>`) lists, as we discussed earlier, and for directories (`<dir>`) and menus (`<menu>`), which we discuss in detail later in this chapter.

## <li>

### *Function*

Defines an item within an ordered, unordered, directory, or menu list

### *Attributes*

`class, dir, id, lang, onClick, onDbClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, style, title, type, value`

### *End tag*

`</li>`; often omitted in HTML

### *Contains*

*flow*

### *Used in*

*list\_content*

Because the end of a list element can always be inferred by the surrounding document structure, most authors omit the ending `</li>` tags for their HTML list elements. That makes sense because it becomes easier to add, delete, and move elements around within a list. However, XHTML requires the end tag, so it's best to get used to including it in your documents.

Although universal in meaning, there are some differences and restrictions to the use of the `<li>` tag for each list type. In unordered and ordered lists, what follows the `<li>` tag may be nearly anything, including other lists and multiple paragraphs. Typically, if it handles indentation at all, the browser successively indents nested list items, and

the content in those items is justified to the innermost indented margin.

Directory and menu lists are another matter. They are lists of short items, like a single word or simple text blurb and nothing else. Consequently, `<li>` items within `<dir>` and `<menu>` tags may not contain other lists or other block elements, including paragraphs, preformatted blocks, or forms.

Clean documents, fully compliant with the HTML and XHTML standards, should not contain any text or other document item inside the unordered, ordered, directory, or menu lists that is not contained within an `<li>` tag. Most browsers are tolerant of violations to this rule, but you can't hold the browser responsible for compliant rendering of exceptional cases, either.

### 7.3.1 Changing the Style and Sequence of Individual List Items

Just as you can change the bullet or numbering style for all of the items in an unordered or ordered list, you can change the style for individual items within those lists. With ordered lists, you also can change the value of the item number. As you'll see, the combinations of changing style and numbering can lead to a variety of useful list structures, particularly when included with nested lists. Do note, however, that the standards have deprecated these attributes in deference to their CSS counterparts.

#### 7.3.1.1 The `type` attribute

Acceptable values for the `type` attribute in the `<li>` tag are the same as the values for the appropriate list type: items within unordered lists may have their `type` set to `circle`, `square`, or `disc`, while items in an ordered list may have their `type` set to any of the values shown previously in [Table 7-1](#).

Be careful. With earlier browsers, such as Netscape Navigator and Internet Explorer Versions 4 and earlier, a change in the bullet or numbering type in one list item similarly affected subsequent items in the

list. Not so for HTML 4-compliant browsers, such as Netscape Version 6 and Internet Explorer Versions 5 and later! The `type` attribute's effects are acute and limited to only the current `<li>` tag. Subsequent items revert to the default type; each must contain the specified type.

The `type` attribute changes the display style of the individual list item's leading number, and only that item, but not the value of the number, which persistently increments by one. [Figure 7-4](#) shows the effect that changing the `type` for an individual item in an ordered list has on subsequent items, as rendered by Internet Explorer from the following XHTML source:

```
<ol>

  <li type=A>Changing the numbering type</li>

  <li type=I>Uppercase Roman numerals</li>

  <li type=i>Lowercase Roman numerals</li>

  <li type=1>Plain ol' numbers</li>

  <li type=a>Doesn't alter the order.</li>

  <li> &lt;-- But, although numbering continues sequen

  <li> types don't persist. See? I should've been a "g

</ol>
```

**Figure 7-4. Changing the numbering style for each item in an ordered list**

figs/htm5\_0704.gif

You can use the style sheet-related `style` and `class` attributes to effect individual type changes in ordered and unordered lists that may or may not affect subsequent list items. See [Chapter 8](#) for details (particularly [Section 8.4.8.5](#)).

### 7.3.1.2 The `value` attribute

The `value` attribute changes the numbers of a specific list item and all of the list items that follow it. Since the ordered list is the only type with sequentially numbered items, the `value` attribute is valid only when used within an `<li>` tag inside an ordered list.

To change the current and subsequent numbers attached to each item in an ordered list, simply set the `value` attribute to any integer. The following source uses the `value` attribute to jump the numbering on items in an XHTML ordered list:

```
<ol>

  <li>Item number 1</li>

  <li>And the second</li>


  <li value=9> Jump to number 9</li>

  <li>And continue with 10...</li>

</ol>
```

Netscape renders the results as shown in [Figure 7-5](#).

**Figure 7-5. The value attribute lets you change individual item numbers in an ordered list**



figs/htm5\_0705.gif

### 7.3.1.3 The style and class attributes

The `style` attribute for the `<li>` tag creates an inline style for the elements enclosed by the tag, overriding any other style rule in effect. The `class` attribute lets you format the content according to a predefined class of the `<li>` tag; its value is the name of that class. [[Section 8.1.1](#)] [[Section 8.3](#)]

### 7.3.1.4 The class, dir, id, lang, event, style, and title attributes

These attributes can be applied to individual list items and have similar effects for ordered and unordered lists. [[Section 7.1.1.3](#)] [[Section 7.1.1.4](#)] [[Section 7.1.1.5](#)] [[Section 7.1.1.6](#)]

## Appendix F. Character Entities

The following table lists the defined standard and proposed character entities for HTML and XHTML, as well as several that are nonstandard but generally supported.

Entity names, if defined, appear for their respective characters and can be used in the character-entity sequence `&name`; to define any character for display by the browser. Otherwise, or alternatively for named characters, use the character's three-digit numeral value in the sequence `&#nnn`; to specially define a character entity. Actual characters, however, may or may not be displayed by the browser, depending on the computer platform and user-selected font for display.

Not all 256 characters in the ISO character set appear in the table. Missing ones are not recognized by the browser as either named or numeric entities.

To be sure that your documents are fully compliant with the HTML 4.0 and XHTML 1.0 standards, use only those named character entities with no entries in the conformance column. Characters with a value of "!!!" in the conformance column are not formally defined by the standards; use them at your own risk.

Numeric entity	Named entity	Symbol	Description	Conformance
<code>&amp;#009;</code>			Horizontal tab	
<code>&amp;#010;</code>			Line feed	
<code>&amp;#013;</code>			Carriage return	

&#032;			Space	
&#033;		!	Exclamation point	
&#034;	&quot;	"	Quotation mark	
&#035;		#	Hash mark	
&#036;		\$	Dollar sign	
&#037;		%	Percent sign	
&#038;	&amp;	&	Ampersand	
&#039;		'	Apostrophe	
&#040;		(	Left parenthesis	
&#041;		)	Right parenthesis	
&#042;		*	Asterisk	
&#043;		+	Plus sign	
&#044;		,	Comma	
&#045;		-	Hyphen	
&#046;		.	Period	



&#047;		/	Slash	
&#048; - &#057;		0-9	Digits 0-9	
&#058;		:	Colon	
&#059;		;	Semicolon	
&#060;	&lt;	<	Less than sign	
&#061;		=	Equals sign	
&#062;	&gt;	>	Greater than sign	
&#063;		?	Question mark	
&#064;		@	Commercial at sign	
&#065; - &#090;		A-Z	Letters A-Z	
&#091;		[	Left square bracket	
&#092;		\	Backslash	
&#093;		]	Right square bracket	
&#094;		^	Caret	

&#095;		_	Underscore	
&#096;		`	Grave accent	
&#097; - &#122;		a-z	Letters a-z	
&#123;		{	Left curly brace	
&#124;			Vertical bar	
&#125;		}	Right curly brace	
&#126;		~	Tilde	
&#130;		,	Low left single quote	!!!
&#131;		₣	Florin	!!!
&#132;		„	Low left double quote	!!!
&#133;		...	Ellipsis	!!!
&#134;		‡	Dagger	!!!
&#135;		‡	Double dagger	!!!
&#136;		^	Circumflex	!!!
&#137;		‰	Permil	!!!

&#138;		Š	Capital S, caron	!!!
&#139;		<	Less than sign	!!!
&#140;		Œ	Capital OE ligature	!!!
&#142;		Ž	Capital Z, caron	!!!
&#145;		`	Left single quote	!!!
&#146;		'	Right single quote	!!!
&#147;		"	Left double quote	!!!
&#148;		"	Right double quote	!!!
&#149;		·	Bullet	!!!
&#150;		-	En dash	!!!
&#151;		—	Em dash	!!!
&#152;		~	Tilde	!!!
&#153;		™	Trademark	!!!
&#154;		š	Small s, caron	!!!

&#155;		>	Greater than sign	!!!
&#156;		Œ	Small oe ligature	!!!
&#158;		š	Small z, caron	!!!
&#159;		ÿ	Capital Y, umlaut	!!!
&#160;	&nbsp;		Nonbreaking space	
&#161;	&iexcl;	¡	Inverted exclamation point	
&#162;	&cent;	¢	Cent sign	
&#163;	&pound;	£	Pound sign	
&#164;	&curren;	¤	General currency sign	
&#165;	&yen;	¥	Yen sign	
&#166;	&brvbar;		Broken vertical bar	
&#167;	&sect;	§	Section sign	
&#168;	&uml;	¨	Umlaut	
&#169;	&copy;	©	Copyright	
&#170;	&ordf;	ª	Feminine ordinal	

&#171;	&laquo;	«	Left angle quote	
&#172;	&not;	¬	Not sign	
&#173;	&shy;	-	Soft hyphen	
&#174;	&reg;	®	Registered trademark	
&#175;	&macr;	—	Macron accent	
&#176;	&deg;	°	Degree sign	
&#177;	&plusmn;	±	Plus or minus	
&#178;	&sup2;	²	Superscript 2	
&#179;	&sup3;	³	Superscript 3	
&#180;	&acute;	´	Acute accent	
&#181;	&micro;	μ	Micro sign (Greek mu)	
&#182;	&para;	¶	Paragraph sign	
&#183;	&middot;	·	Middle dot	
&#184;	&cedil;	¸	Cedilla	
&#185;	&sup1;	¹	Superscript 1	

&#186;	&ordm;	º	Masculine ordinal	
&#187;	&raquo;	»	Right angle quote	
&#188;	&frac14;	<sup>1</sup> / <sub>4</sub>	Fraction one-fourth	
&#189;	&frac12;	<sup>1</sup> / <sub>2</sub>	Fraction one-half	
&#190;	&frac34;	<sup>3</sup> / <sub>4</sub>	Fraction three-fourths	
&#191;	&iquest;	¿	Inverted question mark	
&#192;	&Agrave;	À	Capital A, grave accent	
&#193;	&Aacute;	Á	Capital A, acute accent	
&#194;	&Acirc;	—	Capital A, circumflex accent	
&#195;	&Atilde;	Ã	Capital A, tilde	
&#196;	&Auml;	Ä	Capital A, umlaut	
&#197;	&Aring;	Å	Capital A, ring	
&#198;	&AElig;	Æ	Capital AE ligature	
&#199;	&Ccedil;	Ç	Capital C, cedilla	

&#200;	&Egrave;	È	Capital E, grave accent	
&#201;	&Eacute;	É	Capital E, acute accent	
&#202;	&Ecirc;	Ê	Capital E, circumflex accent	
&#203;	&Euml;	Ë	Capital E, umlaut	
&#204;	&Igrave;	Ì	Capital I, grave accent	
&#205;	&Iacute;	Í	Capital I, acute accent	
&#206;	&Icirc;	Î	Capital I, circumflex accent	
&#207;	&Iuml;	Ï	Capital I, umlaut	
&#208;	&ETH;	Ɑ	Capital eth, Icelandic	
&#209;	&Ntilde;	Ñ	Capital N, tilde	
&#210;	&Ograve;	Ò	Capital O, grave accent	
&#211;	&Oacute;	Ó	Capital O, acute accent	
&#212;	&Ocirc;	Ô	Capital O, circumflex accent	
&#213;	&Otilde;	Õ	Capital O, tilde	

&#214;	&Ouml;	Ö	Capital O, umlaut	
&#215;	&times;	x	Multiply sign	
&#216;	&Oslash;	Ø	Capital O, slash	
&#217;	&Ugrave;	Ù	Capital U, grave accent	
&#218;	&Uacute;	Ú	Capital U, acute accent	
&#219;	&Ucirc;	Û	Capital U, circumflex accent	
&#220;	&Uuml;	Ü	Capital U, umlaut	
&#221;	&Yacute;	Ý	Capital Y, acute accent	
&#222;	&THORN;	Þ	Capital thorn, Icelandic	
&#223;	&szlig;	ß	Small sz ligature, German	
&#224;	&agrave;	à	Small a, grave accent	
&#225;	&aacute;	á	Small a, acute accent	
&#226;	&acirc;	â	Small a, circumflex accent	
&#227;	&atilde;	ã	Small a, tilde	



&#228;	&auml;	ä	Small a, umlaut	
&#229;	&aring;	å	Small a, ring	
&#230;	&aelig;	æ	Small ae ligature	
&#231;	&ccedil;	ç	Small c, cedilla	
&#232;	&egrave;	è	Small e, grave accent	
&#233;	&eacute;	é	Small e, acute accent	
&#234;	&ecirc;	ê	Small e, circumflex accent	
&#235;	&euml;	ë	Small e, umlaut	
&#236;	&igrave;	ì	Small i, grave accent	
&#237;	&iacute;	í	Small i, acute accent	
&#238;	&icirc;	î	Small i, circumflex accent	
&#239;	&iuml;	ï	Small i, umlaut	
&#240;	&eth;	ð	Small eth, Icelandic	
&#241;	&ntilde;	ñ	Small n, tilde	
&#242;	&ograve;	ò	Small o, grave accent	

&#243;	&oacute;	ó	Small o, acute accent	
&#244;	&ocirc;	ô	Small o, circumflex accent	
&#245;	&otilde;	õ	Small o, tilde	
&#246;	&ouml;	ö	Small o, umlaut	
&#247;	&divide;	÷	Division sign	
&#248;	&oslash;	◌	Small o, slash	
&#249;	&ugrave;	ù	Small u, grave accent	
&#250;	&uacute;	ú	Small u, acute accent	
&#251;	&ucirc;	û	Small u, circumflex accent	
&#252;	&uuml;	ü	Small u, umlaut	
&#253;	&yacute;	ý	Small y, acute accent	
&#254;	&thorn;	◌	Small thorn, Icelandic	
&#255;	&yuml;	ÿ	Small y, umlaut	

## 7.4 Nesting Lists

Except inside directories or menus, lists nested inside other lists are fine. Menu and directory lists can be embedded within other lists. Indents for each nested list are cumulative, so do not nest lists too deeply; the list contents could quickly turn into a thin ribbon of text flush against the right edge of the browser document window.

### 7.4.1 Nested Unordered Lists

The items in each nested unordered list may be preceded by a different bullet character at the discretion of the browser. For example, Internet Explorer Version 2 for Macintosh used an alternating series of hollow, solid circular, and square bullets for the various nests in the following source fragment, as shown in [Figure 7-6](#) (other browsers to date haven't been as inventive):

```
<ul>

  <li>Morning Kumquat Delicacies</li>

  <ul>

    <li>Hot Dishes</li>

    <ul>

      <li>Kumquat omelet</li>

      <li>Kumquat waffles</li>

      <ul>

        <li>Country style</li>
```

```
        <li>Belgian</li>

    </ul>

    <li>Kumquats and toast</li>

</ul>

<li>Cold Dishes</li>

<ul>

    <li>Kumquats and cornflakes</li>

    <li>Pickled Kumquats</li>

    <li>Diced Kumquats</li>

</ul>

</ul>

</ul>
```

**Figure 7-6. Bullets change for nested unordered list items**

figs/htm5\_0706.gif

You can change the bullet style for each unordered list and even for individual list items (see the `type` attribute discussion in [Section 7.3.1.1](#)), but the repertoire of bullets is limited. For example, Internet Explorer 6 for Windows and Netscape render a solid disc for level-one items, an open circle for level two, and a solid square for subsequent levels.

## 7.4.2 Nested Ordered Lists

By default, browsers number the items in ordered lists beginning with the Arabic numeral 1, nested or not. It would be great if the standards numbered nested ordered lists in some rational, consecutive manner. For example, the items in the second nest of the third main ordered list might be successively numbered "3.2.1," "3.2.2," "3.2.3," and so on.

With the `type` and `value` attributes, however, you do have a lot more latitude in how you create nested ordered lists. An excellent example is the traditional style for outlining, which uses the many different ways of numbering items offered by the `type` attribute (see [Figure 7-7](#)):

```
<ol type="A">

  <li>A History of Kumquats</li>

  <ol type="1">

    <li>Early History</li>

    <ol type="a">

      <li>The Fossil Record</li>

      <li>Kumquats: The Missing Link?</li>

    </ol>

    <li>Mayan Use of Kumquats</li>
```


```
    <li>Kumquats in the New World</li>

</ol>

    <li>Future Use of Kumquats</li>

</ol>
```

**Figure 7-7. The type attribute lets you do traditional outlining with ordered lists**



figs/htm5\_0707.gif

## 6.6 Creating Searchable Documents

Another extensible form of an HTML link that does not use the `<a>` tag is one that causes the server to search a database for a document that contains a user-specified keyword or words. An HTML document that contains such a link is known as a *searchable* document.

### 6.6.1 The `<isindex>` Tag (Deprecated)

Before it was deprecated in both the HTML 4 and XHTML standards, authors used to use the `<isindex>` tag to pass keywords along with a search engine's URL to the server. The server then matched the keywords against a database of terms to select the next document for display. Today's authors mostly use forms to pass information to the server and supporting programs. See [Chapter 9](#) for details.

## **<isindex>**

### *Function*

Indicates that a document can be searched

### *Attributes*

`action` () , `class`, `dir`, `id`, `lang`, `prompt`, `style`, `title`

### *End tag*

None in HTML; `</isindex>` or `<isindex ... />` in XHTML

### *Contains*

Nothing

### *Used in*

*head\_content*

When a browser encounters the `<isindex>` tag, it adds a standard search interface to the document (rendered by Internet Explorer in [Figure 6-8](#)):

```
<html>

<head>

<title>Kumquat Advice Database</title>

<base href="cgi-bin/quat-query">

<isindex>

</head>
```



```
<body>

<h3>Kumquat Advice Database</h3>


<p>

Search this database to learn more about kumquats!

</body>

</html>
```

**Figure 6-8. A searchable document**



figs/htm5\_0608.gif

The user types a list of space-separated keywords into the field provided. When the user presses the Enter key, the browser automatically appends the query list to the end of a URL and passes the information to the server for further processing.

While the HTML and XHTML standards allow the deprecated `<isindex>` tag to be placed only in the document header, most browsers let the tag appear anywhere in the document and insert the search field in the content flow where the `<isindex>` tag appears. This convenient extension lets you add instructions and other useful elements before presenting the user with the actual search field.

### 6.6.1.1 The prompt attribute


The browser provides a leading prompt just above or to the left of the user-entry field. Internet Explorer's default prompt has even changed over the years. Version 5, for example, used "This is a searchable index. Enter search keywords:". Version 6's prompt is shown in [Figure 6-8](#). That default prompt is not the best for all occasions, so it is possible to change it with the `prompt` attribute.

When added to the `<isindex>` tag, the value of the `prompt` attribute is the string of text that precedes the keyword entry field placed in the document by the browser.

For example, compare [Figure 6-8](#) with [Figure 6-9](#), in which we added the following prompt to the previous source example:

```
<isindex prompt="To learn more about kumquats, enter a
```

**Figure 6-9. The prompt attribute creates custom prompts in searchable documents**



figs/htm5\_0609.gif

Older browsers ignore the `prompt` attribute, but there is little reason not to include a better prompt string for your more up-to-date readership.

### 6.6.1.2 The query URL

Besides the `<isindex>` tag in the header of a searchable document, the

other important element of this special tag is the query URL. By default, it is the URL of the source document itself not good if your document can't handle the query. Rather, most authors use the `<base>` attribute to point to a different URL for the search. [[<base>](#)]

The browser appends a question mark to the query URL, followed by the specified search parameters. Nonprintable characters are appropriately encoded; multiple parameters are separated by plus signs (+).

In the previous example, if a user typed "insect control" in the search field, the browser would retrieve the URL:

```
cgi-bin/quat-query?insect+control
```

### 6.6.1.3 The action attribute

For Internet Explorer only, you can specify the query URL for the index with the `action` attribute. The effect is exactly as if you had used the `href` attribute with the `<base>` tag: the browser links to the specified URL with the search parameters appended to the URL.

While the `action` attribute provides the desirable feature of divorcing the document's base URL from the search index URL, it will cause your searches to fail if the user is not using Internet Explorer. For this reason, we do not recommend that you use the `action` attribute to specify the query URL for the search.

### 6.6.1.4 The class, dir, id, lang, style, and title attributes

The `class` and `style` attributes allow you to supply display properties and class names to control the appearance of the tag, although their value seems limited for `<isindex>`. The `id` and `title` attributes allow you to create a name and title for the tag; the name might be referenced by a hyperlink. [[Section 4.1.1.4](#)] [[Section 4.1.1.4](#)] [[Section 8.1.1](#)] [[Section 8.3](#)]

The `dir` and `lang` attributes define the language used for this tag and the direction in which text is rendered. Again, their use is not apparent with `<isindex>`. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)]

#### **6.6.1.5 Server dependencies**

Like image maps, searchable documents require support from the server to make things work. How the server interprets the query URL and its parameters is not defined by the HTML or XHTML standards.

You should consult your server's documentation to determine how you can receive and use the search parameters to locate the desired document. Typically, the server breaks the parameters out of the query URL and passes them to a program designated by the URL.

## 6.8 Supporting Document Automation

There are two additional header tags that have the primary functions of supporting document automation and interacting with the web server itself and document-generation tools.

### 6.8.1 The `<meta>` Header Element

Given the rich set of header tags for defining a document and its relationship with others that go unused by most authors, you'd think we'd all be satisfied. But no, there's always someone with special needs. These authors want to be able to give even more information about their precious documents information that might be used by browsers, readers of the source, or document-indexing tools. The `<meta>` tag is for those of you who need to go beyond the beyond.

## **<meta>**

### *Function*

Supplies additional information about a document

### *Attributes*

`charset (□), content, dir, http_equiv, lang, name, scheme`

### *End tag*

None in HTML; `</meta>` or `<meta ... />` in XHTML

### *Contains*

Nothing

### *Used in*

*head\_content*

The `<meta>` tag belongs in the document header and has no content. Instead, attributes of the tag define name/value pairs that associate the document. In certain cases, these values are used by the web server serving the document to further define the document content type to the browser.

### **6.8.1.1 The name attribute**

The `name` attribute supplies the name of the name/value pair defined by the `<meta>` tag. Neither the HTML nor the XHTML standard specifies any predefined `<meta>` names. In general, you are free to use any name that makes sense to you and other readers of your source document.

One commonly used name is `keywords`, which defines a set of keywords for the document. When encountered by any of the popular

search engines on the Web, these keywords are used to categorize the document. If you want your documents to be indexed by a search engine, consider putting this kind of tag in the `<head>` of each document:

```
<meta name="keywords" content="kumquats, cooking, peel."
```

If the `name` attribute is not provided, the name of the name/value pair is taken from the `http-equiv` attribute.

### 6.8.1.2 The `content` attribute

The `content` attribute provides the value of the name/value pair. It can be any valid string (enclosed in quotes if it contains spaces). It should always be specified in conjunction with either a `name` or `http-equiv` attribute.

As an example, you might place the author's name in a document with:

```
<meta name="Authors" content="Chuck Musciano & Bill Ke."
```

### 6.8.1.3 The `http-equiv` attribute

The `http-equiv` attribute supplies a name for the name/value pair and instructs the server to include the name/value pair in the MIME document header that is passed to the browser before sending the actual document.

When a server sends a document to a browser, it first sends a number of name/value pairs. While some servers might send a number of these pairs, all servers send at least one:

```
content-type: text/html
```

This tells the browser to expect to receive an HTML document.

When you use the `<meta>` tag with the `http-equiv` attribute, the server

will add your name/value pairs to the content header it sends to the browser. For example, adding:

```
<meta http-equiv="charset" content="iso-8859-1">
```

```
<meta http-equiv="expires" content="31 Dec 99">
```

causes the header sent to the browser to contain:

```
content-type: text/html
```

```
charset: iso-8859-1
```

```
expires: 31 Dec 99
```

Of course, adding these additional header fields makes sense only if your browser accepts the fields and uses them in some appropriate manner.

#### **6.8.1.4 The charset attribute**

Internet Explorer provides explicit support for a `charset` attribute in the `<meta>` tag. Set the value of the attribute to the name of the character set to be used for the document. This is not the recommended way to define a document's character set. Rather, we recommend always using the `http-equiv` and `content` attributes to define the character set.

#### **6.8.1.5 The scheme attribute**

This attribute specifies the scheme to be used to interpret the property's value. This scheme should be defined within the profile specified by the `profile` attribute of the `<head>` tag. [[Section 3.7.1](#)]

### **6.8.2 The <nextid> Header Element (Archaic)**

This tag is not defined in the HTML 4 or XHTML standards and should



not be used. We describe it here for historical reasons.

## <nextid>

### *Function*

Defines the next valid document entity identifier

### *Attributes*

`n`

### *End tag*

None

### *Contains*

Nothing

### *Used in*

*head\_content*

The idea behind the `<nextid>` tag is to provide some way of automatically indexing fragment identifiers.

### 6.8.2.1 The `n` attribute

The `n` attribute specifies the name of the next generated fragment identifier. It is typically an alphabetic string followed by a two-digit number. A typical `<nextid>` tag might look like this:

```
<html>
```

```
<head>
```

```
<nextid n=DOC54>
```

</head>

...

An automatic document generator might use the `nextid` information to successively name fragment identifiers `DOC54`, `DOC55`, and so forth within this document.

## 12.2 Embedded Content

In this section, we cover three tags that support embedded content. The `<object>` tag is in the HTML 4 and XHTML standards. It is a generalized hybrid of the deprecated `<applet>` tag for embedding applets, particularly Java applets, and the `<embed>` tag extension that lets you include an object whose MIME type references the plug-in needed to process and possibly display that object.

The latest standards strongly encourage you to use the `<object>` tag to include applets and other discrete inclusions in your documents, including images (although the standards do not go so far as to deprecate the `<img>` tag). Use `<object>` with the `classid` attribute to insert Java and other applets into a document, along with their execution parameters as contents of the associated `<param>` tag. Use `<object>` with the `data` attribute to download and display non-HTML/XHTML content, such as multimedia, in the user's computing environment. Object data may be processed and rendered by an included applet, by utilities that come with your browser, or by a plug-in ("helper") application that the user supplies.

For applets, the browser creates a display region in the containing text flow exactly like an inline image: without line breaks and as a single large entity. The browser then downloads and executes the applet's program code, if specified, and downloads and renders any included data just after download and display of the document. Execution of the applet continues until the code terminates itself or when the user stops viewing the page containing the applet.

With data, the browser decodes the object's data type and either handles its rendering directly, such as with GIF and JPEG images, or invokes an associated plug-in application for the job.

### 12.2.1 The `<object>` Tag

The `<object>` tag was originally implemented by Microsoft to support its ActiveX controls. Only later did Microsoft add Java support. In a similar manner, Netscape initially supported the alternative `<embed>` and `<applet>` tags for inclusion objects and later provided limited support for the `<object>` tag.

All that jostling for position by the browser giants made us nervous, and we were hesitant in previous editions of this book to even suggest that you use `<object>` at all. We now heartily endorse it, based on the strength of the HTML 4 and (particularly) XHTML standards. Although it's not yet fully supported, expect `<object>` to be well supported soon by the popular browsers, and expect the alternative `<embed>` and `<applet>` tags to be less well supported, if not completely ignored, by future HTML 4/XHTML-compliant browsers.

## **<object>**

### *Function:*

Embeds an object or applet in a document

### *Attributes:*

`align`, `archive`, `border`, `class`, `classid`, `codebase`,  
`codetype`, `data`, `declare`, `dir`, `height`, `hspace`, `id`, `lang`,  
`name`, `notab` (□), `onClick`, `onDblClick`, `onKeyDown`,  
`onKeyPress`, `onKeyUp`, `onLoad`, `onMouseDown`, `onMouseMove`,  
`onMouseOut`, `onMouseOver`, `onMouseUp`, `shapes` (□),  
`standby`, `style`, `tabindex`, `title`, `type`, `usemap`, `vspace`,  
`width`

### *End tag:*

`</object>;` never omitted

### *Contains:*

*object\_content*

### *Used in:*

*text*

The contents of the `<object>` tag may be any valid HTML or XHTML content, along with `<param>` tags that pass parameters to an applet. If the browser can retrieve the requested object and successfully process it, either by executing the applet or by processing the object's data with a plug-in (helper) application, the contents of the `<object>` tag, except for the `<param>` tags, are ignored. If any problem occurs during the retrieval and processing of the object, the browser won't insert the object into the document but instead will display the contents of the `<object>` tag, except for the `<param>` tags. In short, you should provide alternative content in case the browsers cannot handle the `<object>` tag or the

object cannot be loaded successfully.

### 12.2.1.1 The `classid` attribute

Use the `classid` attribute to specify the location of the object, typically a Java class, that you want included by the browser. The value may be the absolute or relative URL of the desired object. Relative URLs are considered to be relative to the URL specified by the `codebase` attribute if it is provided; otherwise, they are relative to the current document's URL.

For example, to execute a clock Java applet contained in a file named *clock.class*, you might include in your HTML document the code:

```
<object classid="clock.class">  
  
</object>
```

The browser locates the code for the applet using the current document's base URL. Hence, if the current document's URL is:

```
http://www.kumquat.com/harvest_time.html
```

the browser retrieves the applet code for our *clock.class* example as:

```
http://www.kumquat.com/clock.class
```

### 12.2.1.2 The `codebase` attribute

Use the `codebase` attribute to provide an alternative base URL from which the browser should retrieve an object. The value of this attribute is a URL pointing to a directory containing the object referenced by the `classid` attribute. The `codebase` URL overrides, but does not permanently replace, the document's base URL, which is the default if you don't use `codebase`. [[Section 6.2](#)]

Continuing with our previous examples, suppose your document comes from *http://www.kumquat.com*, but the clock applet is kept in a separate directory named *classes*. You cannot retrieve the applet by specifying `classid="classes/clock.class"`. Rather, include the `codebase` attribute and new base URL:

```
<object classid="clock.class" codebase="http://www.kumquat.com/classes/">
  <img alt="A clock applet" data-bbox="112 220 890 283"/>
</object>
```

which resolves to the URL:

```
http://www.kumquat.com/classes/clock.class
```

Although we used an absolute URL in this example, you also can use a relative URL. For instance, applets typically are stored on the same server as the host documents, so we'd usually be better off, for relocation's sake, specifying a relative URL for the `codebase`, such as:

```
<object code="clock.class" codebase="/classes/">
  <img alt="A clock applet" data-bbox="112 489 797 554"/>
</object>
```

The `classid` attribute is similar to the `code` attribute of the `<applet>` tag, providing the name of the file containing the object; it is used in conjunction with the `codebase` attribute to determine the full URL of the object to be retrieved and placed in the document.

### 12.2.1.3 The archive attribute

For performance reasons, you may choose to preload collections of objects contained in one or more archives. This is particularly true of Java-based applications, where one Java class relies on many other classes to get its work done. The value of the `archive` attribute is a quote-enclosed list of URLs, each pointing to an archive to be loaded by the browser before it renders or executes the object.



### 12.2.1.4 The codetype attribute

The `codetype` attribute is required only if the browser cannot determine an applet's MIME type from the `classid` attribute or if the server does not deliver the correct MIME type when downloading an object. This attribute is nearly identical to `type` (see [Section 12.2.1.6](#)), except that it is used to identify program code type, whereas `type` should be used to identify data file types.

The following example explicitly tells the browser that the object's code is Java:

```
<object code="clock.class" codetype="application/java">

</object>
```

### 12.2.1.5 The data attribute

Use the `data` attribute to specify the data file, if any, that is to be processed by the object. The `data` attribute's value is the URL of the file, either absolute or relative to the document's base URL or to that which you provide with the `codebase` attribute. The browser determines the data type by the type of object that is being inserted in the document.

This attribute is similar to the `src` attribute of the `<img>` tag, in that it downloads data to be processed by the included object. The difference, of course, is that the `data` attribute lets you include just about any file type, not just an image file. In fact, the `<object>` tag expects, but doesn't require, that you explicitly name an enabling application for the object with the `classid` attribute, or indicate the MIME type of the file via the `type` attribute to help the browser decide how to process and render the data.

For example, here is an image included as an object, rather than as an `<img>` file:

```
<object data="pics/kumquat.gif" type="image/gif">

</object>
```

### 12.2.1.6 The type attribute

The `type` attribute lets you explicitly define the MIME type of the data that appears in the file you declare with the `data` attribute. (Use `codetype` to indicate an applet's MIME type.) If you don't provide data, or if the MIME type of the data is apparent from the URL or is provided by the server, you may omit this attribute. We recommend that you include it anyway, to ensure that the browser handles your data correctly.

For examples of data MIME types, look in your browser preferences for applications. There you'll find a list of the many file data types your browser recognizes and the application, if not the browser itself, that processes and renders that file type.

### 12.2.1.7 The align, class, border, height, hspace, style, vspace, and width attributes

There are several attributes, like the corresponding attributes for the `<img>` tag, that let you control the appearance of the `<object>` display region. The `height` and `width` attributes control the size of the viewing region. The `hspace` and `vspace` attributes define a margin around the viewing region. The value for each of these dimension attributes should be an actual number of pixels.

The `align` attribute determines how the browser aligns the region in context with the surrounding text.<sup>[4]</sup> Use `top`, `texttop`, `middle`, `absmiddle`, `baseline`, `bottom`, or `absbottom` to align the object display space with adjacent text, or `left` and `right` alignments for wraparound content.

[4] The `align` attribute is deprecated in the HTML 4 and XHTML standards because of the

CSS standard, but it is still popularly used and supported.

The display region's dimensions often must match some other applet requirement, so be careful to check these values with the applet programmer. Sometimes, the applet may scale its display output to match your specified region.

For instance, our example clock applet might grow or shrink to fit nearly any size display region. Instead, we might fix it to a square space, 100 x 100 pixels:

```
<object classid="clock.class" height="100" width="100">
```

```
</object>
```

As with `<img>`, use the `border` attribute to control the width of the frame that surrounds the object's display space when you include it as part of a hyperlink. The null value (`border=0`) removes the frame. [Section 5.2.6]

Use the `class` and `style` attributes to control the display style for the content enclosed by the tag and to format the content according to a predefined class of the `<object>` tag. [Section 8.1.1] [Section 8.3]

### 12.2.1.8 The `declare` attribute

The `declare` attribute lets you define an object but restrains the browser from downloading and processing it. Used in conjunction with the `name` attribute, this facility is similar to a forward declaration in a more conventional programming language that lets you defer download of an object until it actually gets used in the document.

### 12.2.1.9 The `id`, `name`, and `title` attributes

Use the `id` or `name` attribute to uniquely label an object. Use the `title` attribute to simply entitle the tag. Each attribute's value is a text string. The browser may choose to display a title to the user or may use it in

some other manner while rendering the document. Use `id` or `name` to reference the object in other elements of your document, including hyperlinks and other objects.

For example, suppose you have two clock applets in your document, along with two applets the user operates to set those clocks. Provide unique labels for the clock applets using the `name` or `id` attribute, then pass those labels to the setting applets using the `<param>` tag, which we discuss in [Section 12.2.2](#):

```
<object classid="clock.class" name="clock1">

</object>

<object classid="clock.class" name="clock2">

</object>

<object classid="setter.class">

    <param name="clockToSet" value="clock1">

</object>

<object classid="setter.class">

    <param name="clockToSet" value="clock2">

</object>
```

Since we have no need to distinguish between the setter applets, we choose not to identify their instances.

All the popular browsers support `name` and `id`.

#### **12.2.1.10 The shapes and usemap attributes**

Recall from our detailed discussion of hyperlinks in [Chapter 6](#), that you can divide a picture into geometric regions and attach a hyperlink to each, creating a so-called image map. The `shapes` and `usemap` attributes for the `<object>` tag generalize that feature to include other object types.

The standard `shapes` attribute informs the browser that the `<object>` tag's contents are a series of hyperlinks and shape definitions. The `usemap` attribute and required URL value point to a `<map>` where you define the shapes and associated hyperlinks, identical to the client-side image maps discussed in [Section 6.5.2](#).

For example, here is the image map we described in [Chapter 6](#), rewritten in XHTML as a "shaped" object:

```
<object data="pics/map.gif" shapes="shapes">

  <a shape="rect" coords="0,0,49,49" href="main.html#l

  <a shape="rect" coords="50,0,99,49" href="main.html#

  <a shape="rect" coords="0,50,49,99" href="main.html#

  <a shape="rect" coords="50,50,99,99" href="main.html

</object>
```

and as the more familiar image map:

```
<object data="pics/map.gif" usemap="#map1">

</object>

...

<map name="map1">

  <area coords="0,0,49,49" href="main.html#link1" />
```

```
<area coords="50,0,99,49" href="main.html#link2" />

<area coords="0,50,49,99" href="main.html#link3" />

<area coords="50,50,99,99" href="main.html#link4" />

</map>
```

You also may take advantage of all the attributes associated with the hyperlink, `<map>`, and `<area>` tags to define and arrange the image map regions. For instance, we recommend that you include alternative (`alt` attribute) text descriptions for each sensitive region of the image map.

#### 12.2.1.11 The `standby` attribute

The `standby` attribute lets you display a message the attribute's value text string during the time the browser is downloading the object data. If your objects are large or if you expect slow network responses, add this attribute as a courtesy to your users.

#### 12.2.1.12 The `tabindex` and `notab` attributes

For Internet Explorer with ActiveX objects only, the `notab` attribute excludes the object from the document tabbing order.

As an alternative to the mouse, users also may press the Tab key to select and the Return or Enter key to activate a hyperlink or to access a form control. Normally, each time the user presses the Tab key, the browser steps to the next hyperlink or form control, in the order in which they appear in the document. Use the HTML 4/XHTML standard `tabindex` attribute and an integer value to modify the position the object occupies in the sequence of Tab-selected elements on the page.

#### 12.2.1.13 The `dir` and `lang` attributes

Use the `dir` and `lang` attributes, like their counterparts for most other tags, to specify the language and dialect of the `<object>`-enclosed contents as well as the direction by which the browser adds text characters to the display. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)]

#### 12.2.1.14 Object event handling

As user-initiated mouse and keyboard events occur within the object, you may want to perform special actions. Accordingly, you can use the 10 standard event attributes to catch these events and execute JavaScript code. We describe JavaScript event handlers more fully in [Section 12.3.3](#).

#### 12.2.1.15 Supporting incompatible browsers

Since some browsers may not support applets or the `<object>` tag, sometimes you may need to tell readers what they are missing. You do this by including body content between the `<object>` and `</object>` tags.

Browsers that support the `<object>` tags ignore the extraneous content inside. Of course, browsers that don't support objects don't recognize the `<object>` tags. Being generally tolerant of apparent mistakes, browsers usually ignore the unrecognized tags and blithely go on to display whatever content appears inside. It's as simple as that. The following fragment tells object-incapable browser users that they won't see our clock example:

```
<object classid=clock.class>
```

```
    If your browser were capable of handling applets, you  
    a nifty clock right here!
```

```
</object>
```

More importantly, object-capable browsers display the contents of the `<object>` tag if they cannot load, execute, or render the object. If you have several objects of similar intent but with differing capabilities, you can nest their `<object>` tags. The browser tries each object in turn, stopping with the first one it can handle. Thus, the outermost object might be a full-motion video. Within that `<object>` tag, you might include a simpler MPEG video, and within that `<object>` tag, a simple GIF image. If the browser can handle full-motion video, your users get the full effect. If that level of video isn't available, the browser can try the simpler MPEG video stream. If that fails, the browser can just display the image. If images aren't possible, the innermost `<object>` tag might contain a text description of the object.

### 12.2.2 The `<param>` Tag

The `<param>` tag supplies parameters for a containing `<object>` or `<applet>` tag. (We discuss the deprecated `<applet>` tag in [Section 12.2.3](#).)



## **<param>**

### *Function:*

Supplies a parameter to an embedded object

### *Attributes:*

`id, name, type, value, valuetype`

### *End tag:*

None in HTML; `</param>` or `<param ... />` in XHTML

### *Contains:*

Nothing

### *Used in:*

*applet\_content*

The `<param>` tag has no content and, with HTML, no end tag. It appears, perhaps with other `<param>` tags, only between an `<object>` or `<applet>` tag and its end tag. Use the `<param>` tag to pass parameters to the embedded object, such as a Java applet, as required for it to function correctly.

### **12.2.2.1 The id, name, and value attributes**

The `<param>` tag has two required attributes: `name` or `id`, and `value`. You've seen these before with forms. Together, they define a name/value pair that the browser passes to the applet.

For instance, our clock applet example might let users specify the time zone by which it sets its hour hand. To pass the parameter identified as "timezone" with the value "EST" to our example applet, specify the

parameters as:

```
<object classid="clock.class">  
  
    <param name="timezone" value="EST" />  
  
</object>
```

The browser passes the name/value pairs to the applet, but that is no guarantee that the applet is expecting the parameters, that the names and values are correct, or that the applet will even use the parameters. Correct parameter names, including capitalization and acceptable values, are determined by the applet author. The wise HTML/XHTML author works closely with the applet programmer or has detailed documentation to ensure that the applet parameters are named correctly and assigned valid values.

#### 12.2.2.2 The `type` and `valuetype` attributes

Use the `type` and `valuetype` attributes to define the type of the parameter the browser passes to the embedded object and how that object is to interpret the value. The `valuetype` attribute can have one of three values: `data`, `ref`, or `object`. The value `data` indicates that the parameter value is a simple string. This is the default value. The `ref` value indicates that the value is a URL of some other resource on the Web. Finally, `object` indicates that the value is the name of another embedded object in the current document. This may be needed to support interobject communication within a document.

The value of the `type` attribute is the MIME media type of the value of the parameter. This usually is of no significance when the parameter value is a simple string, but it can be important when the value is actually a URL pointing to some other object on the Web. In those cases, the embedded object may need to know the MIME type of the object in order to use it correctly. For example, this parameter tells the embedded object that the parameter is actually the URL of a Microsoft Word document:

```
<param name="document" value="http://kumquats.com/quat  
type="application/msword" valuetype="ref" />
```

### 12.2.3 The <applet> Tag (Deprecated)

Use the `<applet>` tag within your documents to download and execute an applet. Also, use the tag to define a region within the document display for the applet's display area. You may supply alternative content within the `<applet>` tag for display by browsers that do not support applets.

## **<applet>**

### *Function:*

Inserts an application into the current text flow

### *Attributes:*

`align, alt, archive, class, code, codebase, height, hspace, id, mayscript (☐), name, object, style, title, vspace, width`

### *End tag:*

`</applet>;` never omitted

### *Contains:*

*applet\_content*

### *Used in:*

*text*

Most applets require one or more parameters that you supply in the document to control their execution. Put these parameters between the `<applet>` tag and its corresponding `</applet>` end tag, using the `<param>` tag. The browser will pass the document-specific parameters to the applet at the time of execution. [[Section 12.2.2](#)]

The `<applet>` tag has been deprecated in the HTML 4 and XHTML standards in deference to the generalized `<object>` tag, which can do the same as `<applet>` and much more. Nonetheless, `<applet>` is a popular tag and remains supported by the popular browsers.

### **12.2.3.1 Applet rendering**

The browser creates an applet's display region in the containing text flow exactly like an inline image: without line breaks and as a single large entity. The browser downloads and executes the applet just after download and display of the document and continues execution until the code terminates itself or the user stops viewing the page containing the applet.

### 12.2.3.2 The align attribute

As with an image, you can control the alignment of an applet's display region with respect to its surrounding text. As with the `<img>` tag, set the `align` attribute's value to `top`, `texttop`, `middle`, `absmiddle`, `baseline`, `bottom`, or `absbottom`, or use the `left` and `right` alignments for wraparound content. [[Section 5.2.6](#)]

### 12.2.3.3 The alt attribute

The `alt` attribute gives you a way to tell users gracefully that something is missing if, for some reason, the applet cannot or will not execute on their computer. Its value is a quote-enclosed message string that, like the `alt` attribute for images, gets displayed in lieu of the applet itself. The `alt` message is only for browsers that support applets. See [Section 12.2.1.15](#) to find out how to inform users of applet-incapable browsers why they can't view an applet.

### 12.2.3.4 The archive attribute

The `archive` attribute collects common Java classes into a single library that is cached on the user's local disk. Once cached, the browser doesn't need to use the network to access an applet; it retrieves the software from the local cache, thereby reducing the inherent delays of additional network activity to load the class.

The value of the `archive` attribute is a URL identifying the archive file.

The suffix of the archive filename may be either `.zip` or `.jar`. Archived `.zip` files are in the familiar ZIP archive format. Archived `.jar` files are in the Java archive format. Archived `.jar` files support compression and advanced features like digital signatures.

You can use the `archive` attribute with any `<applet>` tag, even if the class referenced by the tag's `code` attribute does not exist in the archive. If the class is not found in the archive, the browser simply attempts to retrieve the class relative to the document URL or the `codebase` URL, if specified.

### 12.2.3.5 The code and codebase attributes

The `code` attribute is required. Use `code` to specify the filename, *not* the URL, of the Java class to be executed by the browser. Like `<object>`, make the search relative to another storage location by using the `codebase` attribute described in [Section 12.2.1.2](#) or an archive, as described in [Section 12.2.3.4](#). The extension suffix of the filename should be `.class`. If you don't include the suffix, some browsers append `.class` automatically when searching for the applet.

Here is our clock example from earlier rewritten as an `<applet>`:

```
<applet code="clock.class" codebase="http://www.kumquat.com">
</applet>
```

which the browser retrieves and displays from:

```
http://www.kumquat.com/classes/clock.class
```

### 12.2.3.6 The name attribute

The `name` attribute lets you supply a unique name for this instance of the code class the copy of the applet that runs on the individual user's computer. As with other named elements in your document, providing a

name for the applet lets other parts of your document, including other applets, reference and interact with this one (e.g., for sharing computed results).

### **12.2.3.7 The height, hspace, vspace, and width attributes**

Use the `height` and `width` attributes (identical to the counterparts for the `<img>` and `<object>` tags) to define the size of the applet's display region in the document. Use `hspace` and `vspace` to interpose some empty space around the applet region and thereby set it off from the text. They all accept values indicating the size of the region in pixels. [[Section 5.2.6.10](#)]

### **12.2.3.8 The mayscript attribute**

The `mayscript` attribute, supported only by Netscape, indicates that the Java applet is accessing JavaScript features within the browser. Normally, Java applets attempting to access JavaScript cause a browser error. If your applets access JavaScript, you must specify `mayscript` in the `<applet>` tag.

### **12.2.3.9 The title attribute**

The value of this attribute is a quoted string that provides a title, if necessary, for the applet.

### **12.2.3.10 The object attribute**

This unfortunately named attribute and its string value reference the name of the resource that contains a serialized version of the applet. How and what it does is an enigma; none of the popular browsers support it.

### 12.2.4 The <embed> Tag (Extension)

Use the `<embed>` tag to include a reference in your document to some special plug-in application and perhaps data for that application. The standard analog for `<embed>` is the `<object>` tag with the `data` attribute.



## <embed> □ □

### *Function:*

Embeds an object in a document

### *Attributes:*

`align (□, □), border (□), height (□, □), hidden (□, □),  
hspace (□), name (□, □), palette (□, □), pluginspage (□),  
src (□, □), type (□), units □, □, vspace (□), width (□, □)`

### *End tag:*

None

### *Contains:*

Nothing

### *Used in:*

*text*

With `<embed>`, reference the data object via the `src` attribute and URL value for download by the browser. The browser uses the MIME type of the `src`'d object to determine the plug-in required to process the object. Alternatively, you may also use the `type` attribute to specify a MIME type without an object and thereby initiate execution of a plug-in application, if it exists on the user's computer.

Like all other tags, the nonstandard `<embed>` tag extension has a set of predefined attributes that define parameters and modify the tag's behavior. Unlike most other tags, however, the browsers let you include plug-in-specific name/value attribute pairs in `<embed>` that, instead of altering the action of the tag itself, get passed to the plug-in application for further processing.

For example, this tag:

```
<embed src=movie.avi width=320 height=200 autostart=tr
```

has attributes that are processed by the `<embed>` tag (`src`, `width`, and `height`) and two that are not recognized but rather are passed to the plug-in associated with AVI video clips: `autostart` and `loop`.<sup>[5]</sup>

[5] Internet Explorer has built-in support for AVI movies; Netscape requires a plug-in it finds on the Internet and installs automatically, if the user lets it.

It is not possible to document all the possible attributes that the many different plug-ins might need with their associated `<embed>` tags. Instead, you must turn to the plug-in developer to learn about all of their required and optional attributes for each plug-in that you plan to use in your pages.

#### 12.2.4.1 The `align`, `border`, `height`, `hspace`, `vspace`, and `width` attributes

The browser displays embedded objects to the user in a region set aside within the document window. The `<embed>` tag's `align`, `border`, `height`, `width`, `hspace`, and `vspace` attributes let you control the appearance of that region exactly as they do for the `<img>` tag, so we won't belabor them. [Section 5.2.6]

Briefly, the `height` and `width` attributes control the size of the viewing region. Normally, you should specify the height and width in pixels, but you may use some other units of measure if you also specify the `units` attribute (see Section 12.2.4.8). The `hspace` and `vspace` attributes define a margin, in pixels, around the viewing region. The `align` attribute determines how the browser aligns the region within surrounding text, while the `border` attribute determines the width of the border, if any, surrounding the viewing region.

Internet Explorer and Netscape support the `height`, `width`, and `align`

attributes; only Netscape supports `border`, `hspace`, and `vspace` for the `<embed>` tag.

#### 12.2.4.2 The hidden attribute

The `hidden` attribute makes an object invisible to the user, forcing it to have a height and width of 0. Note that setting `hidden` does not cause the browser to display an empty region within the document, but rather completely removes the object from the containing text flow.

This attribute is useful for audio streams placed within documents. The HTML entry:

```
<embed src=music.wav hidden autostart=true loop=true>
```

embeds an audio object in the page. The browser does not show anything to the user, but rather plays background music for the page. By contrast, the plug-in associated with:

```
<embed src=music.wav>
```

might present an audio control panel to users so that they can start and stop the audio playback, adjust the volume, and so forth.

#### 12.2.4.3 The name attribute

Like other `name` attributes, this one lets you label the embedded object for later reference by other elements in your document, including other objects. The value of the `name` attribute is a character string.

#### 12.2.4.4 The palette attribute

The `palette` attribute is supported by both Netscape and Internet Explorer, but in completely different ways. With Netscape, the value of the `palette` attribute is either `foreground` or `background`, indicating

which palette of window system colors the plug-in uses for its display.

With Internet Explorer, the value of `palette` is a pair of hexadecimal color values, separated by a vertical bar. The first value determines the foreground color used by the plug-in; the second sets the background color. Thus, specifying this `palette`:

```
palette=#ff0000|#00ff00
```

causes the plug-in to use red as its foreground color and green as its background color. For a complete description of hexadecimal color values, see [Appendix G](#).

#### 12.2.4.5 The `pluginspage` attribute

The `pluginspage` attribute, supported only by Netscape, specifies the URL of a web page that provides instruction on where to obtain and how to install the plug-in associated with the embedded object.

#### 12.2.4.6 The `src` attribute

Like its document-referencing counterparts for myriad other tags, the `src` attribute supplies the URL of the data object that you embed in the HTML document. The server providing the object must be configured so that it notifies the browser of the correct MIME type of the object. If not, the browser uses the suffix of the last element of the `src` value—the object's filename in the URL path—to determine the type of the object. The browser uses this MIME type to determine which plug-in it executes to process the object.

If you don't include a `src` attribute with the `<embed>` tag, you must include a `type` attribute to explicitly reference the MIME type and, as a result, the plug-in application.

#### 12.2.4.7 The `type` attribute

Use the `type` attribute in addition to or in lieu of the `src` attribute. Its value explicitly indicates the MIME type of the embedded object, which in turn determines which plug-in the browser invokes to process the object. This attribute is not required if you include the `src` attribute and the browser can determine the object type from the object's URL or server. You must supply a `type` attribute if you don't include the `src` attribute.

It may seem odd to use an `<embed>` tag without a `src` attribute reference to some object, but this is common if the plug-in requires no data or retrieves its data dynamically after it is started. In these cases, the `type` attribute is required so that the browser knows which plug-in to invoke.

#### 12.2.4.8 The units attribute

Pixels are the default unit of measure for the `height` and `width` attributes that control the `<embed>` display space. The `units` attribute lets you explicitly state that the absolute measure is `pixels`, or change it to the relative `en`, which is one-half the current point size of text in the document. With the `en` units, you tailor the object's viewing area (*viewport*) to be proportional to its immediately surrounding content, the size of which is varied by the user.

For example, this tag creates a viewport of 200 x 320 pixels:

```
<embed src=movie.avi height=200 width=320 units=pixels>
```

By changing `units` to `en`, that same viewport, when included within a flow of 12-point text, becomes 1200 x 1920 pixels.

#### 12.2.5 The `<noembed>` Tag (Extension)

Some browsers do not support the `<embed>` tag. The `<noembed>` tag makes it easy to supply alternative content that tells users what they are missing.

## **<noembed>** ☐ ☐

### *Function:*

Supplies content to `<embed>`-incompatible browsers

### *Attributes:*

None

### *End tag:*

`</noembed>`; never omitted

### *Contains:*

Nothing

### *Used in:*

*text*

The popular browsers ignore the contents of the `<noembed>` tag, whereas browsers that do not support the `<embed>` tag display the contents of the `<noembed>` tag. Normally, use the contents of the `<noembed>` tag to display some sort of message placating users of inadequate browsers:

```
<embed src=cool.mov autostart=true loop=true>
```

```
<noembed>To view the cool movie, you need to upgrade to  
a browser that supports the <embed> tag!</noembed>
```

We recommend using a `<noembed>` message only in those cases where the object is crucial for the user to comprehend and use your document. And, in those cases, provide a link to a document that can stand alone without the embedded object, or nicely explain the difficulty.

## 11.3 Frame Layout

Frame layout is similar to table layout. Using the `<frameset>` tag, you can arrange frames into rows and columns while defining their relative or absolute sizes.

### 11.3.1 The `<frameset>` Tag

Use the `<frameset>` tag to define a collection of frames and other framesets and control their spacing and borders. Framesets also may be nested, allowing for a richer set of layout capabilities.

## <frameset>

### *Function:*

Defines a collection of frames

### *Attributes:*

`border` (`[]`), `bordercolor` (`[]`), `class`, `cols`,  
`frameborder` (`[]`), `framespacing` (`[]`), `id`, `onLoad`,  
`onUnload`, `style`, `title`

### *End tag:*

`</frameset>`; never omitted

### *Contains:*

*frameset\_content*

### *Used in:*

*html\_content*

Use the `<frameset>` tag in lieu of a `<body>` tag in the frame document. You may not include any other content except valid `<head>` and `<frameset>` content in a frame document. Combining frames with a conventional document containing a `<body>` section may result in unpredictable browser behavior.

### 11.3.1.1 The rows and cols attributes

The `<frameset>` tag has one required attribute: either `cols` or `rows` your choice. They define the size and number of columns or rows of either frames or nested framesets for the document window. Both attributes accept a quote-enclosed, comma-separated list of values that specifies either the absolute (pixels) or relative (percentage or remaining



space) width (for columns) or height (for rows) for the frames. The number of attribute values determines how many rows or columns of frames the browser displays in the document window.

As with tables, the browser matches the size you give a frameset as closely as possible. The browser does not, however, extend the boundaries of the main document window to accommodate framesets that would otherwise exceed those boundaries or fill the window with empty space if the specified frames don't fill the window. Rather, browsers allocate space to a particular frame relative to all other frames in the row and column and resolutely fill the entire document window. (Did you notice that the main frame window does not have scrollbars?)

For example:

```
<frameset rows="150,300,150">
```

creates three rows of frames, each extending across the entire document window. The first and last frames are set to 150 pixels tall, and the second is set to 300 pixels. In reality, unless the browser window is exactly 600 pixels tall, the browser automatically and proportionately stretches or compresses the first and last frames so that each occupies one quarter of the window space. The center row occupies the remaining half of the window space.

Frame row- and column-size values expressed as percentages of the window dimensions are more sensible. For instance, the following example is effectively identical to the previous one:

```
<frameset rows="25%,50%,25%">
```

Of course, if the percentages don't add up to 100%, the browser automatically and proportionally resizes each row to make up the difference.

If you are like us, making things add up is not a strength. Perhaps some of the frame designers suffer the same difficulty, which would explain why they included the very nifty asterisk option for `<frameset> rows` and

`cols` values. It tells the browser to size the respective column or row to whatever space is left over after putting adjacent frames into the frameset.

For example, when the browser encounters the following frame tag:

```
<frameset cols="100,*">
```

it makes a fixed-sized column 100 pixels wide and then creates another frame column that occupies all of the remaining space in the frameset.

Here's a fancier layout example:

```
<frameset cols="10,*,10">
```

This one creates two very thin columns down the edges of the frameset and gives the remaining center portion to the middle column.

You may also use the asterisk for more than one row- or column-attribute value. In that case, the corresponding rows or columns equally divide the available space. For example:

```
<frameset rows="*,100,*">
```

creates a 100-pixel tall row in the middle of the frameset and equal-sized rows above and below it.

If you precede the asterisk with an integer value, the corresponding row or column gets proportionally more of the available space. For example:

```
<frameset cols="10%,3*,*,*">
```

creates four columns: the first column occupies 10% of the overall width of the frameset. The browser then gives the second frame three-fifths of the remaining space, and the third and the fourth are each given one-fifth of the remaining space.

Using asterisks (especially with the numeric prefix) makes it easy to divide up the remaining space in a frameset.

Be aware, too, that unless you explicitly tell it not to, the browser lets users manually resize the individual frame document's columns and rows and hence change the relative proportions each frame occupies in the frames display. To prevent this, use the `noresize` attribute for the `<frame>` tag, which we describe later. [[<frame>](#)]

### 11.3.1.2 Controlling frame borders and spacing

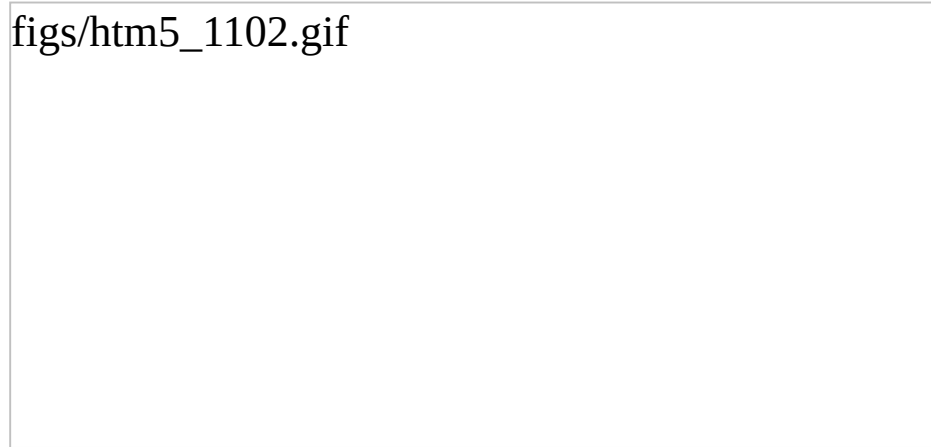
The popular browsers provide attribute extensions that you may use to generally define and change the borders surrounding the frames in a frameset. The HTML 4 and XHTML standards prefer instead that you include these border-related display features via `<style>` tag attributes.

Both Internet Explorer and Netscape accept the `frameborder` attribute to disable or explicitly enable frame borders. (By default, every frame in a frameset as well as the frameset window itself is rendered with a 3D border; see [Figure 11-1](#).) The two browsers' documentations disagree about the particular values for the `frameborder` attribute, but both acknowledge the other's conventions. Hence, setting the value of `frameborder` to `0` or `no` turns off borders (see [Figure 11-2](#)); `1` or `yes` turns on borders.

**Figure 11-1. A simple six-panel frame layout**



**Figure 11-2. The frameborder attribute lets you remove the borders between frames**



figs/htm5\_1102.gif

Internet Explorer and Netscape do disagree, however, as to how you may control the thickness of the borders. Internet Explorer Versions 5 and later support both the `framespacing` and `border` attributes, whose values are the number of pixels you want between frames.

These attributes affect all frames and framesets nested within the current frameset as displayed by Internet Explorer. In practice, you should set it once on the outermost `<frameset>` to create a consistent border appearance for all of the frames in a single page.

Netscape accepts only the `border` attribute to define the border width, with an integer value in pixels. Like Internet Explorer, Netscape lets you include the `frameborder` attribute with any `<frameset>` tag, affecting all nested frames and framesets. Unlike Internet Explorer, Netscape lets you include the `border` attribute only in the outermost `<frameset>`, ensuring that all frame borders are the same width within that `<frameset>`.

Since browsers ignore unsupported attributes, it is possible to define frame borders so that both browsers do the right thing. Just make sure to use the same `framespacing` and `border` values.

Finally, both Netscape and Internet Explorer Versions 5 and 6 let you

control the color of the frame borders using the `bordercolor` attribute (Figure 11-3). It accepts a color name or hexadecimal triple as its value. For example, although you can't see their color, the borders in Figure 11-3 are light green, corresponding to the RGB value of "00CC00." You can find a complete list of color names and values in [Appendix G](#).

**Figure 11-3. Netscape accepts border and bordercolor attributes to control the color and spacing between frames**



### 11.3.1.3 Frames and JavaScript

Internet Explorer and Netscape support JavaScript-related event handlers that let your frame documents react when they are first loaded and when the frame window gets resized (`onLoad`); when they are unloaded from the browser by the user (`onUnload`); when the window containing the frameset loses focus, such as when the user selects another window (`onBlur`); or when the frameset becomes the active window (`onFocus`). Included as `<frameset>` attributes, these event handlers take quote-enclosed lists of JavaScript commands and function calls as their values. For example, you might notify the user when all the contents have been loaded into their respective frames of a lengthy frameset:

```
<frameset onLoad="window.alert('Everything is loaded.
```

These four attributes may also be used with the `<body>` tag. We cover JavaScript event handlers in more detail in [Section 12.3.3](#).

#### 11.3.1.4 Other `<frameset>` attributes

Like most of the other standard tags, the `<frameset>` tag honors four of the standard attributes: `class`, `style`, `title`, and `id`.

Use the `class` attribute to associate a predefined style class with this frame and, via style inheritance, its content. Alternatively, use the `style` attribute to define a style inline with the `<frameset>` tag. We cover styles more completely in [Chapter 8](#).

The `id` attribute creates a unique identifier for the frame, and the `title` attribute creates a title for the frame that might be presented to the user or used by a nonvisual browser. [[Section 4.1.1.4](#)] [[Section 4.1.1.4](#)]

### 11.3.2 Nesting `<frameset>` Tags

You can create some elaborate browser displays with a single `<frameset>`, but the frame layout is unimaginative. Instead, create staggered frames and other more complex layouts with multiple `<frameset>` tags nested within a top-level `<frameset>` in the frame document.

For example, create a layout of two columns, the first with two rows and the second with three rows (as shown in [Figure 11-4](#)), by nesting two `<frameset>` tags with row specifications within a top-level `<frameset>` that specifies the columns:

```
<frameset cols="50%,*">

  <frameset rows="50%,*">

    <frame src="frame1.html">
```

```
    <frame src="frame2.html">

</frameset>

<frameset rows ="33%,33%,*">

    <frame src="frame3.html">


    <frame src="frame4.html">

    <frame src="frame5.html">

</frameset>

</frameset>
```

**Figure 11-4. Staggered frame layouts use nested <frameset> tags**



figs/htm5\_1104.gif

## Chapter 17. Tips, Tricks, and Hacks

We've sprinkled a number of tips, tricks, and hacks throughout this book, along with style guidelines, examples, and instructions. So why have a special chapter on tips, tricks, and hacks? Because it's where many readers will turn when they pick up this book for the first time. HTML and XHTML are the languages, albeit constrained, that make the Web the exciting place that it is. And interested readers want to know, "How do I do the cool stuff ?"



## 5.6 Other Multimedia Content

The Web is completely open-minded about the types of content that can be exchanged by servers and browsers. In this section, we look at a different way to reference images, along with audio, video, and other document formats.

### 5.6.1 Embedded Versus Referenced Content

Images currently enjoy a special status among the various media that can be included within an HTML or XHTML document and displayed inline with other content by all but a few browsers. Sometimes, however, as we discussed earlier in this chapter, you may also reference images externally—particularly large ones in which details are important but not immediately necessary to the document content. Other multimedia elements, including digital audio and video, can be referenced as separate documents external to the current one.

You normally use the anchor tag (`<a>`) to link external multimedia elements to the current document. Just like other link elements selected by the user, the browser downloads the multimedia object and presents it to the user, possibly with the assistance of an external application or plug-in. Referenced content is always a two-step process: present the document that links to the desired multimedia object, then present the object if the user selects the link. [`<a>`]

In the case of images, you can choose how to present images to the user: inline and immediately available via the `<img>` tag, or referenced and subsequently available via the `<a>` tag. If your images are small and critical to the current document, you should provide them inline. If they are large or are only a secondary element of the current document, make them available as referenced content via the `<a>` tag.

If you choose to provide images via the `<a>` tag, it is sometimes a courtesy to your readers to indicate the size of the referenced image in

the referencing document and perhaps provide a thumbnail sketch. Users can then determine whether it is worth their time and expense to retrieve it.

### 5.6.2 Referencing Audio, Video, and Images

You reference any external document, regardless of type or format, via a conventional anchor (`<a>`) link:

```
The <a href="sounds/anthem.au">Kumquat Grower's Anthem  
the thousands of 'quat growers around the world.
```

Just like any referenced document, the server delivers the desired multimedia object to the browser when the user selects the link. If the browser finds that the document is not HTML or XHTML but rather some other format, it automatically invokes an appropriate rendering tool to display or otherwise convey the contents of the object to the user.

You can configure your browser with special helper applications that handle different document formats in different ways. Audio files, for example, might be passed to an audio-processing tool, while video files are given to a video-playing tool. If a browser has not been configured to handle a particular document format, the browser will inform you and offer to simply save the document to disk. You can later use an appropriate viewing tool to examine the document.

Browsers identify and specially handle multimedia files from one of two different hints: either from the file's Multipurpose Internet Mail Extension (MIME) type, provided by the server, or from a special suffix in the file's name. The browser prefers MIME because of its richer description of the file and its contents, but it will infer the file's contents (type and format) from the file suffix: `.gif` or `.jpg`, for GIF and JPEG encoded images, for example, or `.au` for a special sound file.

Since not all browsers look for a MIME type or are necessarily correctly configured with helper applications by their users, you should always use

the correct file suffix in the names of multimedia objects. See [Table 5-1](#) for more information.

### **5.6.3 Appropriate Linking Styles**

Creating effective links to external multimedia documents is critical. The user needs some indication of what the object is and perhaps the kind of application the linked object needs to execute. Moreover, most multimedia objects are quite large, so common courtesy tells us to provide users with some indication of the time and expense involved in downloading them.

In lieu of, or in addition to, the anchor and surrounding text, a small thumbnail of a large image, or a familiar icon that indicates the referenced object's format, is useful.

### **5.6.4 Embedding Other Document Types**

The Web can deliver nearly any type of electronic document, not just graphics, sound, and video files. To display them, however, the client browser needs a helper application installed and referenced. Recent browsers also support plug-in accessory software and, as described in [Chapter 12](#), may extend the browser for some special function, including inline display of multimedia objects.

For example, consider a company whose extensive product documentation was prepared and stored in some popular layout application such as Adobe Acrobat, FrameMaker, Quark XPress, or PageMaker. The Web offers an excellent way for distributing that documentation over a worldwide network, but converting to HTML or XHTML would be too costly at this time.

The solution is to prepare a few HTML or XHTML documents that catalog and link the alternative files and invoke the appropriate display applet. Or, make sure that the users' browsers have the plug-in software or are configured to invoke the appropriate helper application. Adobe's Acrobat

Reader is a very popular plug-in, for example. If the document is in Acrobat (*.pdf*) format, if a link to an Acrobat document is chosen, the tool is started and accordingly displays the document, often right in the browser's window.

## Chapter 10. Tables

Of all the extensions that found their way into HTML and XHTML, none is more welcome than tables. While tables are useful for the general display of tabular data, they also serve an important role in managing document layout. Creative use of tables, as we'll show in this chapter, can go a long way to enliven an otherwise dull document layout. And you may apply all the CSS styles to the various elements of a table to achieve a desktop-published look and feel.

## 13.3 Server -Push Documents

Server-push dynamic documents are driven from the server side. The client/server connection remains open after an initial transfer of data, and the server periodically sends new data to the client, updating the document's display. Server-push is made possible by some special programming on the server side and is enabled by the multipart/mixed-media type feature of Multipurpose Internet Mail Extensions (MIME), the computer industry's standard for multimedia document transmission over the Internet.

Server-push documents currently are not supported by Internet Explorer.

### 13.3.1 The Multipart/Mixed Media Type

As we mentioned earlier in this chapter, in the discussion of client-pull dynamic documents, the HTTP server sends a two-part transmission to the client browser: a header describing the document, followed by the document itself. The document's MIME type is part of the HTTP header field. Normally, the server includes "Content-Type: text/html" in an HTML document's header before sending its actual contents. By changing that content type to "multipart/mixed," you can send an HTML document or several documents in several pieces, rather than in a single chunk. Only Netscape, though, understands and responds to the multipart header field; other browsers either ignore additional parts or refuse the document altogether.

The general form of the MIME multipart/mixed-media `Content-Type` header looks like this:

```
Content-type: multipart/mixed;boundary="SomeRandomStri:
```

This HTTP header component tells the Netscape client to expect the document to follow in several parts and to look for `SomeRandomString`, which separates the parts. That boundary string should be unique and

should not appear anywhere in any of the individual parts. The content of the server-to-client transmission looks like this:

```
--SomeRandomString  
  
Content-type: text/plain
```

Data for the first part

```
--SomeRandomString  
  
Content-type: text/plain
```

Data for the second part

```
--SomeRandomString--
```

The above example has two document parts, both plain text. The server sends each part, preceded by our `SomeRandomString` document-boundary delimiter (which, in turn, is preceded by two dashes), followed by the `Content-Type` field and then the data for each part. The last transmission from server to client is a single reference to the boundary string, followed by two more dashes indicating that this was the last part of the document.

Upon receipt of each part, the Netscape browser automatically adds the incoming data to the current document display.

You have to write a special HTTP server application to enable this type of server-push dynamic document one that creates the special HTTP MIME multipart/mixed header and sends the various documents separated by the boundary delimiter.

### 13.3.2 The Multipart/X-Mixed-Replace Media Type

Server-push dynamic document authors may use an experimental variant of the MIME multipart/mixed media type known as *multipart/x-mixed-replace media*. The difference between this special content type and its predecessor is that, rather than simply adding content to the current display, the "replace" version has each subsequent part replace the preceding one.

The format of the mixed-replace HTTP header is very similar to its multipart/mixed counterpart; the only difference is in the `Content-Type`:

```
multipart/x-mixed-replace;boundary=SomeRandomString
```

All other rules regarding the format of the multipart content are the same, including the boundary string used to separate the parts and the individual `Content-Type` fields for each part of the content.

### 13.3.3 Exploiting Multipart Documents

It is easy to see how you can use the two special MIME multipart content types to create server-push dynamic documents. By delaying the time between parts, you might create an automatically scrolling message in the Netscape browser window. Or by replacing portions of the document through the x-mixed-replace MIME type, you might include a dynamic billboard in your document, or perhaps even animation.

Note that server-push multipart documents need not apply only to HTML or other plain-text documents. Images, too, are a MIME-encoded content type, so you can have the HTTP server transmit several images in sequence as parts of a multipart transmission. Since you may also have each new image replace the previous one, the result is crude animation. Done correctly, over a network of sufficient bandwidth, the effect can be quite satisfying.



### 13.3.3.1 Efficiency considerations

Server-push documents keep a connection open between the client and server for the duration of the dynamic document's activity. For some servers, this may consume extra network resources and may also require that several processes remain active, servicing the open connection. Make sure the server-push process (and, hence, the client/server connection) expires upon completion or after some idle period. Otherwise, someone will inadvertently camp on an endlessly cycling server-push document and choke off other users' access to the server.

Before choosing to implement server-push documents, make sure that your server can support the added processing and networking load. Keep in mind that many simultaneous server-push documents may be active, multiplying the impact on the server and seriously affecting overall server performance.

### 13.3.4 Creating a Server-Push Document

Create a special application that runs with the HTTP server to enable server-push dynamic documents. The application must create the special MIME `Content-Type` header field that notifies the Netscape browser that the following document comes in several parts added to or replacing a portion of the current document. The application must create the appropriate boundary delimiter and send the `Content-Type` header and data for each part, perhaps also delaying transmission of each part by some period of time. Consult your server's documentation to learn how to create a server-side application that can be invoked by accessing a specific URL on the server. With some servers, this may be as simple as placing the application in a certain directory on the server. With others, you may have to bend over backward and howl at the moon on certain days.

#### 13.3.4.1 Server-push example application for NCSA and Apache httpd

The NCSA and Apache *httpd* servers run on most Unix and Linux systems. Administrators usually configure the servers to run server-side applications stored in a directory named *cgi-bin*.

The following is a simple shell script that illustrates how to send a multipart document to a Netscape client via NCSA or Apache *httpd*:<sup>[3]</sup>

[3] It is an idiosyncrasy of NCSA *httpd* that no spaces are allowed in the `Content-Type` field that precedes your multipart document. Some authors like to place a space after the semicolon and before the boundary keyword. Don't do this with NCSA *httpd*; run the whole `Content-Type` together without spaces to get the server to recognize the correct multipart content type.

```
#!/bin/sh

#

# Let the client know we are sending a multipart document
# with a boundary string of "NEXT"

#

echo "HTTP/1.0 200"

echo "Content-type: multipart/x-mixed-replace;boundary="
echo ""

echo "--NEXT"

while true
do

#

# Send the next part, followed by a boundary string
```

```
# Then sleep five seconds before repeating
#
echo "Content-type: text/html"
echo ""
echo "<html>"
echo "<head>"
echo "<title>Processes On This Server</title>"
echo "</head>"
echo "<body>"
echo "<h3> Processes On This Server</h3>"
echo "Date:"
date
echo "<p>"
echo "<pre>"
ps -el
echo "</pre>"
echo "</body>"
echo "</html>"
```

```
echo "--NEXT"
```

```
sleep 5
```

```
done
```

In a nutshell, this example script updates a list of the processes running on the server machine every five seconds. The update continues until the browser breaks the connection by moving on to another document.

We offer this shell script example to illustrate the basic logic behind any server-push document generator. In reality, you should try to create your server-side applications using a more conventional programming language, such as Perl or C. These applications will run more efficiently and can better detect when the client has severed the connection to the server.

## 11.4 Frame Contents

A frame document contains no displayable content, except perhaps a message for non-frames-enabled browsers. Instead, `<frame>` tags inside the one or more `<frameset>` tags (which encapsulate the contents of a frame document) provide URL references to the individual documents that occupy each frame. [`<noframes>`]

### 11.4.1 The `<frame>` Tag

The `<frame>` tag appears only within a `<frameset>`. Use it to set, via its associated `src` attribute, the URL of the document content that initially gets displayed inside the respective frame.

## **<frame>**

### *Function:*

Defines a single frame in a `<frameset>`

### *Attributes:*

`bordercolor` (`□`, `□`), `class`, `frameborder`, `id`, `longdesc`,  
`marginheight`, `marginwidth`, `name`, `noresize`, `scrolling`,  
`src`, `style`, `title`

### *End tag:*

`</frame>`; rarely included in HTML

### *Contains:*

Nothing

### *Used in:*

*frameset\_content*

Browsers place the frame contents into the frameset column by column, from left to right, and then row by row, from top to bottom. Accordingly, the sequence and number of `<frame>` tags inside the `<frameset>` tag are important.

The browser displays empty frames for `<frame>` tags that do not have `src` attributes. It also displays empty frames if the `<frameset>` tag calls for more frames than the corresponding `<frame>` tags define; for instance, if your frame document calls for three columns and you provide only five frames. Orphan frames remain empty, and you cannot put content into them later, even if they have a target `name` or `id` for display redirection. [[Section 11.4.1.2](#)]

### 11.4.1.1 The `src` attribute

The value of the `src` attribute for the `<frame>` tag is the URL of the document that is to be displayed in the frame. There is no other way to provide content for a frame. You shouldn't, for instance, include any `<body>` content within the frame document; the browser ignores the frame tags and displays just the contents of a `<body>` tag if it comes first, or vice versa.

The document referenced by the `src` attribute may be any valid document or any displayable object, including images and multimedia. In particular, the referenced document may itself be composed of one or more frames. The frames are displayed within the referencing frame, providing yet another way of achieving complex layouts using nested frames.

Since the source may be a complete document, all the features of HTML/XHTML apply within a frame, including backgrounds and colors, tables, fonts, and the like. Unfortunately, this also means that multiple frames in a single browser window may conflict with each other. Specifically, if each nested frame document (not a regular HTML or XHTML document) has a different `<title>` tag, the title of the overall browser window is the title of the most recently loaded frame document. The easiest way to avoid this problem is to ensure that all related frame documents use the same title.

### 11.4.1.2 The `name` and `id` attributes

The optional `name` attribute for the `<frame>` tag labels that frame for later reference by a `target` attribute for the hypertext link anchor (`<a>`) tag and the `<form>` tag. This way, you can alter the contents of a frame using a link in another frame. Otherwise, like normal browser windows, hypertext-linked documents replace the contents of the source frame. We discuss names and targets at greater length later in this chapter. [[Section 11.7.1](#)]

Similarly, the `id` attribute uniquely identifies a frame, but the browsers do not support its use for target redirection, even though they do support `id`'s use as a hyperlink target in many other HTML and XHTML tags.

The value of the `name` or `id` attribute is a text string enclosed in quotation marks.

#### 11.4.1.3 The `noresize` attribute

Even though you may explicitly set frame dimensions with attributes in the `<frameset>` tag, users can manually alter the size of a column or row of frames. To suppress this behavior, add the `noresize` attribute to the frame tags in the row or column whose relative dimensions you do not want users fiddling with. For example, for a two-by-two frame document, a `noresize` attribute in any one of the four associated frame tags effectively freezes the relative proportions of all the frames.

The `noresize` attribute is especially useful for frames that contain fixed images serving as advertisements, a button bar, or a logo. By fixing the size of the frame to contain just the image and setting the `noresize` attribute, you guarantee that the image is displayed in the intended manner and that the remainder of the browser window is always given over to the other frames in the document.

#### 11.4.1.4 The `scrolling` attribute

The browser displays vertical and horizontal scrollbars with frames whose contents are larger than the allotted window space. If there is sufficient room for the content, the scrollbars disappear. The `scrolling` attribute for the `<frame>` tag gives you explicit control over whether the scrollbars appear or disappear.

With `scrolling="yes"`, Internet Explorer, but not Netscape, adds scrollbars to the designated frame even if there is nothing to scroll. If you set the `scrolling` attribute value to `no`, scrollbars are never added to



the frame, even if the frame contents are larger than the frame itself. The value `auto`, supported only by Netscape, works as if you didn't include the `scrolling` attribute in the tag; Netscape adds scrollbars only as needed. To achieve `auto` behavior in Internet Explorer, simply omit the `scrolling` attribute altogether.

#### **11.4.1.5 The `marginheight` and `marginwidth` attributes**

The browser normally places a small amount of space between the edge of a frame and its contents. You can change those margins with the `marginheight` and `marginwidth` attributes, each including a value for the exact number of pixels to place around the frame's contents.

You cannot make a margin less than 1 pixel or make it so large that there is no room for the frame's contents. That's because, like most other HTML attributes, these advise they do not dictate to the browser. If your desired margin values cannot be accommodated, the browser ignores them and renders the frame as best it can.

#### **11.4.1.6 The `frameborder` and `bordercolor` attributes**

You can add or remove borders from a single frame with the `frameborder` attribute. Values of `yes` or `1` and `no` or `0` respectively enable or disable borders for the frame and override the value of the `frameborder` attribute for any frameset containing the frame.

Note that the browsers do react somewhat differently to border specifications. Netscape, for instance, removes an individual border only if adjacent frames sharing that border have borders turned off. Internet Explorer, on the other hand, removes those adjacent borders, but only if they are not explicitly turned on in those adjacent frames. Our advice is to explicitly control the borders for each frame if you want to consistently control the borders for all frames across both browsers.

With the popular browsers, you also can change the color of the

individual frame's borders with the `bordercolor` attribute. Use a color name or hexadecimal triple as its value. If two adjacent frames have different `bordercolor` attributes, the resulting border color is undefined. You can find a complete list of color names and values in [Appendix G](#).

#### **11.4.1.7 The title and longdesc attributes**

Like most other standard tags, you can provide a title for a frame with the `title` attribute. The value of the attribute is a quote-enclosed string that describes the contents of the frame. Browsers might display the title, for instance, when the mouse passes over the frame.

If the `title` attribute isn't quite enough for you, the `longdesc` attribute can be used. Its value is the URL of a document that describes the frame. Presumably, this long description might be in some alternative media, suitable for use by a nonvisual browser.

[**SYMBOL**] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R]  
[S] [T] [U] [V] [W] [X] [Z]

"unvisited" link state

<strike> tags

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

`<a>` tags 2nd 3rd [See also [hyperlinks](#)]

- hyperlink states

- linking external documents

- nesting restrictions

abbr attribute (`<th>` and `<td>`)

`<abbr>` tags

above attribute (`<layer>`)

absbottom, absmiddle values 2nd [See also [alignment](#)]

absolute font size

absolute URLs 2nd

- base and relative URLs, combining

accept attribute (`<input type=file>`)

accept-charset attribute (`<form>`)

**accesskey attribute**

- `<a>`

- `<legend>`

- form controls

`<acronym>` tags

**action attribute**

- `<form>`

- `<isindex>`

action buttons (forms)

`\:active` pseudoclass

ActiveX technology

`<address>` tags

addresses

- IP addresses 2nd

- XML DTD, defining for

adjacent selectors

Advanced Research Projects Agency (ARPA)

\:after pseudoelement 2nd

align attribute

<applet>

<caption>

<div>

<embed>

<h#>

<hr>

<iframe>

<img> 2nd

<input type=image>

<legend>

<marquee>

<object>

<p>

<spacer>

<table>

<th> and <td>

<tr>

alignment [See also align attribute; format; page layout]

<center> tags

form elements 2nd

frames

headings

horizontal rules

image buttons (forms)

images 2nd 3rd

layers 2nd

sections

tables

captions

cell contents 2nd

rows

text 2nd

in marquees

paragraphs

whitespace blocks

alink attribute (<body>)

all value (style media)

## **alt attribute**

<applet>

<area>

<img>

alternate value (marquee behavior)

## **ampersand (&)**

entities 2nd 3rd

in URLs

in XHTML

anchors [See also <a> tags; hyperlinks]2nd

## **animation**

frame-by-frame (GIF)

of text

annotated lists [See definition lists]

anonymous FTP

<applet> tags

applets 2nd 3rd

application/x-www-form-urlencoded encoding

## **archive attribute**

<applet>

<object>

<area> tags 2nd 3rd

ARPA (Advanced Research Projects Agency)

articles (newsgroups), identifiers for

ASCII text file format, saving HTML/XHTML documents in

## **at-rules**

@font-face

@import

@media

@page

attribute selectors

supporting browsers

## attributes

for <body> tags

deprecated [See deprecated attributes]

HTML tags

core attributes

images, enabling JavaScript manipulation

## XHTML

case sensitivity in

quoted values

values for valueless HTML attributes

XML

declaring in DTD

required and default

values for

audience, designing for

audio [See also multimedia]<sup>2nd 3rd</sup>

<bgsound> tags

client-pull feature for

mixing

properties

aural value (style media)

authoring tools for XHTML pages

automation, document

AVI movies <sup>2nd</sup> [See also video]

axis attribute (<th> and <td>)

azimuth property

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

`<b>` tags 2nd

background [[See also transparent GIFs](#)]

- audio

- colors 2nd 3rd

- layers

- marquees

- in tables 2nd

- troubleshooting

- images 2nd 3rd

- behind specific elements

- layers

- placement of

- in tables 2nd

- tiling whole background

- troubleshooting

- layers [[See layers](#)]

- style properties for

background-attachment property 2nd

background-color property 2nd

background-image property 2nd

**background attribute**

- `<body>` 2nd

- `<layer>`

- `<table>`

- `<th>` and `<td>`

background property 2nd

background-position property 2nd

background-repeat property 2nd

backslash (\\), CSS2 escape characters and

`<base>` tags 2nd 3rd



base URLs 2nd

<basefont> tags

baseline descriptor

baseline value 2nd [See also alignment]

bbox descriptor

<bdo> tags

\:before pseudoelement 2nd

behavior attribute (<marquee>)

"being visited" link state

below attribute (<layer>)

**bgcolor attribute**

<body> 2nd

<layer>

<marquee>

<table>

<th> and <td>

<tr>

bgproperties attribute (<body>)

<bgsound> tags

bibliographic citations

<big> tags

binary files

blank lines [See paragraphs]

\_blank target 2nd 3rd

blind carbon copy (bcc) field, mail messages

<blink> tags

blinking text 2nd

block items

block quotes

<q> tags

<blockquote> tags

blocks of whitespace

body content 2nd

margins for

<body> tags 2nd 3rd

color extensions

boilerplate HTML documents

uses of <ins> and <del> tags in

boldface text 2nd

## **border attribute**

<embed>

<img>

<input type=image>

<object>

<table> 2nd

border property

border-bottom-width property

border-color property 2nd

border-left-width property

border-right-width property

border-style property 2nd

border-top-width property

border-width property 2nd

border-collapse property

border-spacing property

## **bordercolor attribute**

<frame>

<table>

<th> and <td>

<tr>

## **bordercolorlight, bordercolordark attributes**

<table>

<th> and <td>

<tr>

borders [See also margins]

colors

frame 2nd

image buttons (forms)

images 2nd

size

style properties for

shorthand properties for complex

tables 2nd 3rd

bottom value [See also alignment]2nd 3rd

boundary string

box style properties

<br> tags 2nd [See also line breaks]3rd 4th

vertical <spacer> vs.

braille value (style media)

browser extensions, XHTML Version 1.1 and

browsers [See web browsers]

buffer space [See margins]

bugs, <img> height/width attributes

bulleted (unordered) lists

bullet shape

list marker style properties

nesting

<button> tags

nesting restrictions

**buttons**

form action buttons

mouse [See event attributes JavaScript language]

radio buttons

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#)  
[\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Z\]](#)

cap-height descriptor

<caption> tags 2nd

caption, table

caption-side property

carbon copy (cc) field, mail messages

carriage returns [\[See paragraphs\]](#)

Cascading Style Sheets [\[See CSS\]](#)

case sensitivity 2nd

- elements of style rules

- XHTML tags and attribute names 2nd

case, transforming text in documents

CDATA sections

- JavaScript and CSS declarations, enclosing within  
in XML DTDs

cellpadding attribute (<table>)

cells, table [\[See tables\]](#)

cellspacing attribute (<table>)

center attribute (<img>)

<center> tags

center value (alignment)

centering [\[See alignment\]](#)

centerline descriptor

**CGI (Common Gateway Interface) programs**

- author inability to create or manage  
storing

**char attribute**

- <th> and <td>

- <tr>

character entities 2nd 3rd 4th 5th

- JavaScript

## characters

letter spacing

reserved/unsafe in URLs

word spacing 2nd

## charoff attribute

<th> and <td>

<tr>

## charset attribute

<a>

<meta>

<script>

checkboxes

child document relationship

circular bullets

circular image map areas

## cite attribute

<blockquote>

<ins> and <del>

<q>

<cite> tags 2nd

class attribute 2nd

<a>

<address>

<area>

<blockquote>

<body>

<caption>

<center>

<div>

<dl>, <dt>, and <dd>

<font>

<form> 2nd

<frameset>

- <isindex>
- <layer>
- <map>
- <multicol>
- <object>
- <p> 2nd
- <pre>
- <q>
- <ul>, <ol>, and <li>
- content-based style tags
- form controls
- physical style tags
- table attributes
- classes, style 2nd
  - inheritance and 2nd
- classid attribute, <object>
- classification style properties
- clear attribute (<br>)
- clear images [See transparent GIFs]
- clear property 2nd
- clickable image maps [See image maps]
- client-pull documents
- client-side image maps 2nd 3rd
  - example of
- clients 2nd [See also servers]
- clip attribute (<layer>)
- clip property
- code attribute (<applet>)
- <code> tags 2nd
  - filenames, use with
- codebase attribute (<applet>)
- codetype attribute (<object>)
- <col> tags

<colgroup> tags

colon (\:) and pseudoclasses

## color attribute

<basefont>

<font>

<hr>

color property 2nd

color values

colormaps 2nd

colors 2nd 3rd

background 2nd 3rd

layers

marquees

troubleshooting

<body> tag extensions

border

dithered

frame borders 2nd

graphics file formats and

horizontal rules

hyperlinks 2nd

JavaScript Style Sheets (JSS), specifying values

names

performance considerations

standard color map

style properties for 2nd

tables 2nd 3rd 4th

borders

text 2nd 3rd

transparent GIFs

true-color images

## cols attribute

<col>

`<multicol>`

`<table>`

`<textarea>`

colspan attribute (`<td>` and `<th>`)

columns **[See also tables]**

frames

`<multicol>` tags

multiline text-entry areas

tables 2nd

defining column groups

text layout in 2nd

whitespace between (gutter)

comma (,) in styles

comments 2nd 3rd

`<comment>` tags

in `<style>` tags

in XML DTDs

Common Gateway Interface **[See CGI programs]**

**compact attribute**

`<dl>`

`<ol>`

`<ul>`

conditional sections, XML DTD

content-based style tags 2nd 3rd

table of

Content-Disposition header 2nd

content attribute (`<meta>`) 2nd

content property

open-quote and close-quote values

Content-Type header 2nd 3rd

content types 2nd

application/x-www-form-urlencoded

file-selection controls and



literal\_text, plain\_text, style\_text

multipart/form-data

multipart/mixed

multipart/x-mixed-replace

text/css

text/plain

contextual styles 2nd

controls attribute (<img>)

controls, form [See forms, input controls]

conventions for HTML programming

converting HTML documents to XHTML

coordinates in image maps

**coords attribute**

<a>

<area>

counter-increment property

counter-reset property

counters

CSS (Cascading Style Sheets) [See also CSS2 standard]2nd 3rd

box properties

cascading of effects

classification properties

color and background properties

comments in

CSS2 standard [See CSS2 standard]

font properties

how to use

list properties

using effectively

quick reference

rectangular box model for

style properties

style syntax

- tagless styles (<span>)
- text properties
- CSS2 standard [See also CSS]<sup>2nd</sup>
  - @import and @media at-rules
  - audio properties
  - border shorthand properties
  - box properties
  - browser versions, supporting and non-supporting
  - classification properties
  - counters
  - escape entities
  - font properties
  - font-matching algorithm
  - formatting model
  - generated content properties
  - list properties
  - markers, creating
  - media-specific style sheets, defining
  - pagination control features for printing
  - pseudoclasses <sup>2nd</sup> [See also pseudoclasses]
  - pseudoelements defined in
  - selectors**
    - contextual
    - universal, child, and adjacent
  - style properties
  - style syntax
  - table properties
- cue properties
- custom bullets
- custom image buttons (forms)
- cycling documents

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

dashed borders

data attribute (<object>)

## data, exchanging with XML

- connecting systems

- document exchange

datetime attribute (<ins> and <del>)

<dd> tags

declare attribute (<object>)

declaring XML entities and elements [See DTD for XML]

defer attribute (<script>)

definition lists 2nd

- using appropriately

definitions-src descriptor

<del> tags

## delay

- document refresh

- loading [See performance]

- marquee movement

## deprecated attributes

- align

  - <table>

  - for all tags 2nd

- background, <body>

- bgcolor, <body>

- border, <img>

- class, for lists

- compact, for lists 2nd 3rd

- language, <script>

- link, vlink, and alink, <body>

- name, <a>

noshade, <hr>

size, <hr>

start, for lists

style, for lists

text, <body>

## type

<script>

for lists 2nd 3rd

value, for lists

version, <html> tags

width, <hr>

## deprecated elements

HTML 4.01 standard, and

XHTML and

Version 1.1

## deprecated tags

<dir>

<applet> 2nd

for audio support

<basefont>

<center> 2nd

<dir>

<font>

font-handling tags

<isindex> 2nd

<menu>

<s>

<span>

<strike>

<u>

designing in HTML [See writing HTML documents]

<dfn> tags 2nd

dir attribute 2nd 3rd

<a>

<address>

<area>

<bdo>, overriding with

<blockquote>

<center>

<div>

<dl>, <dt>, and <dd>

<font>

<form>

<head>

<html>

<isindex>

<object>

<p> 2nd

<pre>

<q>

<title>

<ul>, <ol>, and <li>

form controls

table tags

<dir> tags

direction attribute (<marquee>)

directory lists

disabled attribute (form controls)

disc bullets

display property 2nd

displaying XML documents

displays, form contents and

dithering

<div> tags 2nd

<dl> tags 2nd

<!DOCTYPE> command

document-level styles 2nd

advantages and disadvantages of using

document-related events

## documentation

for form elements

HTML tags quick reference

<meta> tags for

documents **[See also content types]**

automation

embedding 2nd

exchanging with XML applications

HTML **[See HTML documents]**

as layers

pathnames

XHTML **[See XHTML documents]**

domains 2nd 3rd

dotted borders

double borders

double quotation marks (") in XHTML attribute values

downloading delay **[See performance]**

downloading images **[See images]**

<dt> tags 2nd

DTD for HTML 2nd 3rd

HTML 4.01 standard

## DTD for XHTML

creating

declaring

XHTML 1.0 standard

DTD for XML 2nd

comments

syntax for

conditional sections

creating (example)

elements

declaration of

grammar of

entities 2nd

declaration of

dynamic documents

client-pull

server-push

dynsrc attribute (<img>)

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#)  
[\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Z\]](#)

elements, nesting [\[See nesting\]](#)

elevation property

`<em>` tags 2nd

email, mailto URL for 2nd

    defining mail header fields

    sending form data via

`<embed>` tags

    for audio

embedded guides

embedded links [\[See hyperlinks\]](#)

embedded objects 2nd

`<applet>` tags

`<embed>` tags

`<noembed>` tags

`<object>` tags

`<param>` tags

embedded tags [\[See also HTML tags\]](#)2nd

emphasis, tags for

`<b>`

`<blink>`

`<dfn>`

`<em>`

`<i>`

`<strong>`

empty elements in XHTML documents

empty-cells property

encoding [\[See also content types\]](#)

    characters [\[See special characters\]](#)

    file-selection controls and

enctype attribute



<form>

<input type=file>

ending tags 2nd 3rd

omitting in HTML

XHTML vs. HTML documents

entities 2nd 3rd 4th

JavaScript

URL encodings

entity and element declarations (XML) [See DTD for XML]

equals sign (=) for tag attributes

escape entities, CSS2 standard

## event attributes

<area> (client-side image maps)

<form>

<input type=button> (push buttons)

<map>

<ul>

<a>

<address>

<blockquote>

<center>

content-based tags

<div>

form controls

frames and

<h#>

<img>

objects and

<p>

physical style tags

<pre>

<q>

table tags

event handlers, JavaScript 2nd [See also event attributes]

executable content

- applets

- JavaScript [See JavaScript language]

- JavaScript style sheets

explicit label associations (forms)

Extended Font Model

Extensible Markup Language [See XML]

extensions, HTML

external style sheets 2nd

- advantages and disadvantages of using

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#)  
[\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Z\]](#)

## face attribute

<basefont>

<font>

family, font

favorite\_font\_size() function

<fieldset> tags

file server

File Transfer Protocol [\[See FTP\]](#)

file-selection controls (forms)

## files

file URLs

HTML [\[See HTML documents\]](#)

listing in directory lists

XHTML [\[See XHTML documents\]](#)

\:first pseudoclass

\:first-child pseudoclass

\:first-letter pseudoelement

\:first-line pseudoelement

fixed value (background image position)

flashing text 2nd

float property 2nd

floating elements, rules for margin collapsing

flood-filling images

flowing text [\[See wrapping text\]](#)

\:focus pseudoclass

font-family property 2nd

font property 2nd

font-size property 2nd 3rd 4th

font-style property 2nd

<font> tags

font-variant property 2nd

font-weight property 2nd

@font-face at-rule

## fonts

color [See colors]

descriptors

font size [See text size]

heading tags to change

HTML tags for

<basefont>

<font>

Extended Font Model

JavaScript Style Sheets (JSS), tags property

style properties for

## footers

rules with

table

for attribute (<label>)

foreground colors

<form> tags 2nd

nesting restrictions

format [See also page layout]

alignment [See alignment]

encoding [See content types]

graphics formats 2nd

HTML documents [See page layout styles]

indentation [See indentation whitespace]

list items

multimedia file formats

paragraph rendering

preformatted text [See preformatted text]

styles [See styles]

forms 2nd

<button> tags

example of

input controls

- action buttons

- checkboxes

- common attributes for

- hidden fields

- labeling and grouping

- multiline text-entry areas

- multiple-choice elements

- radio buttons

- text fields

layout of

- improving

mailto URL, with

nested tables with

programming

- parameters in URLs

writing effectively

FQDNs

fragment identifiers 2nd

- <a> tags as

- tables of contents, and

frame attribute (<table>)

frame-by-frame animation

<frame> tags 2nd

**frameborder attribute**

- <frame>

- <frameset>

frames 2nd

- alignment

- borders

- contents of

<frame> tags 2nd

<frameset> tags 2nd 3rd 4th

as hyperlink targets 2nd 3rd 4th 5th

image maps and

inline

layout

margins and borders

<noframes> tags

opening multiple at one time

scrolling

tips and tricks

Frameset DTD 2nd

<frameset> tags 2nd 3rd 4th

framesets

framespacing attribute (<frameset>)

## **FTP (File Transfer Protocol)**

ftp URLs

obtaining browsers via

fully qualified domain names

[SYMBOL] [A] [B] [C] [D] [E] [F] **[G]** [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R]  
[S] [T] [U] [V] [W] [X] [Z]

general entities, XML

generated content properties

generic style classes

GET method 2nd

GIF (Graphics Interchange Format) 2nd

GIF animation

GIF89a 2nd

glossary document relationship

gopher URLs

grammar, elements

nonterminals and terminals

XML

mixed element content

grammar, HTML

graphics **[See images]**2nd **[See images]**

Graphics Interchange Format **[See GIF]**

grooved borders

grouping form elements

grouping grammar rules, elements

guides, embedded

gutter attribute (<multicol>)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R]  
[S] [T] [U] [V] [W] [X] [Z]

<h#> tags 2nd

handheld value (style media)

hanging indents

head document relationship

head of HTML documents 2nd 3rd 4th

<head> tags 2nd 3rd

## headers

- rules with

- table 2nd

- vertical spacers with

headers attribute (<th> and <td>)

headings 2nd

- allowed content

- images in

- side heads (tables)

- straddle heads (tables)

- text size and

## height attribute

- <applet>

- <embed>

- <iframe>

- <img>

- <marquee>

- <spacer>

- <table>

- <tr>

height property 2nd

helper applications 2nd

hexadecimal character equivalents, escape characters in CSS2

hexadecimal color values



hexadecimal RGB triples

hidden attribute (<embed>)

hidden form fields

hidden text-entry fields

hide value (layer visibility)

home pages 2nd 3rd

horizontal [See also width attribute]

- alignment [See alignment]

- margins [See margins]

- rules 2nd

  - alignment

  - colors

  - combining attributes

  - flat, 2D

  - in headers and footers

  - as section dividers

  - size attribute

  - universal attributes for

  - width

- whitespace

hot keys for hyperlinks

\:hover pseudoclass

<hr> tags 2nd

href attribute

- <a> 2nd

- <area>

- <base>

- <link>

hreflang attribute (<a>)

hspace attribute

- <embed>

- <img>

- <marquee>

## HTML

- creation of
  - deprecated attributes, elements, and tags 2nd [See also deprecated]
  - documents in [See HTML documents]
  - DTD (Document Type Definition) 2nd 3rd 4th
  - extensions to
    - avoiding
    - modules
    - nonstandard
  - grammar of
  - limitations of
  - object model
  - standardizing with XML [See XHTML]
  - styles [See styles]
  - tags [See HTML tags]
  - text editors for 2nd
    - autogenerated markup
    - choosing
    - using effectively
  - tips and tricks
- Version 4.0
- Version 4.01 2nd
  - XHTML, and
- XHTML vs.
- HTML attributes [See HTML tags]
- HTML documents [See also web browsers]
  - boilerplates
  - colors in [See colors]
  - columns [See columns]
  - content vs. appearance
  - content, types of
  - converting to XHTML
  - designing for your audience

- document automation
- document-level styles 2nd
- document-related events
- dynamic
- editorial markup tags
- executable content
- forms [See forms]
- frames [See frames]
- headings in [See headings]
- home pages 2nd 3rd
- images in [See images]
- linking to [See hyperlinks]
- margins for body content
- myfirst.html (example)
- pathnames
- refreshing automatically
- relationships between
- searchable 2nd
- sectioning 2nd
- structure of 2nd 3rd
- style sheets [See styles]
- styles [See styles]
- tables [See tables]
- tables of contents
- titles of 2nd 3rd
- whitespace in [See whitespace]
- <html> tags
- HTML tags 2nd
- <html> tags 2nd
- HTML tags 2nd [See also under specific attribute and tag names]
- attributes for
  - valueless, XHTML values for
- content-based style 2nd 3rd 4th

deprecated [See deprecated tags]

for editorial markup

empty, in XHTML format

font handling

grammar for

nesting

obsolete [See deprecated tags]

omitting

physical style 2nd

quick reference

starting and ending tags 2nd 3rd

styles for [See styles]

tagless styles (<span>)

HTTP (Hypertext Transfer Protocol)

Redirect header

Refresh header

http-equiv attribute (<meta>) 2nd

http servers

http URLs

hyperlinks 2nd 3rd 4th [See also <a> tags]

colors for

effective use of

embedded guides

to external content

image maps, clickable 2nd 3rd

<area> tags 2nd 3rd

<map> tags 2nd 3rd

web browsers and

images and

linking within documents

to multiple frames

navigating with Tab and hot keys

relationships between

states of

targets for 2nd 3rd 4th 5th

image maps and

Hypertext Markup Language **[See HTML]**

Hypertext Transfer Protocol **[See HTTP]**

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R]  
[S] [T] [U] [V] [W] [X] [Z]

<i> tags 2nd

<em> tags vs.

id attribute 2nd

<a>

<address>

<basefont>

<blockquote>

<center>

<div> 2nd

<dl>, <dt>, and <dd>

<form>

<frameset>

<isindex>

<label>

<map>

<object>

<p> 2nd

<q>

<ul>, <ol>, and <li>

for hyperlink targets

form controls

for style classes

table tags

XHTML documents

## identifiers (IDs)

articles in newsgroups

messages on news servers

IE [See Internet Explorer]

IETF (Internet Engineering Task Force)

<iframe> tags

ignored HTML tags

<ilayer> tags

## image maps

- areas 2nd

- clickable 2nd 3rd

  - <area> tags

  - <map> tags

  - web browsers and

- client-side 2nd 3rd 4th

- coordinates

- HTML documents and

- performance

- server-side 2nd 3rd

images **[See also multimedia]**2nd 3rd 4th

- alignment 2nd

- background 2nd 3rd

  - behind specific elements

  - layers

  - placement of

  - in tables 2nd

  - tiling whole background

  - troubleshooting

- borders 2nd

- clickable **[See image maps]**

- combining attributes for

- converting to GIF89a

- custom image buttons (forms)

- download performance

- flowing text around

- graphics formats 2nd

- in headings

- hyperlinks and

- as list item signifiers

- margin around
- preventing from scrolling 2nd
- resizing
- reusing
- rules [See horizontal, rules]
- size
- text flow around
- text in place of
- transparent [See transparent GIFs]
- true-color images
- when to use
- wrapping text around

<img> tags 2nd

- custom image buttons (forms)
- emulating spacers with
- video extensions

implicit label associations (forms)

@import at-rule

imported external style sheets 2nd

- linked vs.

inclusions

indentation

- abusing <dt> for
- block quotes
- nested unordered lists, using for
- paragraphs, with <spacer>
- text-indent property for

index document relationship

infinite value (marquee looping)

inherit value (layer visibility)

inheritance, styles and 2nd

**inline**

- frames



images

items

layers

references

styles 2nd

advantages and disadvantages of using

<input> tags

action buttons

checkboxes (type=checkbox)

file-selection (type=file)

hidden fields (type=hidden)

masked text-entry (type=password)

radio buttons (type=radio)

text-entry (type=text)

<ins> tags

inset borders

interaction pseudoclasses

interlacing

## internationalization

dir and lang attributes

dir attribute, overriding with <bdo>

Internet

Internet Engineering Task Force (IETF)

Internet Explorer 2nd

<comment> tag

action attribute

Active technology and

animated text support

audio features

<basefont> tags, and

<bgsound> tags

color attribute

content-based tags

CSS supporting versions  
evolution of standards, and  
Extended Font Model tags, and  
<img> tags, extensions to  
inline audio, and  
JavaScript supporting versions  
leftmargin attribute  
line-breaking in  
market dominance of  
multimedia support  
notab attribute  
nowrap attribute  
obtaining  
palette attribute  
<q> tags and  
table cell attributes and options  
tags and line-breaking  
text, direction and justification  
title attribute  
topmargin attribute  
video extensions  
intranets  
IP (Internet protocol)  
addresses  
<isindex> tags 2nd 3rd  
ismap attribute (<img>) 2nd 3rd  
italic 2nd [See also emphasis, tags for]3rd

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#)  
[\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Z\]](#)

JavaScript language 2nd

event handlers 2nd 3rd **[See also event attributes]**

frames and

<img> attributes

JavaScript pseudo-protocol

JavaScript pseudo-URLs

<noscript> tags

overriding hyperlink targets

<script> tags

style sheet properties

style sheets (JSS) 2nd

JavaScript Style Sheets **[See JSS]**

JPEG format 2nd

JSS (JavaScript Style Sheets) 2nd

encoding of

style sheet properties

justification **[See alignment, text]**

justify value (align attribute)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [**K**] [L] [M] [N] [O] [P] [Q] [R]  
[S] [T] [U] [V] [W] [X] [Z]

<kbd> tags 2nd

keyboard events

keyboard input, tag for

keyword property values

keywords for documents

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R]  
[S] [T] [U] [V] [W] [X] [Z]

## label attribute

<optgroup>

<option>

<label> tags

nesting restrictions

labeling form elements

lang attribute 2nd 3rd

<a>

<address>

<area>

<blockquote>

<center>

<div>

<dl>, <dt>, and <dd>

<font>

<form>

<head>

<html>

<isindex>

<object>

<p> 2nd

<pre>

<q>

<title>

<ul>, <ol>, and <li>

form controls

table tags

XHTML Version 1.1, absence from

\:lang pseudoclass

language attribute (<script>)

## languages

computer, defining with metalanguages

pseudoclasses for

## layers

alignment 2nd

<ilayer> tags

<layer> tags

visibility of

left alignment [See alignment]

## left attribute

<ilayer>

<layer>

\:left pseudoclass

leftmargin attribute (<body>)

<legend> tags

## length [See size]

length property values

less-than sign (<)

letter-spacing property 2nd

<li> tags 2nd

line breaks 2nd 3rd [See also preformatted text]4th

allowing with <wbr>

nowrap attribute and

suppressing with <nobr>

vertical <spacer>

line-height property 2nd

line-style-image property

line-style-position property

line-style property

line-style-type property

line-through text style

link attribute (<body>)

\:link pseudoclass

<link> tags 2nd 3rd 4th [See also external style sheets]5th 6th

web browser limitations

linked external style sheets 2nd

imported vs.

links [See hyperlinks]

list-style-image property

list-style-position property

list-style property 2nd

list-style-type property 2nd

<listing> tags

lists 2nd

definition lists

directory lists

of hyperlinks

items of 2nd

menu lists

nesting

ordered (numbered)

unordered (bulleted)

ordered (numbered)

selection lists (forms)

style properties for

using effectively

unordered (bulleted)

using appropriately

literal\_text content type

loading delay [See performance]

longdesc attribute

<frame>

<img>

loop attribute

<img>

<marquee>

## looping

- marquee text

- video

lowsrc attribute (<img>)

Lynx

- content-based tags, and

- images, substituting text for

- text display limitations



[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#)  
[\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Z\]](#)

mailto URLs 2nd

- defining mail header fields

- form data via

<map> tags 2nd 3rd

maps [\[See image maps\]](#)

- image [\[See image maps\]](#)

- standard color

margin-border property

margin collapsing

margin property

margin-left property

margin-right property

margin-top property

marginheight attribute (<frame>)

margins [\[See also borders; whitespace\]](#)

- body content

- around CSS boxes

- frames

- images

- marquee areas

- style properties for

- around table cells

marginwidth attribute (<frame>)

markers

markup metalanguage, XML as

<marquee> tags

masked text-entry fields

mathline descriptor

**maxlength attribute**

- <input type=file>

<input type=text>  
mayscript attribute (<applet>)  
@media at-rule  
media attribute (<style>)  
menu lists  
<menu> tags  
message IDs, news servers  
<meta> tags 2nd 3rd  
metalanguages, defining computer languages with  
method attribute (<form>)  
Microsoft Internet Explorer **[See Internet Explorer]**  
Microsoft Word 2000, creating HTML documents with  
middle value **[See also alignment]**2nd 3rd  
MIME types 2nd  
    application/x-www-form-urlencoded  
    file-selection controls and  
    multipart/form-data  
    multipart/mixed  
    multipart/x-mixed-replace  
    text/css  
    text/plain  
missing HTML tags  
mix keyword  
monitor, form contents and  
**monospaced text**  
    <code> tags  
    <kbd> tags  
    <plaintext> tags  
    <tt> tags  
    <var> tags  
Mosaic browser  
mouse-related events  
    pseudoclasses for

mouse-sensitive images [See image maps]

movies [See animation video]

MSIE [See Internet Explorer]

<multicol> tags 2nd

multicolumn layout [See columns]

multiline text-entry areas

multimedia 2nd

- audio

- browser handling of

- client-pull feature for

- common file formats

- GIF animation

- images [See images]

- text animation

- video, <img> extensions for

multipart/form-data encoding

multipart/mixed encoding

multipart/x-mixed-replace encoding

multiple attribute (<select>)

multiple-choice elements (forms)

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [**N**] [O] [P] [Q] [R]  
[S] [T] [U] [V] [W] [X] [Z]

n attribute (<nextid>)

## name attribute

<a>

<applet>

<basefont>

<div>

<embed>

<form>

<frame> 2nd

<img>

<label>

<layer>

<map>

<meta>

<object>

<param>

form input elements

XHTML documents

XHTML Version 1.1, restrictions in

name servers

named form parameters 2nd

named frames [See frames, as hyperlink targets]

namespaces, XHTML DTDs

naming conventions for HTML

navigating with hyperlinks

Navigator [See Netscape]

## nesting

content-based style tags

contextual style rules

<frameset> tags

- HTML tags
- language pseudoclasses and layers **[See layers]**
- lists
- <multicol> tags
- physical style tags
- tables
- XHTML documents, elements in

Netscape 2nd

- content-based tags
- CSS supporting versions
- evolution of standards, and
- JavaScript supporting versions
- line-breaking in
- obtaining
- plug-ins 2nd
- table cell attributes and options
- tags and line-breaking
- text, direction and justification

news URLs

newsgroups

next document relationship

<nextid> tags 2nd

nntp URLs

<nobr> tags 2nd **[See also line breaks]**

- centered content and

<noembed> tags

<noframes> tags

nohref attribute (<area>)

nonstandard attributes, video extensions, Internet Explorer

nonterminals

noresize attribute (<frame>)

<noscript> tags

noshade attribute (<hr>)

"not visited" link state

## notab attribute

<map>

<object>

form controls

## nowrap attribute

<div>

<table>

<th> and <td>

<tr>

numbered (ordered) lists 2nd

list marker style properties

nesting

numbering style

using appropriately

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

## & (ampersand)

entities 2nd 3rd

in URLs

in XHTML

@ at symbol [[See at-rules](#)]

\\ (backslash), CSS2 escape characters and

object attribute (<applet>)

object model (HTML v4.0)

<object> tags 2nd

obsolete tags [[See deprecated tags](#)]

\: (colon) and pseudoclasses

, (comma) in styles

{ } (curly braces)

= (equals sign) for tag attributes

offset [[See alignment](#)]

<ol> tags 2nd

< (less-than sign)

<!-- --> tags 2nd

omitting HTML tags

onAbort attribute 2nd

onBlur attribute

onChange attribute 2nd

onClick attribute 2nd

onDbClick attribute 2nd

onError attribute 2nd

onFocus attribute

onKeyDown attribute 2nd

onKeyPress attribute 2nd

onKeyRelease attribute

onKeyUp attribute

onLoad attribute 2nd 3rd

onMouseDown attribute 2nd

onMouseMove attribute 2nd

onMouseOut attribute 2nd 3rd

onMouseOver attribute 2nd 3rd 4th 5th

onMouseUp attribute 2nd

onReset attribute 2nd 3rd

onSelect attribute 2nd

onSubmit attribute 2nd 3rd

onUnload attribute 2nd

**% (percent sign)**

for character encoding

in URL encoding

+ (plus sign) in URL encoding

**# (pound sign)**

for entities

for name anchors

in URLs

<optgroup> tags

<option> tags

? (question mark) in URLs 2nd

**" (quotation mark)**

for attribute values 2nd

in URLs

ordered (numbered) lists 2nd

list marker style properties

nesting

numbering style

using appropriately

orphans

; (semicolon) in character entities

**/ (slash)**

in ending tags



in URLs 2nd

[] (square brackets)

~ (tilde) in URLs

outset borders

overlining

overriding hyperlink targets

[\[SYMBOL\]](#) [\[A\]](#) [\[B\]](#) [\[C\]](#) [\[D\]](#) [\[E\]](#) [\[F\]](#) [\[G\]](#) [\[H\]](#) [\[I\]](#) [\[J\]](#) [\[K\]](#) [\[L\]](#) [\[M\]](#) [\[N\]](#) [\[O\]](#) [\[P\]](#) [\[Q\]](#) [\[R\]](#)  
[\[S\]](#) [\[T\]](#) [\[U\]](#) [\[V\]](#) [\[W\]](#) [\[X\]](#) [\[Z\]](#)

`<p>` tags 2nd 3rd

padding [\[See margins\]](#)

padding properties

padding property

padding-bottom property

padding-left property

padding-right property

padding-top property

@page at-rule

page boxes

size property

page layout [\[See also format\]](#)2nd

alignment [\[See alignment\]](#)

columns

designing for your audience

forms 2nd

frames

HTML tags for

layers

multiple columns

style sheets [\[See styles\]](#)

tables [\[See tables\]](#)

wrapping text [\[See wrapping text\]](#)

page-break properties

palette attribute (`<embed>`)

panose-1 descriptor

paragraphs 2nd

indenting with `<spacer>`

`<param>` tags

parameter entities, XML

parameters, form 2nd

parent document relationship

\_parent target

parsed and unparsed entities, XML

password input fields

pathnames 2nd

pause properties

PCDATA, XML tags 2nd

## percent sign (%)

- for character encoding

- in URL encoding

percentage property values

## performance

- applets

- background images

- client-pull documents

- colors

- flood-filling

- image maps

- images and

- lowsrc attribute (<img>) for

- marquee movement

- server-push documents

- text

physical style tags 2nd 3rd

- summary of

- table of

physical text wrapping

pitch property

pitch-range property

plain\_text content type

<plaintext> tags

play-during property

plug-in accessories 2nd 3rd 4th

pluginspage attribute (<embed>)

plus sign (+) in URL encoding

polygonal image map area

## ports

- ftp servers

- gopher servers

- nntp

- telnet

- web servers

positioning list item markers

POST method 2nd

## pound sign (#)

- for entities

- for name anchors

- in URLs

<pre> tags 2nd

- nesting restrictions

precedence of styles

preformatted text

- <listing> tags

- <pre> tags

- <xmp> tags

prev document relationship

print formatting for HTML/XHTML documents

- named pages, using

- pagination controlling

  - widows and orphans

- tables

- using multiple formats

print value (style media)

private webs

profile attribute (<head>)

programming forms

parameters in URLs

projection value (style media)

prompt attribute (<isindex>)

properties, style

## Cascading Style Sheets

colors

keywords

length

percentage

URL

JavaScript Style Sheets

property\value pairs, styles

protocols

pseudoclasses, style

\:active

colon (\:)

\:first-child

\:focus

\:hover

hyperlink states

\:lang

\:link

nesting, languages

for user interaction 2nd

\:visited

web browser support for

pseudoelements for styles

\:after

\:before

\:first-letter

\:first-line

punctuation conventions for HTML

push buttons

[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R]  
[S] [T] [U] [V] [W] [X] [Z]

<q> tags

query URLs

QUERY\_STRING variable

question mark (?) in URLs 2nd

quick reference for HTML tags

quotation marks (")

for attribute values 2nd

in URLs

quotes property

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

radio buttons

random URL generator

readability, whitespace and

readonly attribute (form controls)

rectangular image map area

Redirect header

redundant HTML tags

## references

- to external content

- inline

- to multimedia elements

Refresh header

## rel attribute

- `<a>`

- `<link>`

## relationships

- between HTML documents

- between hyperlinks

relative font size

relative frame sizes

relative URLs 2nd

- advantages of using

- `<base>` tags

repeat keyword

repetition grammar rules, elements

Requests for Comments (RFCs)

reserved characters [[See special characters](#)]

reset buttons

resizing images

reusing images



## rev attribute

`<a>`

`<link>`

RFCs (Requests for Comments)

RGB color values [\[See also colors\]](#)<sup>2nd</sup>

richness property

ridged borders

right alignment [\[See alignment\]](#)

`\:right` pseudoclass

## rows attribute

`<frameset>`

`<textarea>`

rows, creating in tables [\[See also tables\]](#)<sup>2nd</sup>

rowspan attribute (`<th>` and `<td>`)

ruby text

rules [\[See horizontal, rules\]](#)

rules attribute (`<table>`)

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

<s> tags

<samp> tags 2nd

scheme attribute (<meta>)

schemes, URL 2nd

scope attribute (<th> and <td>)

screen value (style media)

<script> tags 2nd

scroll value (marquee behavior)

scrollamount attribute (<marquee>)

scrolldelay attribute (<marquee>)

## scrolling

fixing images against 2nd

frames

<marquee> tags and

scrolling attribute (<frame>)

search parameter of URLs

searchable documents 2nd

## sections

document 2nd

performance and

table

## security

ActiveX and

mailto forms, problems with

<select> tags

selected attribute (<option>)

selection lists

selectors

gopher URLs

## style rules

contextual 2nd

multiple

universal, child, and adjacent selectors

\_self target

semicolon (;) in character entities

sequence grammar rules, elements

<server> tags

server-push documents 2nd

server-side applications

server-side image maps 2nd 3rd

servers

data to/from [See forms]

file servers

form programming

ftp servers

gopher servers

http servers

<isindex> tags and

nnntp servers

telnet servers

SGML (Standard Generalized Markup Language)

<!DOCTYPE> command in HTML documents

limitations of

SGML DTD [See DTD for HTML]

shadowing text

**shape attribute**

<a>

<area> 2nd

shapes attribute (<object>)

sharp sign [See pound sign]

show value (layer visibility)

side heads (tables)

size [See also height attribute; size attribute; width attribute]

- applets
- borders 2nd
- column width [See columns]
- CSS boxes 2nd 3rd
- embedded objects 2nd
- font size
- form entry controls
- frames 2nd 3rd
- horizontal rule
- image map areas 2nd
- images
- layers
- line height
- marquee area
- selection lists
- table cells
- tables
- text
  - <basefont> tags
  - <big> tags
  - Extended Font Model
  - <font> tags
  - <small> tags
- text-entry fields
- whitespace blocks

## size attribute

- <basefont>
- <font>
- <hr>
- <input type=file>
- <input type=text>
- <multiple>
- <spacer>

## slash (/)

- in ending tags

- in URLs 2nd

- `/*` and `*/` markers

- slide value (marquee behavior)

- slope descriptor

- `<small>` tags

- software **[See also text editors]**

  - for designers

  - formatting code

  - for writing HTML documents

- solid borders

- sound **[See audio]**

- space **[See whitespace]**

- `<spacer>` tags

## span attribute

- `<col>`

- `<colgroup>`

- `<span>` tags

- speak property

- speak-header property

- special characters 2nd 3rd 4th 5th 6th 7th

  - JavaScript entities

  - in URLs

  - XHTML, handling in

- special processing directives, XML

- speech-rate property

- square brackets ([ ])

- square bullets

## src attribute

- `<bgsound>`

- `<embed>`

- `<frame>`

<img> 2nd

<input type=image>

<layer>

<script>

src descriptor

stacking layers [See layers]

Standard Generalized Markup Language [See SGML]

standardizing HTML 2nd 3rd 4th 5th

XHTML standard

standby attribute (<object>)

**start attribute**

<img>

<ol>

starting tags 2nd

state, hyperlink

stemv and stemh descriptors

straddle heads (tables)

stress property

Strict DTD 2nd

strike-through text style

<strong> tags 2nd

structural tags

style, text [See text]

style attribute 2nd 3rd

<a>

<address>

<area>

<blockquote>

<body>

<caption>

<center>

<div>

<dl>, <dt>, and <dd>

<font>

<form> 2nd

<frameset>

<isindex>

<layer>

<li>

<map>

<multicol>

<object>

<p> 2nd

<pre>

<q>

<ul>, <ol>, and <li>

content-based style tags

form controls

physical style tags

table attributes

<style> tags 2nd 3rd 4th 5th 6th

comments in

dir, lang, and title attributes

@import at-rule

style\_text content type

styles

box properties

classes for 2nd

classification properties

color and background properties

contextual selectors 2nd

CSS properties for

document-level 2nd 3rd

external style sheets 2nd

font style properties

how to use

inline 2nd

JavaScript style sheets (JSS)

properties with CSS equivalents

list properties

media-specific

precedence

property\value pairs

pseudoclasses for

pseudoelements for

style sheets 2nd **[See also CSS; JSS]**

linked vs. imported

XML documents

tagless styles (<span>)

text style properties

web browser limitations

<sub> tags

subdomains **[See also domains]****[See also domains]**2nd

submit buttons

subscripts and superscripts

summary attribute (<table>)

<sup> tags

systems, exchanging data with XML



[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

## Tab key

- navigating form controls

- navigating hyperlinks

- navigating objects

## tabindex attribute

- `<a>`

- `<map>`

- `<object>`

- form controls

`<table>` tags 2nd

table-layout property

tables 2nd 3rd 4th [[See also columns](#)]

- alignment 2nd 3rd

- borders 2nd 3rd

- captions

- cells

- alignment

- padding

- size

- spacing

- colors in 2nd 3rd

- borders

- columns 2nd

- defining column groups

- controlling page layout with

- example of basic structure

- headers and footers

- HTML v4.0 tags for

- nesting

- rows 2nd 3rd

- sectioning

- side heads

- size

- straddle heads

- wrapping text in cells 2nd 3rd

tables of contents

**taborder attribute**

- `<map>`

- form controls

tabs [See whitespace]

tagless styles (`<span>`)

tags

tags property (JavaScript)

**target attribute**

- `<a>` 2nd 3rd

- `<area>`

- `<base>` 2nd

- `<form>`

- special values for

`<tbody>` tags

`<td>` tags 2nd

telnet URLs

- user and password

templates for HTML documents

terminals

terms, definition lists

text

- addresses

- alignment 2nd

- animating

- appearance tags for

  - content-based style tags

  - physical style tags

- block quotes
- breaking lines [\[See line breaks\]](#)
- color of [\[See colors\]](#)<sup>2nd</sup> [\[See colors\]](#)
- flowing around images
- form fields for
- headings [\[See headings\]](#)
- inline references in
- instead of images <sup>2nd</sup>
- margins for body content
- monospaced [\[See monospaced text\]](#)
- multicolumn layout <sup>2nd</sup>
- multiline text-entry areas (forms)
- paragraphs [\[See paragraphs\]](#)
- preformatted
- size of [\[See text size\]](#)
- special characters <sup>2nd</sup> <sup>3rd</sup> <sup>4th</sup> <sup>5th</sup>
  - JavaScript entities
  - URL encodings for
- structural tags
- style properties for
- text-only browsers <sup>2nd</sup>
- text/css encoding
- text/plain encoding
- three-dimensional appearance
- whitespace [\[See whitespace\]](#)
- wrapping [\[See wrapping text\]](#)<sup>2nd</sup> [\[See wrapping text\]](#)

text-align property <sup>2nd</sup>

text attribute (<body>)

text-decoration property <sup>2nd</sup>

text editors <sup>2nd</sup>

text-indent property

**text size**

- <basefont> tags

<big> tags

Extended Font Model

<font> tags

heading tags for

<small> tags

text-entry fields (forms)

text-shadow property

text-transform property 2nd

text/css encoding

text/plain encoding

<textarea> tags

texttop value **[See also alignment]**2nd

<tfoot> tags

<th> tags 2nd

<thead> tags

three-dimensional appearance, text

Tidy utility for HTML-to-XHTML conversions

tilde (~) in URLs

tiling with images

title attribute

<a>

<applet>

<area>

<div>

<dl>, <dt>, and <dd>

<form>

<frame>

<frameset>

<img>

<isindex>

<link>

<map>

<object>

<p> 2nd

<ul>, <ol>, and <li>

form controls

table tags

<title> tags 2nd

## titles

bibliographic

choosing

document 2nd 3rd

forms

frames

hyperlinked documents 2nd

image map area

sections

table captions

toc document relationship

## top attribute

<ilayer>

<layer>

\_top target

top value 2nd 3rd [See also alignment]

topline descriptor

topmargin attribute (<body>)

<tr> tags 2nd

Transitional DTD 2nd

transparent GIFs 2nd

troubleshooting background images/colors

true-color images

<tt> tags 2nd

tty value (style media)

tv value (style media)

## type attribute

<a>

<button>

<embed>

<input> [See forms, input controls]

<li> 2nd

<link>

<object>

<ol>

<param>

<script>

<spacer> 2nd 3rd

<style>

<ul>

type in gopher URLs

typecodes in ftp URLs

typographic conventions for HTML

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

<u> tags

<ul> tags 2nd

underscoring 2nd

unicode-range descriptor

uniform resource locators [[See URLs](#)]

## unique identifiers (IDs)

- articles in newsgroups

- messages on news servers

units attribute (<embed>)

units-per-em descriptor

universal child selectors

unnamed form parameters

unordered (bulleted) lists

unordered lists

### bulleted

- bullet shape

- list marker style properties

- nesting

- using appropriately

- directory lists

unsafe characters in URLs

URLs (uniform resource locators) 2nd

- absolute vs. relative

- <base> tags

- character encodings in

- file URLs

- form parameters in 2nd

- ftp URLs

- generating randomly

- gopher URLs

http URLs

JavaScript pseudoprotocol

javascript URLs

mailto URLs 2nd 3rd

defining mail header fields

news and nntp URLs

query URLs

as style property values

telnet URLs

## usemap attribute

<img> 2nd 3rd

<object>

Usenet news system

user and password, telnet URLs

user-interface design

user-related event handlers



[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

valid XML documents

## valign attribute

<caption>

<table>

<th> and <td>

## value attribute

<li>

<option>

<param>

valuetype attribute (<param>)

<var> tags 2nd

version attribute (<html>)

vertical [[See also height attribute](#)]

alignment [[See alignment](#)]

margins [[See margins](#)]

whitespace

vertical-align property 2nd

video [[See also animation; multimedia](#)]<sup>2nd</sup>

client-pull feature for

<img> extensions

inline

virtual text wrapping

visibility attribute (<layer>)

"visited" link state

\:visited pseudoclass

vlink attribute (<body>)

voice-family property

volume property

## vspace attribute

<embed>

<img>

<marquee>

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

W3C (World Wide Web Consortium)

`<wbr>` tags [[See also line breaks](#)]<sup>2nd</sup>

centered content and

Web

information on

navigating with hyperlinks

web browsers

applet rendering

character entities, rendering

client-pull documents

client-side image maps and

executable content

form limitations

HTML documents, use in editing

**HTML tags**

handling missing

ignoring

image borders

image presentation

images, rendering

incompatible with embedded objects

incompatible with executable content

incompatible with frames

Internet Explorer [[See Internet Explorer](#)]

JavaScript [[See JavaScript language](#)]

leniency in data acceptance

`<link>` tags and

Mosaic browser

Netscape Navigator [[See Netscape](#)]

obtaining

styles [See styles]

text-only 2nd

web servers

<server> tags

server-push documents

webs, private

weight, font

**well-formed documents**

XHTML and

XML 2nd

white-space property 2nd

whitespace

between columns (gutters)

blocks of

frames and

handling in block tags

hanging indents

around horizontal rules

HTML tags for

<img> tags for

indentation [See indentation]

letter spacing

line breaks [See line breaks]

line height

**margins**

body content

images

marquee areas

<nobr> tags

paragraphs [See paragraphs]

readability and

<spacer> tags

around table cells

tabs in preformatted text

word spacing 2nd

widows

**width attribute**

<applet>

<embed>

<hr>

<iframe>

<img>

<layer>

<marquee>

<multicol>

<pre>

<spacer>

<table>

<th> and <td>

width property 2nd

widths descriptor

windows **[See also frames]**2nd **[See also frames]**

as hyperlink targets 2nd 3rd 4th 5th

image maps and

tips and tricks

word processors **[See text editors]**

word-spacing property 2nd

word wrap **[See wrapping text]**

World Wide Web **[See Web]**

World Wide Web Consortium (W3C)

wrap attribute (<textarea>)

wrapping text **[See also line breaks]**

around images 2nd

<multicol> and

nowrap attribute (<div>)

table cell contents 2nd 3rd

in <textarea> entry areas

## writing HTML documents

applets

dynamic documents

editorial markup

forms, how to use 2nd 3rd

programming

hyperlinking effectively

image maps and

lists, how to use 2nd

server-push documents

software for 2nd

styles, how to use

tables

page layout with

table sections

tips and tricks

titles, choosing

user-interface design

www [See Web]

[[SYMBOL](#)] [[A](#)] [[B](#)] [[C](#)] [[D](#)] [[E](#)] [[F](#)] [[G](#)] [[H](#)] [[I](#)] [[J](#)] [[K](#)] [[L](#)] [[M](#)] [[N](#)] [[O](#)] [[P](#)] [[Q](#)] [[R](#)]  
[[S](#)] [[T](#)] [[U](#)] [[V](#)] [[W](#)] [[X](#)] [[Z](#)]

x-height descriptor

XHTML 2nd

- authoring tools

- case sensitivity in style rule elements

- deciding to use

- documents in [[See XHTML documents](#)]

- DTDs

  - deprecated elements, and

- HTML conversion software

- HTML vs.

- machine-generated content and

- tags, quick reference

- Version 1.0 and HTML 4.01

- Version 1.1

  - elements to avoid

  - modules

  - ruby text

- well-formed documents and

- XML, using to define

## XHTML documents

### content

- appearance vs.

- types of

- creating

  - DTDs, declaring

  - example document

- ending tags in

- nesting elements in

XML (Extensible Markup Language) 2nd 3rd

- DTDs

as markup metalanguage

special processing directives

uses for

- connecting systems

- document exchange

- HTML, standardizing with

xmlns attribute, defining namespaces with

<xmp> tags



[SYMBOL] [A] [B] [C] [D] [E] [F] [G] [H] [I] [J] [K] [L] [M] [N] [O] [P] [Q] [R]  
[S] [T] [U] [V] [W] [X] [**Z**]

z-index attribute (<layer>)

z-index property



## **HTML & XHTML: The Definitive Guide, 5th Edition**

By [Bill Kennedy](#), [Chuck Musciano](#)

- [Table of Contents](#)
- [Index](#)
- [Reviews](#)
- [Examples](#)
- [Reader Reviews](#)
- [Errata](#)

Publisher : O'Reilly  
Pub Date : August 2002  
ISBN : 0-596-00382-X  
Pages : 670  
Slots : 1

Copyright

Dedication

Preface

Our Audience

Text Conventions

Versions and Semantics

HTML Versus XHTML

Comments and Questions

Acknowledgments

Chapter 1. HTML, XHTML, and the World Wide Web

Section 1.1. The Internet

Section 1.2. Talking the Internet Talk

Section 1.3. HTML and XHTML: What They Are

Section 1.4. HTML and XHTML: What They Aren't

Section 1.5. Standards and Extensions

Section 1.6. Tools for the Web Designer

Chapter 2. Quick Start

Section 2.1. Writing Tools

Section 2.2. A First HTML Document

Section 2.3. Embedded Tags

Section 2.4. HTML Skeleton

Section 2.5. The Flesh on an HTML or XHTML Document

- Section 2.6. Text
- Section 2.7. Hyperlinks
- Section 2.8. Images Are Special
- Section 2.9. Lists, Searchable Documents, and Forms
- Section 2.10. Tables
- Section 2.11. Frames
- Section 2.12. Style Sheets and JavaScript
- Section 2.13. Forging Ahead

## Chapter 3. Anatomy of an HTML Document

- Section 3.1. Appearances Can Deceive
- Section 3.2. Structure of an HTML Document
- Section 3.3. Tags and Attributes
- Section 3.4. Well-Formed Documents and XHTML
- Section 3.5. Document Content
- Section 3.6. HTML/XHTML Document Elements
- Section 3.7. The Document Header
- Section 3.8. The Document Body
- Section 3.9. Editorial Markup
- Section 3.10. The `<bdo>` Tag

## Chapter 4. Text Basics

- Section 4.1. Divisions and Paragraphs
- Section 4.2. Headings
- Section 4.3. Changing Text Appearance and Meaning
- Section 4.4. Content-Based Style Tags
- Section 4.5. Physical Style Tags
- Section 4.6. Precise Spacing and Layout
- Section 4.7. Block Quotes
- Section 4.8. Addresses
- Section 4.9. Special Character Encoding
- Section 4.10. HTML's Obsolete Expanded Font Handling

## Chapter 5. Rules, Images, and Multimedia

- Section 5.1. Horizontal Rules
- Section 5.2. Inserting Images in Your Documents
- Section 5.3. Document Colors and Background Images
- Section 5.4. Background Audio
- Section 5.5. Animated Text
- Section 5.6. Other Multimedia Content

## Chapter 6. Links and Webs

- Section 6.1. Hypertext Basics
- Section 6.2. Referencing Documents: The URL
- Section 6.3. Creating Hyperlinks

- Section 6.4. Creating Effective Links
- Section 6.5. Mouse-Sensitive Images
- Section 6.6. Creating Searchable Documents
- Section 6.7. Relationships
- Section 6.8. Supporting Document Automation

## Chapter 7. Formatted Lists

- Section 7.1. Unordered Lists
- Section 7.2. Ordered Lists
- Section 7.3. The <li> Tag
- Section 7.4. Nesting Lists
- Section 7.5. Definition Lists
- Section 7.6. Appropriate List Usage
- Section 7.7. Directory Lists
- Section 7.8. Menu Lists

## Chapter 8. Cascading Style Sheets

- Section 8.1. The Elements of Styles
- Section 8.2. Style Syntax
- Section 8.3. Style Classes
- Section 8.4. Style Properties
- Section 8.5. Tagless Styles: The <span> Tag
- Section 8.6. Applying Styles to Documents

## Chapter 9. Forms

- Section 9.1. Form Fundamentals
- Section 9.2. The <form> Tag
- Section 9.3. A Simple Form Example
- Section 9.4. Using Email to Collect Form Data
- Section 9.5. The <input> Tag
- Section 9.6. The <button> Tag
- Section 9.7. Multiline Text Areas
- Section 9.8. Multiple Choice Elements
- Section 9.9. General Form-Control Attributes
- Section 9.10. Labeling and Grouping Form Elements
- Section 9.11. Creating Effective Forms
- Section 9.12. Forms Programming

## Chapter 10. Tables

- Section 10.1. The Standard Table Model
- Section 10.2. Basic Table Tags
- Section 10.3. Advanced Table Tags
- Section 10.4. Beyond Ordinary Tables

## Chapter 11. Frames

- Section 11.1. An Overview of Frames
- Section 11.2. Frame Tags
- Section 11.3. Frame Layout
- Section 11.4. Frame Contents
- Section 11.5. The <noframes> Tag
- Section 11.6. Inline Frames
- Section 11.7. Named Frame or Window Targets

## Chapter 12. Executable Content

- Section 12.1. Applets and Objects
- Section 12.2. Embedded Content
- Section 12.3. JavaScript
- Section 12.4. JavaScript Style Sheets (Antiquated)

## Chapter 13. Dynamic Documents

- Section 13.1. An Overview of Dynamic Documents
- Section 13.2. Client-Pull Documents
- Section 13.3. Server -Push Documents

## Chapter 14. Netscape Layout Extensions

- Section 14.1. Creating Whitespace
- Section 14.2. Multicolumn Layout
- Section 14.3. Layers

## Chapter 15. XML

- Section 15.1. Languages and Metalanguages
- Section 15.2. Documents and DTDs
- Section 15.3. Understanding XML DTDs
- Section 15.4. Element Grammar
- Section 15.5. Element Attributes
- Section 15.6. Conditional Sections
- Section 15.7. Building an XML DTD
- Section 15.8. Using XML

## Chapter 16. XHTML

- Section 16.1. Why XHTML?
- Section 16.2. Creating XHTML Documents
- Section 16.3. HTML Versus XHTML
- Section 16.4. XHTML 1.1
- Section 16.5. Should You Use XHTML?

## Chapter 17. Tips, Tricks, and Hacks

- Section 17.1. Top of the Tips
- Section 17.2. Cleaning Up After Your HTML Editor
- Section 17.3. Tricks with Tables

- Section 17.4. Transparent Images
- Section 17.5. Tricks with Windows and Frames

## Appendix A. HTML Grammar

- Section A.1. Grammatical Conventions

- Section A.2. The Grammar

## Appendix B. HTML/XHTML Tag Quick Reference

- Section B.1. Core Attributes

- Section B.2. HTML Quick Reference

## Appendix C. Cascading Style Sheet Properties Quick Reference

## Appendix D. The HTML 4.01 DTD

## Appendix E. The XHTML 1.0 DTD

## Appendix F. Character Entities

## Appendix G. Color Names and Values

- Section G.1. Color Values

- Section G.2. Color Names

- Section G.3. The Standard Color Map

## Colophon

## Index

## 15.8 Using XML

Our address example is trivial. It hardly scratches the surface of the wide range of applications that XML is suited for. To whet your appetite, here are some common uses for XML that you will certainly be seeing now and in the future.

### 15.8.1 Creating Your Own Markup Language

We touched on this earlier when we mentioned that the latest versions of HTML are being reformulated as compliant XML DTDs. We cover the impact XML has on HTML in the next chapter.

But even more significantly, XML enables communities of users to create languages that best capture their unique data and ideas. Mathematicians, chemists, musicians, and professionals from hundreds of other disciplines can create special tags that represent unique concepts in a standardized way. Even if no browser exists that can accurately render these tags in a displayable form, the ability to capture and standardize information is tremendously important for future extraction and interpretation of these ideas.

For more mainstream XML applications with established audiences, it is easy to envision custom browsers being created to appropriately display the information. Smaller applications or markets may have more of a challenge creating markup languages that enjoy such wide acceptance. Creating the custom display tool for a markup language is difficult; delivering that tool for multiple platforms is expensive. As we've noted, some of these display concerns can be mitigated by appropriate use of style sheets. Luckily, XML's capabilities extend beyond document display.

### 15.8.2 Document Exchange

Because XML grew out of the tremendous success of HTML, many

people think of XML as yet another document-display tool. In fact, the real power of XML lies not in the document-display arena, but in the world of data capture and exchange.

Despite the billions of computers deployed worldwide, sharing data is as tedious and error-prone as ever. Competing applications do not operate from common document-storage formats, so sending a single document to a number of recipients is fraught with peril. Even when vendors attempt to create an interchange format, it still tends to be proprietary and often is viewed as a competitive advantage for participating vendors. There is little incentive for vendors to release application code for the purpose of creating easy document-exchange tools.

XML avoids these problems. It is platform-neutral, generic, and can perform almost any data-capture task. It is equally available to all vendors and can easily be integrated into most applications. The stabilization of the XML standard and the increasing availability of XML authoring and parsing tools is making it easier to create XML markup languages for document capture and exchange.

Most importantly, document exchange rarely requires document presentation, thus eliminating "display difficulties" from the equation. Often, an existing application uses XML to include data from another source and then uses its own internal display capabilities to present the data to the end user. The cost of adding XML-based data exchange to existing applications is relatively small.

### **15.8.3 Connecting Systems**

A level below applications, there is also a need for systems to exchange data. As business-to-business communication increases, this need grows even faster. In the past, this meant that someone had to design a protocol to encode and exchange the data. With XML, exchanging data is as easy as defining a DTD and integrating the parser into your existing applications.

The data sets exchanged can be quite small. Imagine shopping for a new



PC on the Web. If you could capture your system requirements as a small document using an XML DTD, you could send that specification to a hundred different vendors to quote you a system. If you extend that model to include almost anything you can shop for from cars to hot tubs XML provides an elegant base layer of communication among cooperating vendors on the Internet.

Almost any data that is captured and stored can more easily be shared using XML. For many systems, the XML DTDs may define a data-transfer protocol and nothing more. The data may never actually be stored using the XML-defined markup; it may exist in an XML-compatible form only long enough to pass on the wire between two systems.

One increasingly popular use of XML is Web Services, which make it possible for diverse applications to discover each other and exchange data seamlessly over the Internet, regardless of their programming language or architecture. For more information on Web Services, consult *Web Services Essentials* by Ethan Cerami (O'Reilly).

In conjunction with XML-based data exchange, the Extensible Stylesheet Language, or XSL, is increasingly being used to describe the appearance and definition of the data represented by these XML DTDs. Much like CSS and its ability to transform HTML documents, XSL supports the creation of style sheets for any XML DTD. CSS can be used with XML documents as well, but it is not as programmatically rich as XSL. While CSS stops with style sheets, XSL is a style language. XSL certainly addresses the need for data display, and it also provides rich tools that allow data represented with one DTD to be transformed into another DTD in a controlled and deterministic fashion. A complete discussion of XSL is beyond the scope of this book; consult *XSLT* by Doug Tidwell (O'Reilly) for complete details.

The potential for XML goes well beyond that of traditional markup and presentation tools. What we now see and use in the XML world is only scratching the surface of the potential for this technology.

## **15.8.4 Standardizing HTML**

Last, but certainly not least, XML is being used to define a standard version of HTML known as XHTML. XHTML retains almost all of the features of HTML 4.01, but it also introduces a number of minor (and a few not-so-minor) differences. The next chapter compares and contrasts XHTML and HTML, mapping out the differences so that you can begin creating documents that comply with both the HTML and XHTML standards.

## 10.1 The Standard Table Model

The standard model for tables is fairly straightforward: a table is a collection of numbers and words arranged in rows and columns of *cells*. Most cells contain the data values; others contain row and column headers that describe the data.

Define a table and include all of its elements between the `<table>` tag and its corresponding `</table>` end tag. Table elements, including data items, row and column headers, and captions, each have their own markup tags. Working from left to right and top to bottom, you define, in sequence, the header and data for each column cell across and down the table.

The latest standards also provide a rich collection of tag attributes, many of which once were popular extensions to HTML as supported by the popular browsers. They make your tables look good, including special alignment of the table values and headers, borders, table rule lines, and automatic sizing of the data cells to accommodate their content. The various popular browsers have slightly different sets of table attributes; we'll point out those variations as we go.

### 10.1.1 Table Contents

You can put nearly anything you might have within the body of an HTML or XHTML document inside a table cell, including images, forms, rules, headings, and even another table. The browser treats each cell as a window unto itself, flowing the cell's content to fill the space, but with some special formatting provisions and extensions.

### 10.1.2 An Example Table

Here's a quick example that should satisfy your itching curiosity to see what an HTML table looks like in a source document and when finally

rendered, as shown in [Figure 10-1](#). More importantly, it shows you the basic structure of a table, from which you can infer many of the elements, tag syntax and order, attributes, and so on, and to which you may refer as you read the following various detailed descriptions:

```
<table border cellspacing=0 cellpadding=5>

  <caption align=bottom>

    Kumquat versus a poked eye, by gender</caption>

  <tr>

    <td colspan=2 rowspan=2></td>

    <th colspan=2 align=center>Preference</th>

  </tr>

  <tr>

    <th>Eating Kumquats</th>

    <th>Poke In The Eye</th>

  </tr>

  <tr align=center>

    <th rowspan=2>Gender</th>

    <th>Male</th>

    <td>73%</td>

    <td>27%</td>
```

```
</tr>

<tr align=center>

  <th>Female</th>


  <td>16%</td>

  <td>84%</td>

</tr>

</table>
```

**Figure 10-1. HTML table example rendered by Netscape (top) and by Internet Explorer (bottom)**



figs/htm5\_1001.gif

### 10.1.3 Missing Features

At one time, standard HTML tables didn't have all the features of a full-fledged table-generation tool you might find in a popular word processor. Rather, the popular browsers, Internet Explorer and Netscape in particular, provided extensions to the language.

What was missing was support for running headers and footers, particularly useful when printing a lengthy table. Another was control over table rules and divisions.

Today, the standards are ahead of the browsers in terms of table features; HTML 4 and XHTML standardize the many extensions and provide additional solutions.

## 4.10 HTML's Obsolete Expanded Font Handling

In earlier versions of this book, we rejoiced that HTML Version 3.2 had introduced a font-handling model for richer, more versatile text displays. When HTML 4 deprecated these special font-handling tags, we nonetheless included them in the same prominent position within this chapter, since they were still part of the HTML 3.2 standard and were still very popular with HTML authors, besides being well supported by all the popular browsers. We could not do the same for this edition of the book.

Like many deprecated HTML tags and attributes, the expanded font-handling tags of HTML 3.2 were here yesterday and are gone today. Netscape 6, the second most popular browser in use today, has dropped support for the some of the tags altogether. Since Internet Explorer, the world's most popular browser, still displays them, we include the Extended Font Model tags at the end of this chapter, with all the implicit red flags waving hard.

The W3C wants authors to use cascading style sheets, not acute tags and attributes, for explicit control of the font styles, colors, and sizes of the text characters. That's why these extended font tags and related attributes have fallen into disfavor. It's now time for you to eschew the extended font tags, too.

### 4.10.1 The Extended Font Size Model

Instead of absolute point values, the Extended Font Model of HTML 3.2 uses a relative means for sizing fonts. Sizes range from 1, the smallest, to 7, the largest; the default (base) font size is 3.

It is almost impossible to state reliably the actual font sizes used for the various virtual sizes. Most browsers let the user change the physical font size, and the default sizes vary from browser to browser. It may be helpful to know, however, that each virtual size is successively 20% larger or smaller than the default font size, 3. Thus, font size 4 is 20%

larger, font size 5 is 40% larger, and so on, while font size 2 is 20% smaller and font size 1 is 40% smaller than font size 3.

### 4.10.2 The `<basefont>` Tag (Deprecated)

The `<basefont>` tag lets you define the basic size for the font that the browser will use to render normal document text. We don't recommend that you use it, as it has been deprecated in the HTML 4 and XHTML standards and is no longer supported by Netscape.



## **<basefont>**

### *Function*

Defines the base font size for relative font-size changes

### *Attributes*

`color, face, id, name, size`

### *End tag*

`</basefont>`; often omitted in HTML

### *Contains*

Nothing

### *Used in*

*block, head\_content*

The `<basefont>` tag recognizes the `size` attribute, whose value determines the document's base font size. It may be specified as an absolute value, from 1 to 7, or as a relative value (by placing a plus or minus sign before the value). In the latter case, the base font size is increased or decreased by that relative amount. The default base font size is 3.

Internet Explorer supports two additional attributes for the `<basefont>` tag: `color` and `name`. HTML 4 also defines the `face` attribute as a synonym for the `name` attribute. These attributes control the color and typeface used for the text in a document and are used just like the analogous `color` and `face` attributes for the `<font>` tag, described in the next section.

HTML 4 also defines the `id` attribute for the `<basefont>` tag, allowing

you to label the tag uniquely for later access to its contents. [[Section 4.1.1.4](#)]

Authors typically include the `<basefont>` tag in the head of an HTML document, if at all, to set the base font size for the entire document. Nonetheless, the tag may appear nearly anywhere in the document, and it may appear several times throughout the document, each with a new `size` attribute. With each occurrence, the `<basefont>` tag's effects are immediate and hold for all subsequent text.

In an egregious deviation from the HTML and SGML standards, Internet Explorer does *not* interpret the ending `</basefont>` tag as terminating the effects of the most recent `<basefont>` tag. Instead, the `</basefont>` end tag resets the base font size to the default value of 3, which is the same as writing `<basefont size=3>`.

The following example source and [Figure 4-21](#) illustrate how Internet Explorer responds to the `<basefont>` tag and `</basefont>` end tag:

```
Unless the base font size was reset above,
```

```
Internet Explorer renders this part in font size 3.
```

```
<basefont size=7>
```

```
This text should be rather large (size 7).
```

```
<basefont size=6> Oh,
```

```
<basefont size=4> no!
```

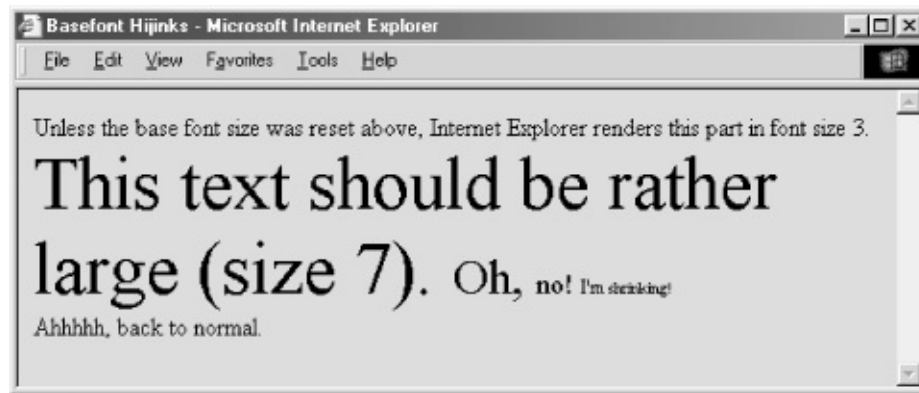
```
<basefont size=2> I'm
```

```
<basefont size=1> shrinking!
```

```
</basefont><br>
```

Ahhhh, back to normal.

**Figure 4-21. Playing with <basefont>**



We recommend against using `</basefont>`; use `<basefont size=3>` instead.

### 4.10.3 The `<font>` Tag (Deprecated)

The `<font>` tag lets you change the size, style, and color of text. We don't recommend that you use it, because it has been deprecated in the HTML 4 and XHTML standards (even though it is still supported by Internet Explorer and Netscape). But should you decide to ignore our advice, use it like any other physical or content-based style tag for changing the appearance of a short segment of text.

## <font>

### *Function*

Sets the font size for text

### *Attributes*

`class, color, dir, face, id, lang, size, style, title`

### *End tag*

`</font>;` never omitted

### *Contains*

*text*

### *Used in*

*text*

To control the color of text for the entire document, see the attributes for the `<body>` tag, described in [Section 5.3.1](#).

### 4.10.3.1 The size attribute

The value of the `size` attribute must be one of the virtual font sizes (1-7) described earlier, defined as an absolute size for the enclosed text or preceded by a plus or minus sign (+ or -) to define a relative font size that the browser adds to or subtracts from the base font size (see the `<basefont>` tag, [Section 4.10.2](#)). The browsers automatically round the size to 1 or 7 if the calculated value exceeds either boundary.

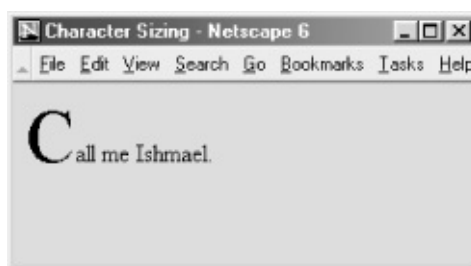
In general, use absolute size values when you want the rendered text to be an extreme size, either very large or very small, or when you want an entire paragraph of text to be a specific size.

For example, using the largest font for the first character of a paragraph makes for a crude form of illuminated manuscript (see [Figure 4-22](#)):

```
<p>
```

```
<font size=7>C</font>all me Ishmael.
```

**Figure 4-22. Exaggerating the first character of a sentence with the size attribute for <font>**



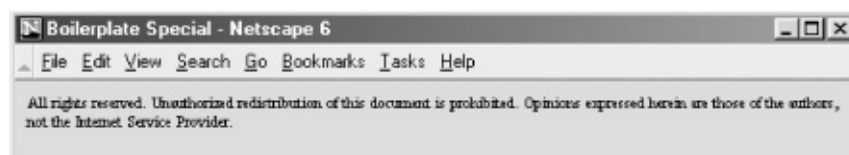
Also, use an absolute font when inserting a delightfully unreadable bit of "fine" print boilerplate or legalese at the bottom of your document (see [Figure 4-23](#)):

```
<p>
```

```
<font size=1>
```

```
All rights reserved. Unauthorized redistribution of th  
prohibited. Opinions expressed herein are those of the  
Internet Service Provider.
```

**Figure 4-23. Use the tiniest font for boilerplate text**

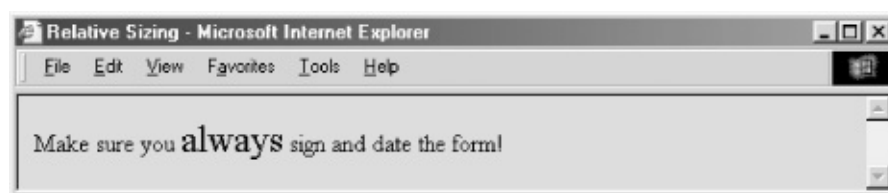


Except for the extremes, use relative font sizes to render text in a size different than the surrounding text, to emphasize a word or phrase. For an exaggerated example, see [Figure 4-24](#):

<p>

Make sure you <font size=+2>always</font> sign and date the form!

**Figure 4-24. Use relative sizes for most text embellishments**



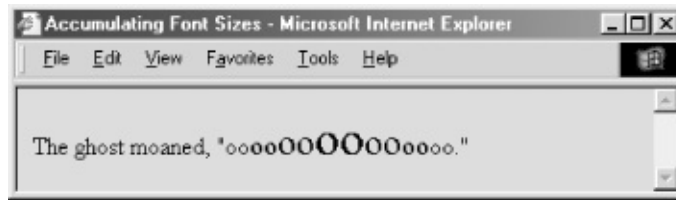
If your relative size change results in a size greater than 7, the browser uses font size 7. Similarly, font sizes less than 1 are rendered with font size 1.

Note that specifying `size=+1` or `size=-1` is identical in effect to using the `<big>` and `<small>` tags, respectively. However, nested relative changes to the font size are not cumulative, as they are for the alternative tags. Each `<font>` tag is relative to the base font size, not the current font size. For example (see [Figure 4-25](#)):

<p>

The ghost moaned, "oo<font size=+1>oo<font size=+2>oo<font size=+3>oo</font>oo</font>oo</font>oo."

**Figure 4-25. Relative font sizes accumulate**



Contrast this with the `<big>` and `<small>` tags, which increase or decrease the font size one level for each nesting of the tags. [[Section 4.5.2](#)]

### 4.10.3.2 The color attribute

Still supported by the popular browsers, the `color` attribute for the `<font>` tag sets the color of the enclosed text. The value of the attribute may be expressed in either of two ways: as the red, green, and blue (RGB) components of the desired color, or as a standard color name. Enclosing quotes are recommended but not required.

The RGB color value, denoted by a preceding pound sign, is a six-digit hexadecimal number. The first two digits are the red component, from 00 (no red) to FF (bright red). Similarly, the next two digits are the green component and the last two digits are the blue component. Black is the absence of color, #000000; white is all colors, #FFFFFF.

For example, to create basic yellow text, you might use:

```
Here comes the <font color="#FFFF00">sun</font>!
```

Alternatively, you can set the enclosed font color using any one of the many standard color names. See [Appendix G](#) for a list of common ones. For instance, you could have made the previous sample text yellow with the following source:

```
Here comes the <font color=yellow>sun</font>!
```

### 4.10.3.3 The face attribute

In earlier versions, Internet Explorer and Netscape Navigator let you change the font style in a text passage with the `face` attribute for the `<font>` tag.<sup>[8]</sup> Neither browser appears to support this attribute anymore.

[8] For the HTML purist, the once-powerful user who had ultimate control over the browser, this is egregious indeed. Form over function; look over content what next? Embedded video commercials you can't stop?

The quote-enclosed value of `face` is one or more display font names separated with commas. The font face displayed by the browser depends on which fonts are available on the individual user's system. The browser parses the list of font names, one after the other, until it matches one with a font name supported by the user's system. If none match, the text display defaults to the font style set by the user in the browser's preferences. For example:

This text is in the default font. But,

```
<font face="Braggadocio, Machine, Zapf Dingbats">
```

```
heaven only knows</font>
```

```
what font face is this one?
```

If the browser user has the Braggadocio, Machine, or none of the listed font typefaces installed in her system, she will be able to read the "heaven only knows" message in the respective or default font style. Otherwise, the message will be garbled, because the Zapf Dingbats font contains symbols, not letters. Of course, the alternative is true, too; you may intend that the message be a symbol-encoded secret.

#### 4.10.3.4 The `dir` and `lang` attributes

The `dir` attribute lets you advise the browser which direction the text within the tag should be displayed in, and `lang` lets you specify the language used for the tag's contents. [[Section 3.6.1.1](#)] [[Section 3.6.1.2](#)]



#### 4.10.3.5 The class, id, style, and title attributes

You can associate additional display rules for the `<font>` tag using style sheets. The rules can be applied to the `<font>` tag using either the `style` or `class` attribute. [\[Section 8.1.1\]](#) [\[Section 8.3\]](#)

You also can assign a unique id to the `<font>` tag, as well as a less rigorous title, using the respective attribute and accompanying quote-enclosed string value. [\[Section 4.1.1.4\]](#) [\[Section 4.1.1.4\]](#)

## 7.2 Ordered Lists

Use an ordered list when the sequence of the list items is important. A list of instructions is a good example, as are tables of contents and lists of document footnotes or endnotes.

### 7.2.1 The `<ol>` Tag

The typical browser formats the contents of an ordered list just like an unordered list, except that the items are numbered instead of bulleted. The numbering starts at one and is incremented by one for each successive ordered list element tagged with `<li>`. [`<li>`]

## <ol>

### *Function*

Defines an ordered list

### *Attributes*

`class, compact, dir, id, lang, onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, start, style, title, type`

### *End tag*

`</ol>`; never omitted

### *Contains*

*list\_content*

### *Used in*

*block*

HTML 3.2 introduced a number of features that provide a wide variety of ordered lists. You can change the start value of the list and select any of five different numbering styles.

Here is a sample XHTML ordered list:

```
<h3>Pickled Kumquats</h3>
```

Here's an easy way to make a delicious batch of pickles:

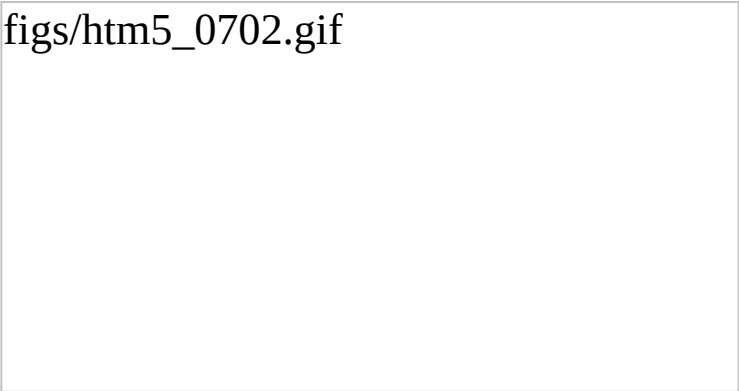
```
<ol>
```

```
  <li>Rinse 50 pounds of fresh kumquats</li>
```

```
<li>Bring eight gallons white vinegar to rolling boi  
<li>Add kumquats gradually, keeping vinegar boiling<  
<li>Boil for one hour, or until kumquats are tender<  
<li>Place in sealed jars and enjoy!</li>  
</ol>
```

Netscape renders the example as shown in [Figure 7-2](#).

**Figure 7-2. An ordered list**



figs/htm5\_0702.gif

#### 7.2.1.1 The start attribute

Normally, browsers automatically number ordered list items beginning with the Arabic numeral 1. The `start` attribute for the `<ol>` tag lets you change that beginning value. To start numbering a list at 5, for example:

```
<ol start=5>  
  
<li> This is item number 5.</li>  
  
<li> This is number six!</li>
```

```
<li> And so forth...</li>

</ol>
```

### 7.2.1.2 The `type` attribute

By default, browsers number ordered list items with a sequence of Arabic numerals. Besides being able to start the sequence at some number other than 1, you can use the `type` attribute with the `<ol>` tag to change the numbering style itself. The attribute may have a value of `A` for numbering with capital letters, `a` for numbering with lowercase letters, `I` for capital Roman numerals, `i` for lowercase Roman numerals, or `1` for common Arabic numerals. (See [Table 7-1](#).)

**Table 7-1. HTML `type` values for numbering ordered lists**

| Type value     | Generated style          | Sample sequence |
|----------------|--------------------------|-----------------|
| <code>A</code> | Capital letters          | A, B, C, D      |
| <code>a</code> | Lowercase letters        | a, b, c, d      |
| <code>I</code> | Capital Roman numerals   | I, II, III, IV  |
| <code>i</code> | Lowercase Roman numerals | i, ii, iii, iv  |
| <code>1</code> | Arabic numerals          | 1, 2, 3, 4      |

The `start` and `type` attribute extensions work in tandem. The `start` attribute sets the starting value of the item integer counter at the beginning of an ordered list. The `type` attribute sets the actual numbering style. For example, the following ordered list starts numbering

items at 8, but because the style of numbering is set to `i`, the first number is the lowercase Roman numeral "viii." Subsequent items are numbered with the same style, and each value is incremented by 1, as shown in this HTML example, and rendered as shown in [Figure 7-3](#).<sup>[1]</sup>

[1] Notice that we don't include the `</li>` end tag in the HTML example but do in all the XHTML ones. Some end tags are optional with HTML but must be included in all XHTML documents.

```
<ol start=8 type="i">

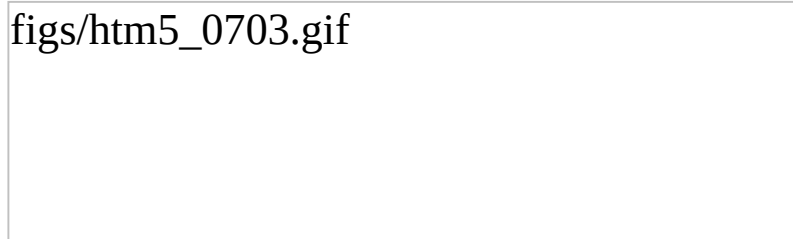
  <li> This is the Roman number 8.

  <li> The numerals increment by 1.

  <li> And so forth...

</ol>
```

**Figure 7-3. The start and type attributes work in tandem**



figs/htm5\_0703.gif

The type and value of individual items in a list can be different from those of the list as a whole, described in [Section 7.3.1](#). As mentioned earlier, the `start` and `type` attributes are deprecated in HTML 4 and XHTML. Consider using style sheets instead.

### 7.2.1.3 Compact ordered lists

Like the `<ul>` tag, the `<ol>` tag has an optional `compact` attribute that is deprecated in the HTML 4 and XHTML standards. Unless you absolutely

need to use it, don't.

#### **7.2.1.4 The class, dir, id, lang, event, style, and title attributes**

These attributes are applicable with ordered lists, too; their effects are identical to those for unordered lists. [[Section 7.1.1.3](#)] [[Section 7.1.1.4](#)] [[Section 7.1.1.5](#)] [[Section 7.1.1.6](#)]

## 7.1 Unordered Lists

Like a laundry or shopping list, an unordered list is a collection of related items that have no special order or sequence. The most common unordered list you'll find on the Web is a collection of hyperlinks to other documents. Some common topic, like "Related Kumquat Lovers' Sites," allies the items in an unordered list, but they have no order among themselves.

### 7.1.1 The `<ul>` Tag

The `<ul>` tag signals to the browser that the following content, ending with the `</ul>` tag, is an unordered list of items. Inside, each item in the unordered list is identified by a leading `<li>` tag. Otherwise, nearly anything HTML/XHTML-wise goes, including other lists, text, and multimedia elements. [`<li>`]



## <ul>

### *Function*

Defines an unordered list

### *Attributes*

`class, compact, dir, id, lang, onClick, onDblClick, onKeyDown, onKeyPress, onKeyUp, onMouseDown, onMouseMove, onMouseOut, onMouseOver, onMouseUp, style, title, type`

### *End tag*

`</ul>`; never omitted

### *Contains*

*list\_content*

### *Used in*

*block*

Typically, the browser adds a leading bullet character and formats each item on a new line, indented somewhat from the left margin of the document. The actual rendering of unordered lists, although similar for the popular browsers (see [Figure 7-1](#)), is not dictated by the standards, so you shouldn't get bent out of shape trying to attain exact positioning of the elements.

Here is an example XHTML unordered list, which Internet Explorer renders with bullets, as shown in [Figure 7-1](#):

Popular Kumquat recipes:

```
<ul>
```

```
<li>Pickled Kumquats</li>


<li>'Quats and 'Kraut (a holiday favorite!)</li>

<li>'Quatshakes</li>

</ul>
```

There are so many more to please every palate!

**Figure 7-1. A simple unordered list**



figs/htm5\_0701.gif

Tricky HTML authors sometimes use nested unordered lists, with and without `<li>`-tagged items, to take advantage of the automatic, successive indenting. You can produce some fairly slick text segments that way. Just don't depend on it for all browsers, including future ones. Rather, it's best to use the `border` property with a style definition in the paragraph (`<p>`) or division (`<div>`) tag to indent nonlist sections of your document (see [Chapter 8](#)).

#### **7.1.1.1 The type attribute**

The graphical browsers automatically bullet each `<li>`-tagged item in an unordered list. Netscape and Internet Explorer use a solid circle, for example. Browsers that support HTML 3.2 and later versions, including

4.0 and 4.01, as well as XHTML 1.0, let you use the `type` attribute to specify which bullet symbol you'd rather have precede items in an unordered list. This attribute may have a value of either `disc`, `circle`, or `square`. All the items within that list will thereafter use the specified bullet symbol, unless an individual item overrides the list bullet type, as described later in this chapter.

With the advent of standard Cascading Style Sheets, the W3C has deprecated the `type` attribute in HTML 4 and in XHTML. Expect it to disappear.

### 7.1.1.2 Compact unordered lists

If you like wide-open spaces, you'll hate the optional `compact` attribute for the `<ul>` tag. It tells the browser to squeeze the unordered list into an even smaller, more compact text block. Typically, the browser reduces the line spacing between list items; it also may reduce the indentation between list items, if it does anything at all with indentation (usually it doesn't).

Some browsers ignore the `compact` attribute, so you shouldn't depend on its formatting attributes. Also, the attribute is deprecated in the HTML 4 and XHTML standards, so it hasn't long to live.

### 7.1.1.3 The class and style attributes

The `style` and `class` attributes bring CSS-based display control to lists, providing far more comprehensive control than you would get through individual attributes like `type`. Combine the `style` attribute with the `<ul>` tag, for instance, to assign your own bullet icon image, rather than using the common circle, disc, or square. The `class` attribute lets you apply the style of a predefined class of the `<ul>` tag to the contents of the unordered list. The value of the `class` attribute is the name of a style defined in some document-level or externally defined style sheet. For more information, see [Chapter 8](#). [[Section 8.1.1](#)] [[Section 8.3](#)]

#### 7.1.1.4 The lang and dir attributes

The `lang` attribute lets you specify the language used within a list, and `dir` lets you advise the browser which direction the text should be displayed in. The value of the `lang` attribute is any of the ISO standard two-character language abbreviations, including an optional language modifier. For example, adding `lang=en-UK` tells the browser that the list is in English ("en") as spoken and written in the United Kingdom ("UK"). Presumably, the browser may make layout or typographic decisions based upon your language choice. [[Section 3.6.1.2](#)]

The `dir` attribute tells the browser which direction to display the list contents in from left to right (`dir=ltr`), like English or French, or from right to left (`dir=rtl`), as with Hebrew or Chinese. [[Section 3.6.1.1](#)]

#### 7.1.1.5 The id and title attributes

Use the `id` attribute to specially label the unordered list. An acceptable value is any quote-enclosed string that uniquely identifies the list and can later be used to unambiguously reference the list in a hyperlink target, for automated searches, as a style-sheet selector, and for a host of other applications. [[Section 4.1.1.4](#)]

You also can use the optional `title` attribute and quote-enclosed string value to identify the list. Unlike an `id` attribute, a `title` does not have to be unique. [[Section 4.1.1.4](#)]

#### 7.1.1.6 The event attributes

The many user-related events that may happen in and around a list, such as when a user clicks or double-clicks within its display space, are recognized by current browsers. With the respective "on" attribute and value, you may react to those events by displaying a user dialog box or activating some multimedia event. [[Section 12.3.3](#)]