

Building web pages with XHTML 1.1



John W. Shipman

2011-08-24 17:37

Abstract

Describes XHTML 1.1, the current preferred language for building World Wide Web pages.

This publication is available in Web form¹ and also as a PDF document². Please forward any comments to tcc-doc@nmt.edu.

Table of Contents

1. What is XHTML?	3
2. A small, complete example page	3
3. Rules for the XML notation	5
4. Differences between XHTML and HTML	6
4.1. Paired tags	6
4.2. The empty element syntax	6
4.3. Case sensitivity	7
4.4. Quoted attributes with values	7
4.5. Fragment identifiers	7
5. Separating content and presentation with CSS	7
6. Basic data types	8
6.1. The length datatype	8
6.2. The ID datatype	8
6.3. The IDREF and IDREFS datatypes	8
6.4. URIs and link targets: where hyperlinks point	9
6.5. The link datatype	9
6.6. Media type	10
6.7. MIME types: Defining a resource's format	10
7. Content model notation	10
8. Overall structure of an XHTML file	12
8.1. The root element: <code>html</code>	13
8.2. The <code>head</code> element: Overall page information	13
8.3. The <code>base</code> element: Specifying the document's base URI	14
8.4. The <code>link</code> element: Related documents	14
8.5. The <code>meta</code> element: Page meta-information	15
8.6. The <code>style</code> element: Specifying presentation style	16
8.7. The <code>script</code> element: Including executable code	17
8.8. The <code>noscript</code> element: What to do when your script can't be run	18
8.9. The <code>body</code> element	18
9. The block elements	19

¹ <http://www.nmt.edu/tcc/help/pubs/xhtml/>

² <http://www.nmt.edu/tcc/help/pubs/xhtml/xhtml.pdf>

9.1. The heading elements: <code>h1</code> , <code>h2</code> , <code>h3</code> , ..., <code>h6</code>	19
9.2. The <code>address</code> element: Who wrote this page?	20
9.3. The <code>p</code> element: Regular text paragraph	20
9.4. The <code>blockquote</code> element: Block-style quotations	21
9.5. The <code>div</code> element: A generic block container	22
9.6. The <code>pre</code> element: Display verbatim text	22
9.7. The <code>ul</code> element: Unnumbered or “bullet” lists	23
9.8. The <code>li</code> element: List item	24
9.9. The <code>ol</code> element: Numbered lists	24
9.10. The <code>dl</code> element: Definition lists	25
9.11. The <code>hr</code> element: horizontal ruled line	26
10. Inline content: <code>InLine.model</code>	26
10.1. <code>a</code> : Hyperlink	27
10.2. <code>abbr</code> : Abbreviation	28
10.3. <code>acronym</code> : Acronym	29
10.4. <code>cite</code> : Title of a work	29
10.5. <code>code</code> : Part of a computer program	29
10.6. <code>del</code> : Deleted material	29
10.7. <code>dfn</code> : Definition of a term	30
10.8. <code>em</code> : Emphasis	30
10.9. <code>img</code> : Include an image	30
10.10. <code>ins</code> : Inserted material	31
10.11. <code>kbd</code> : Keyboard input	31
10.12. <code>q</code> : Inline quotations	31
10.13. <code>samp</code> : Sample computer output	31
10.14. <code>span</code> : The generic inline container	32
10.15. <code>strong</code> : Strong emphasis	32
10.16. <code>sub</code> : Subscript	32
10.17. <code>sup</code> : Superscript	32
10.18. <code>var</code> : Variable name	32
11. Tables: the <code>table</code> element	32
11.1. Specifying table column properties	34
11.2. Sectioning a table with <code>thead</code> , <code>tbody</code> , and <code>tfoot</code>	35
11.3. Table rows: the <code>tr</code> element	36
11.4. Table cells: the <code>td</code> and <code>th</code> elements	37
12. <code>Flow.model</code> : Arbitrary content	37
13. The <code>object</code> element: Embedded multimedia and applet objects	38
13.1. The <code>param</code> element: Passing arguments to applications	39
13.2. How to delay instantiation of an object	40
14. Forms: The <code>form</code> element	41
14.1. The <code>input</code> forms control	42
14.2. The <code>label</code> element: Label a control	47
14.3. The <code>button</code> forms control	48
14.4. The <code>select</code> forms control: menus	48
14.5. The <code>option</code> element: One choice inside a <code>select</code> control	50
14.6. The <code>optgroup</code> element: A group of choices inside a <code>select</code> control	51
14.7. The <code>textarea</code> forms control: multiline text input	51
14.8. The <code>fieldset</code> element: Adding structure to a <code>form</code>	52
14.9. What makes a control successful?	53
14.10. Writing your form handler script	53
14.11. The URL encoding method for forms data	54
15. Standard attributes	55

15.1. The <code>xml:lang</code> attribute	55
15.2. The <code>charset</code> attribute: Declaring a character set	55
15.3. The common attributes: <code>Common.attrib</code>	55
15.4. The <code>id</code> attribute: Assigning a unique identifier to an element	56
15.5. The <code>class</code> attribute: Declaring an element's CSS class	56
15.6. The <code>title</code> attribute: Titling an element	56
15.7. The <code>tabindex</code> attribute: Specifying tab traversal order	57
16. Event attributes	57
17. Legacy and unrecommended elements	58
17.1. Deprecated features	58
17.2. Features that are not recommended	58

1. What is XHTML?

XHTML stands for eXtended HyperText Markup Language. You can use it to build pages on the World Wide Web. XHTML is a refinement of HTML, the original language of the Web invented by Tim Berners-Lee in the early 1990s.

This document is a quick reference for XHTML 1.1. Not every feature is covered, just the features that most people will need most of the time. For the exact definition of the language, see the *XHTML 1.1 Recommendation*³ at the World Wide Web Consortium (W3C) web site.

If you would like to check your page for validity, the W3C Consortium maintains an online validator⁴ for XHTML.

2. A small, complete example page

Here is a minimal XHTML Web page with nothing on it but a title, a major heading, one paragraph of text, and a bullet list with two bullets.

sample.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>Your page title here</title>
  </head>
  <body>
    <h1>Your major heading here</h1>
    <p>
      This is a regular text paragraph.
    </p>
    <ul>
      <li>
        First bullet of a bullet list.
      </li>
      <li>
        This is the <em>second</em> bullet.
      </li>
    </ul>
```

³ <http://www.w3.org/TR/xhtml11/>

⁴ <http://validator.w3.org/>

```
</body>
</html>
```

XHTML is a “document type” of XML, so it obeys certain structural rules; see Section 3, “Rules for the XML notation” (p. 5) for the complete rules. For the moment, all you need to know is that you use *tags* to structure a page. Tags always start with a less-than symbol (“<”) and end with a greater-than symbol (“>”).

Here is a line-by-line explanation of the example above.

<!DOCTYPE html ...>

To declare that your page is written in XHTML 1.1, include this special “document type identifier” as the first lines of your file.

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">

This is the start tag for the entire page. It has two attributes. The `xmlns` attribute specifies that this page is written in XHTML. The `xml:lang` attribute states that the language of the page is English.

Why, you might ask, are there two different methods for identifying the page as XHTML, once in the DOCTYPE identifier, and once in this `xmlns` attribute? The former method is the old way, and the latter is a newer, better way to declare the document type. For full validity in old and new tool environments, use both.

<head>

The page is divided into two regions, the head and the body. This is the start tag for the head section.

<title>Your page title here</title>

This line titles the page. Browsers will display the text “Your page title here” in the border around the browser window, not anywhere on the page.

</head>

The end tag terminating the head section of the page.

<body>

The start tag for the body section of the page.

<h1>Your major heading here</h1>

The `h1` element specifies a level-one heading, that is, the largest level of heading. This displays the text “Your major heading here” as a heading in large type.

<p>

A start tag that indicates the start of a normal text paragraph.

This is a regular text paragraph.

Content of the text paragraph.

</p>

Shows where the paragraph ends.

This start tag begins a bullet list; “ul” stands for “unnumbered list”.

Starts a new “bullet” paragraph in the bullet list; “li” stands for “list item.”

First bullet of a bullet list.

Text of the first bulleted paragraph.

End tag for the first bullet.

Start tag for the second bullet.

This is the *second* bullet.

Text of the second bulleted paragraph. The `` and `` tags mark up the word “second” as emphasized text, which is usually displayed in italic type.

End tag for the second bullet.

End tag for the entire bullet list.

</body>

End tag for the body section of the page.

</html>

End tag for the entire page.

3. Rules for the XML notation

XHTML uses the rules for XML (eXtensible Markup Language) to structure your document. Here is a brief summary of XML; you must adhere to these rules to construct a valid XHTML document.

It is good practice (though not required) to start your document with a `DOCTYPE` declaration that specifies that it is written in XHTML. For the recommended format of this declaration, see Section 2, “A small, complete example page” (p. 3).

Structure your document with *tags*. Each tag starts with a less-than symbol (`<`) and ends with a greater-than symbol (`>`).

Each tag defines the start or end of a region of the page. Such a region is called an *element*. Most elements consist of a *start tag*, followed by some *content*, followed by an *end tag*.

A start tag has this general format:

```
<type attrname="attrvalue" ...>
```

where the *type* indicates the kind of tag. Following the type name there may be zero or more *attributes* that describe various properties. Each attribute consists of an attribute name, followed by an equal sign, and a value enclosed in quotes. You may use either double quotes (e.g., `id="Neptune"`) or single quotes (e.g., `class='disreputable'`) to enclose the attribute's value.

An end tag has this format:

```
</type>
```

Some whitespace (any mixture of spaces, tabs, and newlines) is required between the *type* and the first attribute, and also between attributes. Space is also allowed before the closing `>`.

XML also allows a different type of element called the *empty element*. Here is the general form of an empty element:

```
<type attrname="attrvalue" .../>
```

An empty element functions exactly the same as a start tag and an end tag with nothing between them. These two lines are functionally identical in XHTML:

```
<hr class='heavy'></hr>
<hr class='heavy' />
```

Elements must be *properly nested*. That is, there must be an `html` element, called the *root element*, that encloses all other elements on the page. Also, an element must be completely inside, or completely outside, other elements. You can't have two elements that partially overlap; for example, the sequence “`<h1>Analyzing <cite>Ulysses</h1></cite>`” is **not** valid because the `cite` element is partially inside and partially outside the `h1` element.

XML allows you to place comments in your document. Here is the general form of a comment:

```
<!--text-->
```

where *text* is any text. The text may not contain two consecutive hyphens (- -).

4. Differences between XHTML and HTML

If you already know HTML, there are several important differences in XHTML. HTML is a document type constructed using an older, more general markup framework called SGML (Standard Generalized Markup language), and it allows for a lot of sloppiness in the construction of a Web page.

However, XHTML is an XML document type, and XML has stricter rules about constructing valid documents. This is a good thing, as it makes validation of your Web page easier. Many good tools exist that help you build valid documents in XML.

4.1. Paired tags

In XHTML, all elements must have one of two forms, either *paired tags* or an *empty element* (see Section 3, “Rules for the XML notation” (p. 5)).

By contrast, HTML allowed you to omit the closing tag. In the early days of the Web, the rule was “to start a new paragraph, use the start tag `<p>`.” But in XHTML, you must also supply the closing `</p>` end tag at the end of a paragraph.

HTML	XHTML
<code><p>First paragraph. <p>Second paragraph.</code>	<code><p>First paragraph.</p> <p>Second paragraph.</p></code>

4.2. The empty element syntax

In XML, if an element has no *content*, it can be represented as an *empty element* of this form:

```
<element-name attributes.../>
```

This rule affects HTML elements such as `hr` (horizontal rule), which draws a horizontal ruled line across the page:

HTML	XHTML
<code><hr></code>	<code><hr/></code>

HTML	XHTML
------	-------

4.3. Case sensitivity

XHTML element and attribute names are case-sensitive and must be lowercase; HTML element and attribute names are case-insensitive. So, for example, to emphasize a phrase, you use the `em` element:

HTML	XHTML
Either of these would work: <pre>emphasized text emphasized text</pre>	<pre>emphasized text</pre>

4.4. Quoted attributes with values

In XHTML, all attribute values must be enclosed in either single quotes (' ... ') or double quotes (" ... "). HTML allowed you to use unquoted attributes.

Furthermore, XHTML requires that all attributes must have a value. HTML accepted attribute names with no value.

HTML	XHTML
<pre><title border=1>...</pre>	<pre><title border='1'>...</pre>
<pre><input type=checkbox checked>...</pre>	<pre><input type="checkbox" checked="checked">...</pre>

4.5. Fragment identifiers

In HTML, named locations in the document (also known as *fragment identifiers*) were defined by the “name” attribute. In XHTML, this is replaced by the `id` attribute. Any element can carry an `id` attribute.

Suppose you want to define a fragment identifier on a page so that the browser will jump to there if the URL ends in “#planks”:

HTML	XHTML
<pre><h2>Planks</h2></pre>	<pre><h2 id="planks">Planks</h2></pre>

5. Separating content and presentation with CSS

HTML as originally proposed had a lot of features that allowed page designers to specify the appearance of the page in detail. However, XHTML tries to separate *presentation*—the fine details such as fonts, colors, and spacing—from structure and content.

The preferred way to specify presentation is to use CSS (Cascading Stylesheet Language) to write a *stylesheet* that describes presentation details, and place it in a separate file from the XHTML.

- CSS is described in a separate publication, *Styling Web pages with CSS-2*⁵.
- When you write your XHTML page, use a `link` element to connect that page with its stylesheet. See Section 8.4, “The `link` element: Related documents” (p. 14).

The good news is that you don't have to use CSS. If browsers render your page in a reasonable way, there's no reason to use CSS.

6. Basic data types

Many of XHTML's attributes use a set of standard data types such as lengths, names, and so on. The following sections describe these standard data types.

6.1. The length datatype

To specify a linear dimension, you can use any of these forms:

- A number by itself is interpreted as a number of pixels.
- A number followed by a percent sign (%) is interpreted as some percentage of available space. For example, if an element fills the width of the screen, the length “50%” would mean half the width of the screen.
- When describing the columns of a table, you can specify their widths as a number followed by an asterisk (*). The available width is then parceled out proportionally among those numbers. For example, if you specify the widths of the columns of a three-column table as “1*”, “2*”, and “3*”, the first column would get 1/6 of the available width, the second column would get 1/3 (2/6), and the third column would get half (3/6).

6.2. The ID datatype

An ID attribute is used to name some element. Accordingly, it must follow the XML rules for names:

- The first or only character must be a letter.
- The remaining characters must be letters, digits, hyphens (-), underscores (_), colons (:), or periods (.).

Examples of valid ID attributes: `x`, `b7-224.9`, and `z_and1r0n`.

6.3. The IDREF and IDREFS datatypes

An attribute of type IDREF is a reference to some other element with an attribute of type ID (as specified in Section 6.2, “The ID datatype” (p. 8)).

An attribute of type IDREFS is a list of references to elements with ID attributes, separate by spaces.

⁵ <http://www.nmt.edu/tcc/help/pubs/css/>

6.4. URIs and link targets: where hyperlinks point

The basic addressing scheme for the World Wide Web is the *Universal Resource Identifier* or URI. For example, the URI of the New Mexico Tech homepage is "http://www.nmt.edu/".

In general, a URI has as many as four parts:

```
method://host/path#fragment
```

method

The method name describes the general protocol for retrieving resources. Most Web pages use method `http`, which stands for HyperText Transfer Protocol. There are many others, such as `ftp`: for File Transfer Protocol.

host

This portion describes a specific host machine. For example, `www.nmt.edu` is a specific processor that serves Web pages.

path

Describes a specific document on the host machine.

fragment

Refers to a specific location in the document. If there is no "#" symbol, the URI refers to the beginning of the document.

In an XHTML document, a fragment identifier refers to the element that has that `id` attribute. For example, if a document contained this paragraph, the fragment identifier "#mongooses" would point there:

```
<p id='mongooses'>
  The plural of "mongoose" is "mongooses," not "mongeese."
  The word is Hindic in origin, so it does not follow the rule
  for "goose."
</p>
```

6.5. The link datatype

The `rel` and `rev` attributes, used in the `link` and `a` elements, describe the relationship between two resources (such as web pages and stylesheets).

Here is a list of the recognized link types. When used as a `rel` attribute, each link type defines the relationship between the *current* document, denoted as *C*, and some *referenced* resource, denoted as *R*. (The `rev` attribute defines the same relationship but in reverse order; in that case, *C* is the referenced document, while *R* is the current document.)

<code>stylesheet</code>	<i>R</i> is a stylesheet to be used in rendering <i>C</i> . This is probably the most common use of the <code>link</code> element. Here's an example. Suppose you have a CSS stylesheet named <code>gothic.css</code> . To apply that stylesheet to the current document, you would use this element: <pre><link rel='stylesheet' href='gothic.css' type='text/css' /></pre>
<code>alternate</code>	<i>R</i> is another version of <i>C</i> , perhaps in a different language, or set up for a different media type.
<code>appendix</code>	<i>R</i> is an appendix of <i>C</i> .
<code>contents</code>	<i>R</i> is the table of contents for the set of documents that contains <i>C</i> .

copyright	<i>R</i> is the copyright statement for <i>C</i> .
glossary	<i>R</i> contains definitions of terms used in <i>C</i> .
help	<i>R</i> is a document providing help for users of <i>C</i> .
index	<i>R</i> is the index for the set of documents containing <i>C</i> .
next	<i>R</i> is the next document in sequence after <i>C</i> .
prev	<i>R</i> is the preceding document in sequence before <i>C</i> .
section	<i>R</i> is a section of the document starting at <i>C</i> .
start	<i>R</i> is the first in a series of documents that includes <i>C</i> .
subsection	<i>R</i> is a subsection of document <i>C</i> .

6.6. Media type

These types are used to describe different media in which content might be presented.

screen	Normal computer screens.
print	Presentation on fixed-size pages, especially paper.
all	Suitable for all types of presentation.
aural	Presentation through a speech synthesizer.
braille	Tactile presentation for the blind.
handheld	Screens with limited size, low resolution, limited colors, and/or poor bandwidth.
projection	Presentation on a projector.
tty	Screens or printing terminals that can print only fixed-size characters.
tv	Screens with low resolution that may not be scrollable.

6.7. MIME types: Defining a resource's format

MIME stands for Multipurpose Internet Mail Extensions. A MIME type defines a way of structuring a file. A MIME type is expressed as a type and subtype separated by a slash. Examples: An XHTML file has type `text/html`, and a CSS stylesheet has type `text/css`.

For a full list, see the *MIME Media Types* page⁶ at the Internet Assigned Numbers Authority.

7. Content model notation

We define each element of XHTML by its structure: which attributes are required and allowed, and the element's content.

To specify these things, we use the notation of Relax NG Compact Syntax (RNC). This notation is defined elsewhere in *Relax NG Compact Syntax (RNC)*⁷, but we summarize the relevant parts here.

element *ename* { *content* }

Defines an XHTML element called *ename*. The *content* describes what can go inside the element—attributes, as well as the content between the start and end tags.

⁶ <http://www.iana.org/assignments/media-types/index.html>

⁷ <http://www.nmt.edu/tcc/help/pubs/rnc/>

attribute *aname* { *content* }

The element allows an attribute named *aname* whose value conforms to the given *content*.

The order of attributes is unimportant. When a given element is shown with multiple attributes, they can occur in any order.

text

Any text is allowed as content.

***content*?**

The given *content* is optional.

content*

The given *content* can occur zero or more times.

***content*+**

The given *content* must occur at least once, but may be repeated; that is, it can occur one or more times.

***C*₁, *C*₂**

Content *C*₁ must be followed by *C*₂.

***C*₁ & *C*₂**

Content *C*₁ and content *C*₂ must both occur, but can be in either order.

empty

Used to indicate that an element cannot have any content, and must use the empty-tag format:

```
<tag-name attrName="attrValue" .../>
```

xsd:datatype

The content must match one of the standard data types defined in the XML Schema standard. This standard defines a large number of data types, but only a few are used in XHTML:

xsd:nonNegativeInteger

A number greater than or equal to zero.

xsd:positiveInteger

A number greater than zero.

xsd:anyURI

A Universal Resource Identifier.

xsd:ID

A valid XML identifier. See Section 6.2, "The ID datatype" (p. 8).

xsd:IDREF

A reference to an XML identifier using the same rules as `xsd:ID`.

xsd:IDREFS

A list of references to XML identifiers separated by whitespace.

xsd:NMTOKEN

A name that conforms to the rules for XML names. For these rules, see Section 6.2, "The ID datatype" (p. 8). The difference between name tokens and IDs is that IDs must be unique in a document, while name tokens can be used in more than one place.

xsd:NMTOKENS

A list of `xsd:NMTOKEN` names separated by whitespace.

Here is a small example illustrating several of these features. Suppose we are defining an element called `corral`. This element has a required attribute called `corral-name` containing a name. It also allows an optional attribute called `mud-type` that contains an identifier for the type of mud that it makes when it gets wet. The element content consists of one required `fence` child followed by zero or more `horse` children:

```
element corral
{ attribute corral-name { text },
  attribute mud-type { IDREF }?,
  fence,
  horse*
}
```

An example of this element might look like:

```
<corral corral-name="0. K." mud-type="red-clay">
  <fence>6' chain link</fence>
  <horse>Trigger</horse>
  <horse>Domino</horse>
</corral>
```

The notation here is adapted from the Relax NG stylesheets provided with James Clark's `nxml-emacs` editing mode for the *emacs* text editor. For further information, see James Clark's Thai Open Source pages⁸.

8. Overall structure of an XHTML file

In order to be a valid XHTML page, your file must have at least this structure:

```
sample.html
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
 "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>T</title>
    H
  </head>
  <body>
    B
  </body>
</html>
```

That is, the root `html` element must have two child elements named `head` and `body`.

The `head` element's first child element must be a `title` element with some title content *T*. You can add other optional elements to describe the page's overall characteristics at position *H*.

Inside the `body` element you will place the actual content of your page. This may include text paragraphs, figures, tables, forms, and many other types of content.

The following sections describe the exact syntax of these elements and some others you will use to give overall structure to the page.

⁸ <http://www.thaiopensource.com/>

8.1. The root element: `html`

This element must be the root (outermost) element of any valid XHTML page. Content model:

```
element html
{ attribute xml:lang { text }?,
  attribute xmlns { xsd:anyURI }?,
  head,
  body
}
```

xml:lang

Defines the language of the document. See Section 15.1, “The `xml:lang` attribute” (p. 55).

xmlns="http://www.w3.org/1999/xhtml"

Defines this document as XHTML 1.1.

head

Document header element. See Section 8.2, “The `head` element: Overall page information” (p. 13).

body

Document body. See Section 8.9, “The `body` element” (p. 18).

8.2. The head element: Overall page information

Just after the `<html>` start tag, you'll need a `head` element that specifies general information about the page (as opposed to details of its contents). Here's the content model:

```
element head
{ attribute xml:lang { text }?,
  title & base? & link* & meta* & style* & script*
}
```

For the `xml:lang` attribute, see Section 15.1, “The `xml:lang` attribute” (p. 55).

The first child of the `head` element is required: a `title` element that declares the overall page title. Browsers will display this title in the decorative border around the browser, not in the actual body of the page.

Unlike `title` elements elsewhere, the one inside the `head` element may not have any other markup tags inside it.

The `title` element is the only required element inside the `head` element. You may also use any number of the following elements in any order:

- Section 8.3, “The `base` element: Specifying the document's base URI” (p. 14). This is important if you want to specify how relative URI references in your document are translated to absolute URIs.
- Section 8.4, “The `link` element: Related documents” (p. 14). These elements allow you to describe relationships between this page and other pages. For example, if this page is a section that appears in a table of contents elsewhere, you can describe that relationship.
- Section 8.5, “The `meta` element: Page meta-information” (p. 15). These elements describe other aspects of the document such as the author, search information, and such.
- Section 8.6, “The `style` element: Specifying presentation style” (p. 16). These elements describe how you want your page to look in terms of fonts, colors, spacing, and other presentational markup.

- `script` elements containing executable scripts associated with your page; see Section 8.7, “The `script` element: Including executable code” (p. 17).

Here is the content model for the `title` element:

```
element title
{ attribute xml:lang { text }?,
  text
} &
```

Note that the `title` element may contain only ordinary text. Do not use tags inside a `title` element.

8.3. The `base` element: Specifying the document's base URI

References to URIs (Universal Resource Identifiers) come in two flavors: absolute and relative. An absolute URI must start with a method name, such as “`http:`”. A relative URI is one that omits this information and specifies a path relative to the current document. For example, if your page `index.html` is in the same directory as another page `crikey.html`, you can use the relative URI reference `href="crikey.html"`.

However, in some situations, some programs need to be able to translate a relative URI reference to an absolute URI. To continue the example above, if the reference to “`crikey.html`” occurs on a page whose absolute URI is `http://crox.edu/index.html`, the relative reference would translate to an absolute `http://crox.edu/crikey.html`.

If you're just throwing together some Web pages, this probably won't affect you. But if at some later point you start using some of the niftier Web software, and you get into trouble with the expansion of relative URLs, recall that you can specify the base URI of the current page with an element of this form:

```
<base href="baseURI" />
```

To continue the example above, to define the proper base URI of page `index.html`, the `head` element would contain this element:

```
<base href="http://crox.edu/index.html" />
```

8.4. The `link` element: Related documents

This element defines a relationship between this page and some other page (referred to as the “linked resource”). For example, you might want to point to the table of contents for the document containing this page. Here is the content model:

```
element link
{ attribute charset { text }?,
  attribute href { xsd:anyURI }?,
  attribute hreflang { text }?,
  attribute type { text }?,
  attribute rel { xsd:NMTOKENS }?,
  attribute rev { xsd:NMTOKENS }?,
  attribute media { text }?,
  Common.attrib,
  empty
}
```

charset

The character set of the linked resource, if different from the referring document. See Section 15.2, “The `charset` attribute: Declaring a character set” (p. 55).

href

The URI of the linked resource.

hreflang

The language code of the linked resource. For definitions of the language codes, see Section 15.1, “The `xml:lang` attribute” (p. 55).

type

The MIME type of the linked resource. This describes the resources format, e.g., HTML or CSS. See Section 6.7, “MIME types: Defining a resource's format” (p. 10).

rel

Describes the type of relationship between this file and the linked resource. For possible values, see Section 6.5, “The link datatype” (p. 9). For example, if document `beverly.html` is the one following this one, you might use the element “`<link rel='next' href='beverly.html' />`”.

rev

Describes a reverse relationship, the opposite of the `rel` attribute. For example, if the current document precedes `beverly.html` in the logical sequence, you might use the element “`<link rev='prev' href='beverly.html' />`”.

media

Describes the media for which the linked resource is set up. For possible values, see Section 6.6, “Media type” (p. 10).

Common.attrib

A `link` element can include any of the attributes discussed in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Here is an example of a `link` element that specifies that the document uses `deadwood.css` as its CSS stylesheet:

```
<link rel='stylesheet' href='deadwood.css' type='text/css' />
```

8.5. The meta element: Page meta-information

You can use any number of `meta` elements inside your `head` element to describe several kinds of information about the document.

Each `meta` element names some property of the page and specifies a value for that property. In some cases, the `name` attribute names the property and the `content` attribute gives the value for the property. Some properties are defined by the HTTP protocol; in that case, the `http-equiv` attribute names the property and the `content` attribute gives its value.

Here is the content model:

```
element meta
{ attribute xml:lang { text }?,
  attribute http-equiv { xsd:NMTOKEN },
  attribute name { xsd:NMTOKEN },
  attribute content { text }?,
  attribute scheme { text }?,
```

```
empty
}
```

xml:lang

Use this attribute to specify the language used in the value of the `content` attribute; see Section 15.1, “The `xml:lang` attribute” (p. 55).

http-equiv

When specifying properties defined by HTTP, this attribute states the name of the property.

name

The name of the property being defined.

content

The value of the property being defined.

scheme

This attribute, when used, describes how the `content` property is interpreted.

The property names and values allowed here are many. Here are just a few examples. The first example gives the author's name, and tags the name as French.

```
<meta name="author" content="Pierre de Beaumarchais" xml:lang="fr">
```

The next example defines a few keywords to help search engines determine what's on the page:

```
<meta name='keywords' content='grackles, icterids, spackle'>
```

Here is an example of the use of the `http-equiv` attribute to define the document type as HTML and the character set as US ASCII:

```
<meta http-equiv='Content-Type' content='text/html; charset=US-ASCII'>
```

For a detailed discussion of the `meta` element, refer to the relevant section of the HTML 4.01 specification⁹.

Note

The Resource Description Framework (RDF) specification¹⁰ describes a much more detailed and generalized way to specify document metadata. RDF metadata is more suitable for automated processing.

8.6. The `style` element: Specifying presentation style

The best way to specify style is to attach an external CSS stylesheet to the document using the `link` element with a `rel='stylesheet'` attribute; see Section 8.4, “The `link` element: Related documents” (p. 14).

However, you can include CSS stylesheet rules directly in your document using the `style` element.

```
element style
{ attribute type { text },
  attribute xml:lang { text },
  attribute media { text }?;
```

⁹ <http://www.w3.org/TR/REC-html40/struct/global.html#h-7.4.4>

¹⁰ <http://www.w3.org/RDF/>

```
text
}
```

type

This required attribute specifies the MIME type of the stylesheet. This will usually be `type='text/css'`. For other possible values, see Section 6.7, "MIME types: Defining a resource's format" (p. 10).

xsl:lang

Specifies the language used in the `title` attribute. See Section 15.1, "The `xml:lang` attribute" (p. 55).

media

You can supply multiple style sheets for different media. In that case, use the `media` attribute in each `style` element to specify the kind of media; see Section 6.6, "Media type" (p. 10).

Here's a brief example. The CSS rules here specify that level h1 headings will use navy blue type, and the page background will be silver.

```
<style type='text/css'>
  h1 { color: blue; }
  body { background-color: silver; }
</style>
```

8.7. The script element: Including executable code

A *script* is an executable program attached to your Web page. There are two ways to attach a script:

1. To execute the script when the page is loaded, include the program in a `script` element.
2. You can also set up a script that will be executed when the user performs certain operations, like clicking or moving a mouse over an element. For this form of scripting, see Section 16, "Event attributes" (p. 57).

There is no guarantee your script will execute. In some browsing environments, scripts are disabled. Furthermore, not all languages are supported. To provide substitute content in these cases, see Section 8.8, "The `noscript` element: What to do when your script can't be run" (p. 18).

Assuming that scripting is allowed in your chosen language, any output generated by your script will replace the `script` element before the page is rendered. For example, here is a complete page that uses Javascript to generate the page dynamically:

sample.html

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
  <head>
    <title>This title will be displayed</title>
  </head>
  <body>
    <script type="text/javascript">
      document.write ( "<p>Javascript succeeded.</p>" )
    </script>
    <noscript>
      <p>Javascript failed.</p>
    </noscript>
```

```
</body>
</html>
```

If the above page is loaded in an environment that allows Javascript, the page will display the paragraph "Javascript succeeded."; if the script can't be run, it will display the paragraph "Javascript failed."

Here is the content model:

```
element script
  attribute type { text },
  attribute src { xsd:anyURI }?,
  attribute charset { text }?,
  attribute xml:space { "preserve" }?,
  text
}
```

type

This required attribute must specify the kind of script being executed. The value is a MIME type; see Section 6.7, "MIME types: Defining a resource's format" (p. 10).

src

There are two places you can put the actual code of your script. You can include it directly as the content of the `script` element. Alternately, you can place the script in a separate file and use the `src` attribute to point at the script's URI. Here's an example:

```
<script type="text/intercal" src="ical/ignatz.ic"/>
```

charset

If you use the `src` attribute to invoke a script outside the current document, and that script is in a different character set, you can use the `charset` attribute to specify that character set. See Section 15.2, "The `charset` attribute: Declaring a character set" (p. 55).

xml:space

If whitespace is significant in your scripting language, use the `xml:space="preserve"` attribute.

Unless you are using the `src` attribute to point at an external script, place your script between the `<script>` and `</script>` tags.

8.8. The `noscript` element: What to do when your script can't be run

If you use the `script` element to generate page content dynamically in a scripting language, you should also provide some content wrapped inside a `noscript` element in case scripting is disabled in your environment or in case the specific scripting language isn't supported.

For an example of `noscript`, see Section 8.7, "The `script` element: Including executable code" (p. 17).

8.9. The `body` element

Inside the `body` element, place the content of your page. This element is a child of the root `html` element. Here is the content model:

```
element body
{ attribute onload { text }?,
  attribute onunload { text }?,
  Common.attrib,
```

```
Block.class+
}
```

onload, onunload

Actions to be taken when the page is loaded or removed from the browser; see Section 16, “Event attributes” (p. 57).

Common.attrib

See Section 15.3, “The common attributes: Common.attrib” (p. 55).

Block.class+

One or more block elements. See Section 9, “The block elements” (p. 19).

9. The block elements

When the `Block.class` abbreviation appears in an element's content models, that element can contain any of the XHTML block elements.

All block elements represent rectangular areas on the page, as opposed to inline elements, which can start or end in the middle of a line inside a block element.

Here are the block elements in XHTML:

- There are six levels of headings: `h1` (first-level headings), `h2`, `h3`, `h4`, `h5`, and `h6`. For their structure, see Section 9.1, “The heading elements: `h1`, `h2`, `h3`, ..., `h6`” (p. 19).
- Section 9.2, “The `address` element: Who wrote this page?” (p. 20).
- Section 9.3, “The `p` element: Regular text paragraph” (p. 20).
- Section 9.4, “The `blockquote` element: Block-style quotations” (p. 21).
- Section 9.5, “The `div` element: A generic block container” (p. 22).
- Section 9.6, “The `pre` element: Display verbatim text” (p. 22).
- Section 9.7, “The `ul` element: Unnumbered or “bullet” lists” (p. 23).
- Section 9.9, “The `ol` element: Numbered lists” (p. 24).
- Section 9.10, “The `dl` element: Definition lists” (p. 25).
- Section 9.11, “The `hr` element: horizontal ruled line” (p. 26).
- Section 11, “Tables: the `table` element” (p. 32).
- Section 14, “Forms: The `form` element” (p. 41).

9.1. The heading elements: `h1`, `h2`, `h3`, ..., `h6`

Use an `h1` element for a major heading within your document. You can also use `h2` for a minor heading, `h3` for a subheading under a minor heading, and so forth, all the way down to `h6`. Here is the content model:

```
Heading.contents = Common.attrib, Inline.model
element h1 { Heading.contents }
element h2 { Heading.contents }
element h3 { Heading.contents }
element h4 { Heading.contents }
```

```
element h5 { Heading.contents }
element h6 { Heading.contents }
```

Common.attrib

All the heading elements can contain the attributes described in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Inline.model

Inside the heading element, use any inline content; see Section 10, “Inline content: `Inline.model`” (p. 26).

Examples:

```
<h1>Many meetings</h1>
  <h2>How to read <cite>Finnegan's Wake</cite></h2>
```

9.2. The address element: Who wrote this page?

It is good practice to place an `address` element at the bottom of your page, so readers will know how to contact you if they have questions or comments about your page.

Here is the content model:

```
element address
{ Common.attrib,
  Inline.model
}
```

Common.attrib

The `address` element can include any of the common attributes described in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Inline.model

You can place any inline content inside this element. See Section 10, “Inline content: `Inline.model`” (p. 26).

Here is an example. This might be placed just before the `</body>` end-tag. The `div` elements format each part of the address on a separate, unindented line.

```
<address>
  Sister Zoot, Castle Anthrax;
  <a href="mailto:zoot@anthrax.edu">zoot@anthrax.edu</a>
</address>
```

9.3. The p element: Regular text paragraph

To display an ordinary paragraph of text, enclose it in a `p` element. Content model:

```
element p
{ Common.attrib,
  Inline.model
}
```

Common.attrib

With the `p` element, you can use any of the common attributes discussed in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Inline.model

Inside a paragraph, you can use any mixture of text and inline elements. See Section 10, “Inline content: `Inline.model`” (p. 26).

The example below shows two consecutive paragraphs from *Anguish Languish* by Howard L. Chace¹¹.

```
<p>
  Wants pawn term dare worsted ladle gull hoe lift wetter
  murder inner ladle cordage honor itch offer lodge, dock,
  florist. Disk ladle gull orphan worry Putty ladle rat cluck
  wetter ladle rat hut, an fur disk raisin pimple colder
  Ladle Rat Rotten Hut.
</p>
<p>
  Wan moaning Ladle Rat Rotten Hut's murder colder inset.
</p>
```

9.4. The `blockquote` element: Block-style quotations

There are two ways to include a quotation in your document. If your quotation is to be “run in,” inside a regular paragraph, see the `q` element in Section 10, “Inline content: `Inline.model`” (p. 26).

The usual practice for longer quotes is to display them as a separate block, often with wider margins than the surrounding material. For such longer quotes, place the quoted material within a `blockquote` element. Here is the content model:

```
element blockquote
{ attribute cite { xsd:anyURI }?,
  Common.attrib,
  Block.class+
}
```

`cite`

This optional attribute allows you to point at an online source for the quotation. Use the URI of the online source as the value of this attribute.

Common.attrib

Any of the attributes described in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Block.class+

You will need to wrap the content of the `blockquote` element inside one or more block elements such as `p` or `div`. See Section 9, “The block elements” (p. 19).

Here's an example:

```
<p>
  As a great duck once said:
</p>
<blockquote cite="http://www.imdb.com/title/tt0052139/">
  <div>
```

¹¹ <http://www.justanyone.com/allanguish.html>

```
    Ho! Ha ha! Guard! Turn! Parry! Dodge! Spin! Thrust!  
</div>  
</blockquote>
```

9.5. The `div` element: A generic block container

The purpose of the `div` element is to hold a block-shaped chunk of content. Its content will occupy a rectangular area on your document when it is rendered. It is similar to the `p` (paragraph) element.

The main reason XHTML has a `div` element, in addition to a `p` element, is to allow you to create blocks of content that have specific style rules determined by a CSS stylesheet. Typically, you will use a `div` element with a `class` attribute to identify what kind of block you want, and then write a CSS rule that applies to that particular type of element and `class` attribute.

Here is the content model:

```
element div  
{ Common.attrib,  
  Flow.model  
}
```

Common.attrib

All the attributes defined in Section 15.3, “The common attributes: `Common.attrib`” (p. 55) can be used in a `div` element.

Flow.model

The content inside a `div` element can be any mixture of inline elements, block elements, or text. See Section 12, “`Flow.model`: Arbitrary content” (p. 37).

Here is an example of a `div` element.

```
<div class='warning'>  
  Be sure to inflate shoes before crossing the water.  
</div>
```

9.6. The `pre` element: Display verbatim text

The purpose of the `pre` element is to display a block of text exactly as it appears in the original XHTML; “pre” stands for “preformatted.”

The `pre` element works differently than a normal paragraph (the `p` element): in normal paragraphs, all whitespace (including spaces, tabs, and line breaks) is “normalized;” that is, leading and trailing spaces are ignored, and each clump of whitespace that occurs between two words is reduced to a single space. Then the resulting text is reformatted into lines that fit into the available space.

However, this kind of mashing-and-extruding process is unsuitable for the display of text such as computer programs, poetry, and such. To preserve all whitespace and line breaks in a group lines from your XHTML, enclose those lines in a `pre` element. Here is the content model:

```
element pre  
{ Common.attrib,  
  Inline.model  
}
```

Common.attrib

In a `pre` element, you can use any of the common attributes described in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Inline.model

You can use any mixture of text and inline elements as the content of a `pre` element; see Section 10, “Inline content: `Inline.model`” (p. 26).

Most browsers will not only preserve the spacing of the lines inside, but they will also render it in a monospaced font so that the letters line up in nice columns. In the example below, we definitely want the line numbers and statements to align vertically.

```
<pre>
 10 PRINT "BASIC IS OVER FORTY YEARS OLD!"
 20 GOTO 10
 30 END
</pre>
```

9.7. The `ul` element: Unnumbered or “bullet” lists

Use the `ul` (unnumbered list) element when you want to format a “bullet list” like this one:

- This is the first bullet.
- Second bullet here.

The “bullet” is the symbol, usually a round black dot, that appears before each item in the list. Here's the XHTML content model:

```
element ul
{ Common.attrib,
  li+
}
```

Common.attrib

You can use any of the attributes enumerated in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

li

Each bullet's content must be wrapped in an `li` (list item) element. There must be at least one `li` element, but there may be any number. See Section 9.8, “The `li` element: List item” (p. 24) for that element's content.

This example shows how the bullet list above would be formatted in XHTML.

```
<ul>
  <li>
    This is the first bullet.
  </li>
  <li>
    Second bullet here.
  </li>
</ul>
```

9.8. The `li` element: List item

The `li` element is a container for each bullet in an unnumbered list (`ul`), or each item in a numbered list (`ol`). Here is the content model:

```
element li
{ Common.attrib,
  Flow.model
}
```

Common.attrib

Any of the attributes from Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Flow.model

The content can be any mixture of text, inline elements, and block elements. See Section 12, “`Flow.model`: Arbitrary content” (p. 37).

9.9. The `ol` element: Numbered lists

To create a numbered list of blocks, use the `ol` (ordinal list) element. This is very similar to Section 9.7, “The `ul` element: Unnumbered or “bullet” lists” (p. 23), but the list items will be preceded by numbers instead of bullets. Here's the content model:

```
element ol
{ Common.attrib,
  li+
}
```

Common.attrib

Any of the attributes from Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

li

Each item in the list must be wrapped in a `li` (list item) element; see Section 9.8, “The `li` element: List item” (p. 24). There must be at least one `li` element, but there can be any number.

Here's an example from the *U. S. Declaration of Independence*. You don't have to assign numbers yourself; this will be formatted with the list items labeled 1, 2, 3.

```
<ol>
  <li>
    He has affected to render the Military independent of and
    superior to the Civil Power.
  </li>
  <li>
    He has combined with others to subject us to a jurisdiction
    foreign to our constitution, and unacknowledged by our
    laws; giving his Assent to their Acts of pretended Legislation:
  </li>
  <li>
    For depriving us in many cases, of the benefit of Trial by Jury:
  </li>
</ol>
```

9.10. The `dl` element: Definition lists

One of the most useful XHTML constructs is the `dl` or definition list element. Typically you will use this for lists containing definitions of terms. Here's an example:

akagai

Red clam.

sawagani

A tiny freshwater crab, usually grilled and eaten whole.

A definition list is usually presented with the terms set “flush left” (that is, unindented), and each definition is formatted as an indented block below the term. Here is the content model:

```
element dl
{ Common.attrib,
  (dt | dd)+
}
dt = element dt
{ Common.attrib,
  Inline.model
}
dd = element dd
{ Common.attrib,
  Block.model
}
```

Common.attrib

All three elements (`dl`, `dt`, and `dd`) may carry the attributes described in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

(dt | dd)+

The `dt` element is used for each terms, and the `dd` element contains the definition. Usually, inside a `dl` list, you will use the sequence `dt`, `dd`, `dt`, `dd`, and so on. However, in general, you can use any mixture of these two elements in any order.

dt

Enclose each term in a `dt` element. The content of this element may be any mixture of text and inline elements; see Section 10, “Inline content: `Inline.model`” (p. 26).

dd

Enclose each definition in a `dd` element. The definition itself must be one or more block elements; see Section 9, “The block elements” (p. 19).

As an example, here is the above list of sushi ingredient definitions in XHTML:

```
<dl>
  <dt>akagai</dt>
  <dd>
    <p>
      Red clam.
    </p>
  </dd>
  <dt>sawagani</dt>
  <dd>
    <p>
      A tiny freshwater crab, usually grilled and
```

```
        eaten whole.  
    </p>  
</dd>  
</dl>
```

9.11. The hr element: horizontal ruled line

To draw a horizontal rule (black line) across the page as a separator, use an empty hr element. Content model:

```
element hr  
{ Common.attrib,  
  empty  
}
```

Common.attrib

An hr element can carry any of the attributes discussed in Section 15.3, “The common attributes: Common.attrib” (p. 55).

In particular, if you want to change the color or thickness or other appearance of a rule, you can give it a `class` attribute that ties it to a CSS stylesheet rule. See Section 5, “Separating content and presentation with CSS” (p. 7).

empty

No content is allowed inside an hr element. Encode it as `<hr/>`.

10. Inline content: Inline.model

In our content models, the abbreviation `Inline.model` refers to any content that can be placed inside a block. Here is its content model:

```
Inline.model = { text | Inline.class }*  
Inline.class =  
{ a | abbr | acronym | cite | code | del | dfn | em | img | ins |  
  kbd | object | q | samp | span | strong | sub | sup | var  
}
```

That is, inline content may be ordinary text (not enclosed in any element), or one of the inline elements such as `code` and `em`, or any mixture of text and inline elements in any order.

Note

Except where otherwise indicated, the content model of each inline element is:

```
{ Common.attrib,  
  Inline.model  
}
```

That is, they can bear any of the attributes listed in Section 15.3, “The common attributes: Common.attrib” (p. 55), and they can contain any mixture of text and other inline elements.

- Section 10.1, “a: Hyperlink” (p. 27).

- Section 10.2, “abbr: Abbreviation” (p. 28).
- Section 10.3, “acronym: Acronym” (p. 29).
- Section 10.4, “cite: Title of a work” (p. 29).
- Section 10.5, “code: Part of a computer program” (p. 29).
- Section 10.6, “del: Deleted material” (p. 29).
- Section 10.7, “dfn: Definition of a term” (p. 30).
- Section 10.8, “em: Emphasis” (p. 30).
- Section 10.9, “img: Include an image” (p. 30)
- Section 10.10, “ins: Inserted material” (p. 31).
- Section 10.11, “kbd: Keyboard input” (p. 31).
- Section 13, “The object element: Embedded multimedia and applet objects” (p. 38).
- Section 10.12, “q: Inline quotations” (p. 31).
- Section 10.13, “samp: Sample computer output” (p. 31).
- Section 10.14, “span: The generic inline container” (p. 32).
- Section 10.15, “strong: Strong emphasis” (p. 32).
- Section 10.16, “sub: Subscript” (p. 32).
- Section 10.17, “sup: Superscript” (p. 32).
- Section 10.18, “var: Variable name” (p. 32).

10.1. a: Hyperlink

The **a** element is the workhorse of the Internet: it allows you to link to another document. For example, if you want to link to the New Mexico Tech web page, and use the text “NMIMT” as the link text, encode it like this:

```
<a href="http://www.nmt.edu/">NMIMT</a>
```

Here is the full content model:

```
element a
{ attribute href { xsd:anyURI }?,
  attribute charset { text }?,
  attribute type { text }?,
  attribute hreflang { text }?,
  attribute rel { text }?,
  attribute rev { text }?,
  attribute tabindex { xsd:nonNegativeInteger }?,
  attribute onblur { text }?,
  attribute onfocus { text }?,
  Common.attrib,
  Inline.model
}
```

href

To make a hyperlink to another location, use that location's URI as the value of this attribute. Your link can point to the beginning of another document, a named location inside another document, or a named location inside the same document. For the rules governing URIs, see Section 6.4, "URIs and link targets: where hyperlinks point" (p. 9).

charset

Describes the character set of the link's target. For legal values, see Section 15.2, "The `charset` attribute: Declaring a character set" (p. 55).

type

Describes the MIME type of the document at the link's target. See Section 6.7, "MIME types: Defining a resource's format" (p. 10).

hreflang

Describes the language of the document located at the target of the `href` attribute. For language codes, see Section 15.1, "The `xml:lang` attribute" (p. 55).

rel

Describes the relationship of this document to the one at its target. See Section 6.5, "The link datatype" (p. 9).

rev

Describes the relationship of the target document to this document—the reverse relationship to that given by the `rel` attribute. See Section 6.5, "The link datatype" (p. 9).

tabindex

This attribute specifies how this hyperlink is traversed in the tabbing order. See Section 15.7, "The `tabindex` attribute: Specifying tab traversal order" (p. 57).

onblur, onfocus

See Section 16, "Event attributes" (p. 57).

Common.attrib

You can use any of the attributes from Section 15.3, "The common attributes: `Common.attrib`" (p. 55).

For example, of particular utility is the `id` attribute. You can attach an `id` attribute to any element in the document, and link to that element using a fragment identifier. For a discussion of fragment identifiers, see Section 6.4, "URIs and link targets: where hyperlinks point" (p. 9).

Inline.model

The content of the `a` element can be any mixture of text and inline elements; see Section 10, "Inline content: `Inline.model`" (p. 26).

Everything inside of this element will act as a hyperlink: text, images, or whatever.

10.2. `abbr`: Abbreviation

It is good practice to enclose abbreviations in an `abbr` element. You can help your readers by providing the expanded form of the abbreviation as a `title` attribute. Here's an example:

```
<abbr title="species">sp.</abbr>
```

10.3. acronym: Acronym

An acronym is an abbreviation made from the initial letters of several words. As with the `abbr` element, you can help your readers by providing the expanded form of the acronym as a `title` attribute. Example:

```
<acronym title="Problem Exists Between Keyboard And Chair" >PEBKAC</abbr>
```

10.4. cite: Title of a work

Use this element to enclose the names of books, movies, and other works whose names are generally italicized. Example:

```
My favorite movie is <cite>Blazing Saddles</cite>.
```

10.5. code: Part of a computer program

Mark up excerpts from computer programs, such as variable names and keywords, with a `code` element. Example:

```
Next we call the <code>dwimify()</code> function.
```

10.6. del: Deleted material

To show that text is being deleted in this version, enclose the deleted text in a `del` tag. Here is its content model:

```
element del
{ attribute cite { xsd:anyURI },
  attribute datetime { text },
  Common.attrib,
  Inline.model
}
```

cite

If you would like to document the reasons for this deletion, you can point to that documentation by including its URI as the value of this attribute.

datetime

The date and time of the insertion. The preferred timestamp format is given in the *W3C Date and time formats* recommendation¹².

Common.attrib

As with all inline elements, you can use the attributes from Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Example:

```
See, my comb is green, and yours is orange.
<del>Now let us go out and get some womens.</del>
```

¹² <http://www.w3.org/TR/NOTE-datetime>

10.7. dfn: Definition of a term

When you first define a term, enclose that term in a `dfn` inline. Example:

```
This technique is called <dfn>kriging</dfn>.
```

10.8. em: Emphasis

Use the `em` element to emphasize text. Generally, this element is presented in italics. Example:

```
You can't <em>make up</em> stuff like this.
```

10.9. img: Include an image

Use the `img` element to place photos, diagrams, or other artwork on your page. The image must be available at a location given by some URI.

Note

This element is deprecated in favor of the more generalized `object` element. See Section 13, “The `object` element: Embedded multimedia and applet objects” (p. 38).

Here is the content model:

```
element img
{ attribute src { xsd:anyURI },
  attribute alt { text },
  attribute longdesc { xsd:anyURI }?,
  attribute height { text }?,
  attribute width { text }?,
  Common.attrib,
  empty
}
```

src

Use the URI of the image file as the value of the `src` attribute.

alt

You should always provide a textual description of the image so blind readers will know what they aren't seeing.

longdesc

To make available a more complete description of the image, place that description at some URI and use that URI as the value of the `longdesc` attribute.

height

If you don't provide a `height` attribute, the height of the image will be its natural size. You can override that height by specifying this attribute as a length; see Section 6.1, “The length datatype” (p. 8).

width

As with the `height` attribute, you can override the image's natural width by providing a `width` attribute whose value is specified as described in Section 6.1, “The length datatype” (p. 8).

Common.attrib

This element supports all the attributes from Section 15.3, “The common attributes: Common.attrib” (p. 55).

empty

No content is allowed inside an `img` element.

Here's an example:

```

```

10.10. ins: Inserted material

To show that material is being inserted in the current version, you can wrap the insertion in an `ins` element. This element also allows the `cite` and `datetime` attributes described in Section 10.6, “Deleted material” (p. 29), so you can document the time of the insertion or point to an online description of information about the insertion.

10.11. kbd: Keyboard input

Use this inline element to show that its contents are text entered by a user.

```
Type <kbd>run</kbd> to start Tinguely's self-destroying robot.
```

10.12. q: Inline quotations

For quoting text within a paragraph, wrap the quoted text in a `q` element. Quote marks will be added for you; you don't have to include your own. Here's an example:

```
Don't ever say <q>Don't ever say <q>Don't ever say  
that.</q></q>
```

The browser will even handle quotes within quotes correctly, shifting from double to single quotes. The above example will be rendered as:

Don't ever say “Don't ever say ‘Don't ever say that.’”

```
As the Tasmanians say, <q xml:lang='fr'>C'est la vie.</q>
```

In the above example, the `xml:lang` attribute specifies that the quote itself is in French.

The `q` element also allows a `cite` attribute so that you can link to a URI. For example, you may want to link the quote back to its source on the Web. Example:

```
<acronym>GIGO</acronym> stands for <q  
cite='http://catb.org/~esr/jargon/html/G/GIGO.html'>Garbage  
In, Garbage Out</q>.
```

10.13. samp: Sample computer output

Enclose sample computer output in a `samp` element. Example:

```
If it succeeds, the program will print "<samp>No errors</samp>".
```

10.14. span: The generic inline container

This inline element carries no specific meaning of its own. Its primary purpose is to serve as a generic inline text container that can be tagged with a `class` or `id` attribute so that a CSS stylesheet can specify how that text is to be formatted. For example, suppose you want keywords in your document to be formatted in red, boldface type. In your XHTML, you would enclose those keywords in something like this:

```
<span class='keyword'> ... </span>
```

Then you would link your page to a CSS stylesheet that uses the selector “`span.keyword`” to specify red, boldface type.

10.15. strong: Strong emphasis

This element is like `em` (emphasis), except that it connotes strong emphasis. Generally this element is presented in **boldface** type.

10.16. sub: Subscript

Subscript: text inside this element is displayed in a smaller font and shifted down below the baseline. Example: to get “Drink H₂O every day”, type:

```
Drink H<sub>2</sub>O every day.
```

10.17. sup: Superscript

Superscript: text inside this element is displayed in a smaller font and shifted above the baseline. Example: to get “Canada does indeed have a July 4th”, type:

```
Canada does indeed have a July 4<sup>th</sup>.
```

10.18. var: Variable name

Use the `var` element to enclose names of variables (algebraic or program variables, or arguments to a function). Example:

```
First set <var>x</var> to 0.
```

11. Tables: the table element

XHTML tables allow you to arrange content in rows and columns. Before we dive into the general case, here is a small complete example table showing the state names, state capitals, and state birds of two states.

```
<table>
  <tr>
    <td>New Mexico</td>
    <td>Santa Fe</td>
    <td>Greater Roadrunner</td>
```

```

</tr>
<tr>
  <td>Arizona</td>
  <td>Phoenix</td>
  <td>Cactus Wren</td>
</tr>
</table>

```

This table will have two rows and three columns. The `table` element wraps around the whole table. Each row is contained in a `tr` (table row) element. Within a row, each `td` (table detail) element contains one “cell” of the table, that is, the content at the intersection of one row and one column.

Here is the content model for tables in general.

```

element table
{ attribute summary { text }?,
  attribute border { xsd:int },
  attribute frame
  { "void" | "above" | "below" | "hsides" | "lhs" | "rhs" |
    "vsides" | "box" | "border"
  }?,
  attribute rules { "none" | "groups" | "rows" | "cols" | "all" }?,
  attribute cellspacing { text }?,
  attribute cellpadding { text }?,
  Common.attrib,
  element caption
  { Common.attrib,
    Inline.model
  }?,
  (col* | colgroup*),
  ((thead?, tfoot?, tbody+) | tr+)
}

```

summary

Use this optional attribute if you would like to provide a textual summary of the table's contents. This is especially useful for blind readers.

border

To place a border N pixels wide around the whole table, use an attribute `border="N"`.

frame

To place a border around some sides of the table, use an attribute with one of these values:

<code>void</code>	Don't use any borders (the default value).
<code>above</code>	Place a border only along the top of the table.
<code>below</code>	Place a border only along the bottom.
<code>hsides</code>	Place borders on the top and bottom.
<code>vsides</code>	Place borders on the left and right sides.
<code>lhs</code>	Place a border only on the left.
<code>rhs</code>	Place a border only on the right.
<code>border</code>	Place a border around all four sides.
<code>box</code>	Same as <code>border</code> .

rules

Use this attribute to specify when ruled lines are drawn between the cells of a table. Values:

<code>none</code>	Don't use ruled lines between cells (the default value).
<code>groups</code>	Place ruled lines after the table head (<code>thead</code>) if there is one, before the table footer (<code>tfoot</code>) if there is one, and between column groups specified with <code>colgroup</code> .
<code>rows</code>	Place ruled lines between rows.
<code>cols</code>	Place ruled lines between columns.
<code>all</code>	Place ruled lines between rows and between columns, forming a complete grid.

cellspacing

By default, the cells in the table are packed tightly together. You can use the table's `cellspacing` attribute to add extra spacing between the cells, horizontally and vertically. The value of this attribute is a length; see Section 6.1, "The length datatype" (p. 8).

cellpadding

By default, the size of the contents of each cell in the table are determined by what is in the cell. The `align` and `valign` attributes determine where the contents are shifted if they do not completely fill the cell. For example, if a column is 2cm wide, the column uses `align="left"`, and one particular cell is 1cm wide, its contents are placed against the left side of the cell, and the extra space will appear on the right.

The purpose of the `cellpadding` attribute of a table is to put a little extra space on all four sides of the content of a cell. The value of this attribute is a length; see Section 6.1, "The length datatype" (p. 8). For example, `cellpadding='4px'` would make sure there is a blank space inside the perimeter of each cell, four pixels wide.

Common.attrib

You can use any of the attributes from Section 15.3, "The common attributes: `Common.attrib`" (p. 55).

caption

To add a caption to the table, use this optional element. It allows the usual common attributes (Section 15.3, "The common attributes: `Common.attrib`" (p. 55)), and its content is any mixture of text and inline elements.

(`col*` | `colgroup*`)

To specify certain properties of the columns of your table, use either a sequence of `col` elements or a sequence of `colgroup` elements. See Section 11.1, "Specifying table column properties" (p. 34).

((`thead?`, `tfoot?`, `tbody+`) | `tr+`)

In a simple table, encode the rows as a series of `tr` (table row) elements. See Section 11.3, "Table rows: the `tr` element" (p. 36).

For large tables, you can structure the table as a heading section, a footer section, and one or more body sections. See Section 11.2, "Sectioning a table with `thead`, `tbody`, and `tfoot`" (p. 35).

11.1. Specifying table column properties

There are two ways you can describe the properties of the columns in your table. You can provide a series of `col` elements, each of which describes one column. If there are lots of columns, and some groups of them share properties, you can instead provide a series of `colgroup` elements that describe each group of columns. In that case, inside each `colgroup` element is an optional sequence of `col` elements further describing the columns in that group.

Here is the content model for the `colgroup` element.

```
element colgroup
{ col.attlist,
  col*
}
col.attlist =
  align.attrib?,
  valign.attrib?,
  attribute span { text }?,
  attribute width { text }?
align.attrib =
  attribute align { "left" | "center" | "right" | "justify" }
valign.attrib =
  attribute valign { "top" | "middle" | "bottom" | "baseline" },
  Common.attrib
col = element col
{ col.attlist,
  empty
}
```

span

If you want to describe several adjacent columns, use an attribute `span='N'` where *N* is the number of columns. If you do this, you won't need a sequence of `col` elements inside the `colgroup`.

width

Use this attribute if you want to specify the width of the column. Its value is a length; see Section 6.1, "The length datatype" (p. 8).

align

Specifies the horizontal positioning. By default, the content of each cell in the table is left-aligned (except for `th` elements, which are centered by default). To override these values, use `align='left'` to left-justify the content; use `align='center'` to center it; use `align='right'` to right-justify it; and use `align='justify'` to format the content as a paragraph with straight left and right margins.

valign

Specifies the vertical positioning. By default, the content of each cell is centered (`valign="middle"`). To set up a default vertical position for the column group or column, use `valign="top"` to push it to the top, `valign="middle"` for vertical centering, or `valign="bottom"` to push it down to the bottom. The `valign="baseline"` option lines up the text baselines of the cells in that column with the baselines of the other cells in the row.

Common.attrib

See Section 15.3, "The common attributes: `Common.attrib`" (p. 55).

col

The `col` element uses the same list of attributes.

11.2. Sectioning a table with `thead`, `tbody`, and `tfoot`

The body of a table may consist of a sequence of `tr` (table row) elements. However, in this case, if the table is printed, any headings or footers of the table will appear only once.

If your table is large and you would like a more usable printed rendering, structure the table as a `thead` element containing the heading row or rows, followed optionally by a `tfoot` element containing the footer row or rows, followed by one or more `tbody` elements with the main part of the table. If you use this structure, a printed rendering of the table will repeat the heading and footer on each page.

As an example, here is a slightly more elaborate version of the state capital and bird table above, using this triune structure.

```
<table>
  <thead>
    <tr>
      <th>State name</th>
      <th>State capital</th>
      <th>State bird</th>
    </tr>
  </thead>
  <tbody>
    <tr>
      <td>New Mexico</td>
      <td>Santa Fe</td>
      <td>Greater Roadrunner</td>
    </tr>
    <tr>
      <td>Arizona</td>
      <td>Phoenix</td>
      <td>Cactus Wren</td>
    </tr>
  </tbody>
</table>
```

11.3. Table rows: the `tr` element

Enclose each row of your table in a `tr` element. Here is the content model:

```
element tr
{ align.attrib?,
  valign.attrib?,
  Common.attrib,
  (th | td)+
}
```

align.attrib?, valign.attrib?

The `align` and `valign` attributes control the default horizontal and vertical positioning of the cells in the row; they are defined in Section 11.1, “Specifying table column properties” (p. 34).

Common.attrib

The common attributes are discussed in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

(th | td)+

Enclose each ordinary cell of the row in a `td` element. If the cell is a row heading, enclose it in a `th` element. See Section 11.4, “Table cells: the `td` and `th` elements” (p. 37).

11.4. Table cells: the td and th elements

A cell is where one row and one column intersect. The content of each cell is enclosed in a `td` (table detail) element, unless it is a column or row heading—those use a `th` element, which has the same structure. Here is the content model:

```
element td
{ Cell.attrib,
  Flow.model
}
element th
{ Cell.attrib,
  Flow.model
}
Cell.attrib =
  attribute rowspan { xsd:positiveInteger }?,
  attribute colspan { xsd:positiveInteger }?,
  align.attrib?,
  valign.attrib?,
  Common.attrib
```

rowspan

Normally each `td` or `th` element is placed in exactly one cell. However, you can merge multiple cells together to form a larger cell by using a `rowspan="N"` attribute, where *N* is the number of vertically adjacent cells to merge.

For example, an element “`<td rowspan='3'>...</td>`” would place the content of that `td` element into a wider space consisting of three cells—the current cell plus the cells from the same column in the next two rows that follow it.

colspan

This attribute works like `rowspan`, but it spans a horizontally adjacent group of cells into one. For example, if the cell in column 5 of a row was specified as `<td colspan='4'>...</td>`, its content would fill columns 5, 6, 7, and 8 of that row.

align.attrib?, valign.attrib?

These attributes control the horizontal and vertical placement content in the cell. For their definitions, see Section 11.1, “Specifying table column properties” (p. 34).

Common.attrib

Any of the attributes from Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

12. Flow.model: Arbitrary content

The content of a number of XHTML elements can be any mixture of text, inline elements, and block elements. Here is the content model of the abbreviation `Flow.model`:

```
Flow.model = ( text | Inline.model | Block.class )*
```

For example, the content of a bullet element (`li`) inside a bullet list (`ul`) doesn't have to be enclosed in any additional elements, but if you wish, you can enclose it in any number of inline or block elements. See Section 9.7, “The `ul` element: Unnumbered or “bullet” lists” (p. 23).

13. The object element: Embedded multimedia and applet objects

Use XHTML's `object` element to embed images, multimedia objects, and *applets* (applications that are intended to be displayed in a rectangular area of your document). The `object` element is an inline element.

Here is the content model:

```
element object
{ attribute classid { xsd:anyURI }?,
  attribute codebase { xsd:anyURI }?,
  attribute data { xsd: anyURI }?,
  attribute type { text }?,
  attribute codetype { text }?,
  attribute standby { text }?,
  attribute declare { "declare" }?,
  attribute height { text }?,
  attribute width { text }?,
  attribute name { text }?,
  attribute tabindex { xsd:nonNegativeInteger }?,
  Common.attrib,
  param*,
  Flow.model
}
```

classid

This URI points to an application to be run. The application will supply the content to be displayed in the document.

codebase

If supplied, this attribute gives the base URI used to expand any relative URI references in the `classid` and `data` attributes.

data

If your application needs to know where its data lives, you can supply the URI of that data in this attribute.

type

Use this attribute to specify the MIME type of the data specified by the `data` attribute. See Section 6.7, "MIME types: Defining a resource's format" (p. 10).

codetype

Use this attribute to specify the MIME type of the application specified in the `classid` attribute. See Section 6.7, "MIME types: Defining a resource's format" (p. 10).

standby

If you think your application might take more than a fraction of a second to load, you may want to set the `standby` attribute to some text that will be displayed while your application is loading. For example: `standby="Loading..."`.

declare="declare"

Normally, the browser will execute your application as soon as your page is loaded. However, if you use the `declare="declare"` attribute, the browser will remember the `id` of the object and defer execution until some other object refers to that `id` later. See Section 13.2, "How to delay instantiation of an object" (p. 40).

height, width

You can override the inherent dimensions of the object by supplying the height and width of the object's area in these attributes. These attributes are deprecated. Use CSS instead; see Section 5, "Separating content and presentation with CSS" (p. 7).

name

If this object is part of a form, use the `name` attribute to give a "control name" to the object on the form. See Section 14, "Forms: The `form` element" (p. 41).

tabindex

You can use this attribute to specify when the object will get focus when the user uses the `tab` key. See Section 15.7, "The `tabindex` attribute: Specifying tab traversal order" (p. 57).

Common.attrib

You can use any of the attributes from Section 15.3, "The common attributes: `Common.attrib`" (p. 55).

param*

You can pass arguments to the application by supplying one or more `param` elements. See Section 13.1, "The `param` element: Passing arguments to applications" (p. 39).

Flow.model

The content of an `object` element can be any mixture of text, inline objects, and block objects. See Section 12, "Flow.model: Arbitrary content" (p. 37).

For a single, static object such as a GIF image, simply use a `classid` attribute that points to the image. For blind readers, supply some plain text content inside the element that will be displayed when images can't be rendered. Here's an example:

```
<object data="grundoon.png" type="image/png">
  A drawing of Grundoon Groundchuck.
</object>
```

In general, the content of an `object` will often be another `object` element. The rule here is that the browser will try to render the outer `object` first; if that fails, it will render the content.

When you are including embedded applications in your Web page, keep in mind that not all applications will work in all environments. Hence, it is good practice to use a nested series of `object` elements, with each inner element more likely to render correctly. Here's an example: the browser will try first to run the Python application `zoot.py`. If it can't do that, it tries to display an animation in MPEG format. If that isn't possible either, it tries to display a PNG image. Finally, if even displaying an image is impossible (as for a blind reader), it displays the text "A picture of Zoot".

```
<object classid="http://www.anthrax.edu/cgi-bin/zoot.py">
  <object data="zoot.mpeg" type="application/mpeg">
    <object data="zoot.png" type="image/png" >
      A picture of Zoot.
    </object>
  </object>
</object>
```

13.1. The `param` element: Passing arguments to applications

The purpose of the `param` element is to pass parameter values to an `object` element. This is necessary only when the object is a script that expects those values to be supplied to it. Any number of named values may be passed to the object, one in each `param` element.

Here is the content model:

```
element param
{ attribute name { text },
  attribute value { text }?,
  attribute valuetype { 'data' | 'ref' | 'object' }?,
  attribute type { text }?,
  attribute id { xsd:ID }?,
  empty
}
```

name

Set this attribute to the name of the parameter you want to pass to the object.

value

Use this attribute for the value that you are passing to the object.

valuetype

This attribute specifies what kind of value you are passing. It must be one of these:

data	The value attribute is a string that will be passed to the application.
ref	The value is a URI to be passed to the application as a string.
object	Use this form to pass a reference to an object element as a parameter. For an explanation of this technique, see Section 13.2, “How to delay instantiation of an object” (p. 40).

type

When you use **valuetype='ref'**, set the **type** attribute to the MIME type of the resource at the URI specified by the **value** attribute. See Section 6.7, “MIME types: Defining a resource's format” (p. 10) for permissible values.

id

Use this attribute to attach a unique identifier to the object. This is required when you are passing an object to an object: see Section 13.2, “How to delay instantiation of an object” (p. 40).

empty

No content is allowed in a **param** element.

Here's an example. Suppose you have a map-viewer applet named `mapview.py` that displays a topographic map. Further suppose that it is expecting a variable named `start-quad` to contain the name of the map where it starts. This example would pass the string “Water Canyon, NM” to the applet:

```
<object classid="mapview.py">
  <param name='start-quad' value='Water Canyon, NM' />
  [text to be displayed if the map viewer cannot be rendered]
</object>
```

13.2. How to delay instantiation of an object

Unless you specify otherwise, your object will be “instantiated” (evaluated and turned into Web content) when your page is loaded.

Instead, you can delay instantiation of your object until later:

- You can set up a link so that the object is not instantiated until the reader clicks on that link.
- You can use this object as a parameter to a different object.

To delay instantiation of an object, use the `declare="declare"` attribute in the `object` element, and give it a unique identifier with the `id` attribute.

To refer to the object in a link:

```
<a href="#idref">...</a>
```

where the *idref* is the `id` attribute of the previously declared object. For example:

```
<object classid="mapview.py" id="mapper" declare="declare">
  <param name='start-quad' value='Water Canyon, NM' />
</object>
...
<a href="#mapper">Map of Water Canyon</a>
```

In the above example, the page will have a link with the text “Map of Water Canyon”. The map viewer will not run until the user clicks on that link.

Suppose you want to pass an object as a parameter to another object. Use this `param` element in the second object:

```
<param name='pname' value='#oid' valuetype='object' />
```

where *#oid* is the `id` attribute of the object to be passed in.

14. Forms: The form element

Your document can contain one or more *forms* that the user can fill out.

- A form contains a mixture of regular text and *controls* such as checkboxes, text fields, and buttons.
- Every control must have a *control name*, specified by its `name` attribute. Control names have meaning only inside their form; you may reuse control names on different forms of the same document.
- Each form must have a *Submit* button that gathers up the values from the controls and sends them to your *handler script* for processing.
- When the user clicks on the *Submit* button, the data value from each *successful* control is sent to the handler script as name/value pair, where the name is the value of the control's `name` attribute, and the value represents what appears on the user's screen at the time of submission. For the definition of a successful control, see Section 14.9, “What makes a control successful?” (p. 53).

Each form is enclosed in a `form` block element with this content model:

```
element form
{ attribute action { xsd:anyURI },
  attribute method { 'get' | 'post' }?,
  attribute enctype { text }?,
  attribute onsubmit { text }?,
  attribute onreset { text }?,
  Common.attrib,
  ( fieldset | Block.class )+
}
```

action

Set the value of this attribute to point at a script that will be executed when the user clicks on the form's *Submit* button. For example, if this is a survey form, you might write a handler script named `survey.cgi`, and point it at with the attribute `action='survey.cgi'`.

method

This attribute selects how values are passed to the handler script. The value must be either `'get'` or `'post'`. For details, see Section 14.10, “Writing your form handler script” (p. 53).

enctype

When you use `method='post'`, this attribute specifies which protocol you are using to transmit the form values. See Section 14.10, “Writing your form handler script” (p. 53).

onsubmit, onreset

Actions to be taken when the form is submitted or reset. See Section 16, “Event attributes” (p. 57).

Common.attrib

Form elements can carry all the usual attributes discussed in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

(fieldset | Block.class)+

Your form must be structured as a sequence of one or more block elements; see Section 9, “The block elements” (p. 19).

You can also structure your form as a sequence of `fieldset` elements. See Section 14.8, “The `fieldset` element: Adding structure to a form” (p. 52).

The various controls that can appear on a form are described in succeeding sections:

- Section 14.1, “The `input` forms control” (p. 42).
- Section 14.3, “The `button` forms control” (p. 48).
- Section 14.4, “The `select` forms control: menus” (p. 48).
- Section 14.7, “The `textarea` forms control: multiline text input” (p. 51).
- You can use an embedded image or applet as a control. You must give the object a control name by using the `name` attribute in the `object` element. See Section 13, “The `object` element: Embedded multimedia and applet objects” (p. 38).

14.1. The `input` forms control

An input element may represent one of several types of forms controls. Here is the content model:

```
element input
{ attribute name { text },
  attribute type
  { 'text' | 'password' | 'checkbox' | 'radio' | 'file' | 'submit'
  'image' | 'reset' | 'hidden'
  }?,
  attribute value { text }?,
  attribute checked { 'checked' }?,
  attribute size { xsd:nonNegativeInteger }?,
  attribute maxlength { xsd:nonNegativeInteger }?,
  attribute alt { text }?,
  attribute tabindex { xsd:nonNegativeInteger }?,
  attribute onchange { text }?,
```

```
attribute onselect { text }?,  
attribute onfocus { text }?,  
attribute onblur { text }?,  
Common.attrib,  
empty  
}
```

name

Use this attribute to give a control name to the element. The name must conform to the naming rules described in Section 6.2, “The ID datatype” (p. 8).

type

Selects the type of control. See the sections for the various control types below.

value

This attribute specifies a value for the element. Refer to the table of control types below for its usage.

checked= ' checked'

For checkboxes, specify this attribute if you want the checkbox to be set (on) initially.

For radiobuttons, specify this attribute for the radiobutton that is on initially by default.

size

For text and password fields, this specifies the displayed size of the field as a number of characters. For example, `size= '10'` would display a ten-character text field.

maxlength

For text and password fields, this specifies the *capacity* of the field. If the capacity exceeds the size, the field will scroll horizontally as the user enters more text than the displayed size.

alt

If the control is a graphic, use this attribute to supply alternate text for non-graphic users.

tabindex

See Section 15.7, “The `tabindex` attribute: Specifying tab traversal order” (p. 57).

onchange, onselect, onfocus, onblur

See Section 16, “Event attributes” (p. 57).

Common.attrib

You can attach to an `input` element any of the attributes described in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

empty

No content is allowed inside an `input` element.

Here are the sections describing the various `input` controls.

- Section 14.1.1, “The `text` control” (p. 44).
- Section 14.1.2, “The `password` control” (p. 44).
- Section 14.1.3, “The `checkbox` control” (p. 44).
- Section 14.1.4, “The `radio` control: `radiobutton`” (p. 44).
- Section 14.1.5, “The `file` control: Uploading user files” (p. 45).
- Section 14.1.6, “The `submit` control” (p. 45).
- Section 14.1.7, “The `image` control: A graphical submit button” (p. 46).

- Section 14.1.8, “The reset control: Form clear button” (p. 46).
- Section 14.1.9, “The hidden control” (p. 47).

14.1.1. The text control

Creates a field where the user can enter one line of text from the keyboard. Use the `size` attribute to specify the size of the field, in characters; the default is 20. Use the `maxLength` attribute if you want to limit how much text the user can enter.

Here's an example of a text field and its associated label. See Section 14.2, “The label element: Label a control” (p. 47).

What is your favorite color?

```
<input type="text" name="fave-color" id="fave-text"/>
<label for="fave-text">What is your favorite color?</label>
```

14.1.2. The password control

A password field works the same as a text field, except that each character entered by the user is displayed as “*”, so that no one looking at the browser window can see what text has been entered. An example:

Enter your secret name

```
<input type="password" name="pwd" id="pwd-field"/>
<label for="pwd-field">Enter your secret name</label>
```

14.1.3. The checkbox control

Creates an unlabeled checkbox. If the control has a `value` attribute, that value is passed to the handler script as the value for this control's name. If no `value` is given, the value “on” will be sent to the handler.

If the control has `checked= ' checked '`, the checkbox will be set (on) initially. Otherwise, it will initially be cleared (off).

Here is an example:

Does your fish have a license?

```
<input type="checkbox" name="fish-license" id="eric"/>
<label for="eric">Does your fish have a license?</label>
```

14.1.4. The radio control: radiobutton

Radiobuttons allow the user to select one of a set of mutually exclusive choices. All the radiobuttons in a set have the same `name` attribute, but each has a different `value` attribute. When the form is submitted, the value of the currently selected radiobutton will be sent to the handler script.

You should specify `checked= ' checked '` for one of the radiobuttons in a set; that radiobutton will be set initially.

Here's an example of a set of three radiobuttons.

Who should investigate the moaning noises?

Velma

Shaggy

Scooby

The label above the radiobuttons is ordinary text. Note that each `input` element has the same control name (`name` attribute). Each radiobutton and its label are placed on a separate line by wrapping them inside a `div` element.

```
Who should investigate the moaning noises?
<div>
  <input type="radio" name="mystery" id="velma" value="Velma"
    checked="checked" />
  <label for="velma">Velma</label>
</div>
<div>
  <input type="radio" name="mystery" id="shag" value="Shaggy" />
  <label for="shag">Shaggy</label>
</div>
<div>
  <input type="radio" name="mystery" id="scoob" value="Scooby" />
  <label for="scoob">Scooby</label>
</div>
```

14.1.5. The file control: Uploading user files

This control places two elements on a form: a text field, and a *Browse* button. The user can upload a file by clicking on the button and navigating to the file using his browser's customary file-browser facility.

Here is an example:

```
<input type="file" name="user-file"/>
```

Note
When using this kind of control, be sure that the containing `form` element has an attribute `“enc-type=“multipart/form-data”`.

14.1.6. The submit control

This control creates a *Submit* button on the form. Use the `value` attribute to specify the text that will appear on the button; the default text label is `“Submit”`.

For an example, see Section 14.1.8, *“The reset control: Form clear button”* (p. 46).

14.1.7. The image control: A graphical submit button

An `input` element with a `type='image'` attribute creates a graphic *Submit* button. The `src` attribute points to the image file.

When the user clicks on this image, the form is submitted, and *two* name/value pairs are sent to the handler script. The first name/value pair contains the x coordinate where the user clicked on the image, relative to the top left corner; the name for this coordinate is the name from the `name` attribute with `".x"` appended.

Similarly, the y coordinate where the user clicked is sent to the handler; the name is the control name with `".y"` appended.

For example, suppose an element is defined like this:

```
<input type='image' name='target' src='bullseye.png' />
```

The graphic `bullseye.png` is displayed in the form.

If the user were to click on that graphic at a spot that is 39 pixels from the left side of the graphic, and 7 pixels from the top, the handler script would receive two name/value pairs: `target.x=39` and `target.y=7`.

Note

Use of the `type='image'` control is not recommended, because blind users cannot use it. If you want to use graphic buttons but give the user choices, use multiple *Submit* controls (either `type='submit'` or `type='image'`), and give each of them a different name. The handler script will be able to tell from the control name which *Submit* button was used. See also Section 14.3, "The `button` forms control" (p. 48).

14.1.8. The reset control: Form clear button

Creates a *Reset* button on the form. Use the `value` attribute to specify the text that will appear on the button; the default text label is "Reset".

The purpose of a *Reset* button is to restore the controls on the form to the values they have when the form is first displayed. In particular, a text control will have its `value` attribute filled in (if it has one); any radiobutton or checkbox with the `checked='checked'` attribute will be set, and others cleared; and so on.

Here is an example of a submit button and a reset button packed together on the same line by placing them inside a `div` element.



```
<div>
  <input type="submit" value="Send" />
  <input type="reset" value="Clear" />
</div>
```

14.1.9. The hidden control

You can pass data to the handler script without displaying it on the form by using an `input` element with `type='hidden'`. Use the `value` attribute to specify the data to be passed to the handler script.

14.2. The `label` element: Label a control

The purpose of this element is to attach a clickable label to a control.

Why is ordinary text not enough to label a control? Consider, for example, this fragment of a form, showing a checkbox with the text “Cumin” next to it:

```
<div>
  <input type='checkbox' name='cumin' /> Cumin
</div>
```

The text “Cumin” identifies the checkbox, but the user can't click on that to set the checkbox. Only clicking on the checkbox itself sets it.

So the `label` element allows you to define a label that is associated with a form control. For checkboxes and radiobuttons, for example, the user can then set the control by clicking on the label as well as the checkbox or radiobutton symbol itself.

Here is the content model:

```
element label
{ attribute for { xsd:IDREF },
  attribute onfocus { text }?,
  attribute onblur { text }?,
  Common.attrib,
  Inline.model
}
```

for

To link this label to a control, define an `id` attribute on that control, and then include a `for` attribute on the label that specifies that same `id` value.

onfocus, onblur

Actions to be taken when the element gains or loses focus; see Section 16, “Event attributes” (p. 57).

Common.attrib

You can use any of the attributes from Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Inline.model

Inside the label you can use any mixture of text and inline elements; see Section 10, “Inline content: `Inline.model`” (p. 26).

Here is the above example redone with a `label`. The user will be able to set or clear the checkbox by clicking on the label.

```
<div>
  <input type='checkbox' name='cumin' id='cumin-cb' />
  <label for='cumin-cb'>Cumin</label>
</div>
```

14.3. The button forms control

The `button` element is rendered as a pushbutton. Unlike the `input` element, you can put content inside the element, such as text or images. Here is the content model:

```
element button
{ attribute name { text },
  attribute value { text }?,
  attribute type { 'submit' | 'reset' | 'button' }?,
  attribute disabled { 'disabled' }?,
  attribute tabindex { xsd:nonNegativeInteger }?,
  attribute onfocus { text }?,
  attribute onblur { text }?,
  Common.attrib,
  Flow.model
}
```

name

The button's control name.

value

The value to be sent to the handler script when this button is activated.

type

A value of `type='submit'`, the default value, creates this control as a submit button. A value of `type='reset'` creates a reset button. Using `type='button'` creates a pushbutton; use the event attributes to connect such a button with code to handle it. See Section 16, “Event attributes” (p. 57).

If the `type` attribute is omitted, the control is created as a submit button.

disabled

If you use `disabled='disabled'`, this control will not respond to user events.

tabindex

Specifies where this control lies in the tab traversal order. See Section 15.7, “The `tabindex` attribute: Specifying tab traversal order” (p. 57).

onfocus, onblur

See Section 16, “Event attributes” (p. 57).

Common.attrib

This control can carry any of the attributes described in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Flow.model

The content inside a `button` element can be any mixture of text, inline elements, and block elements. See Section 12, “`Flow.model`: Arbitrary content” (p. 37).

Here is an example of a button that displays image file `panic.png`, and submits the form when clicked.

```
<button name='panic-button' value='panic' type='submit'>
  <object data='panic.png' type='image/png' />
</button>
```

14.4. The select forms control: menus

This control allows the user to select one of a fixed set of choices. There are two general styles:

- A *drop-down menu* takes up very little space on the form. When the user clicks on it, a list of the choices will drop down.
- If you have more room, you can display a *scrollable list* of choices. You specify how many lines to show with the `size` attribute. If there are more choices than that, the control will have a scrollbar that allows the user to scroll through all the choices.

Here is the content model:

```

element select
{ attribute name { text },
  attribute size { xsd:positiveInteger }?,
  attribute multiple { 'multiple' }?,
  attribute tabindex { xsd:nonNegativeInteger }?,
  attribute onchange { text }?,
  attribute onfocus { text }?,
  attribute onblur { text }?,
  Common.attrib,
  ( option | optgroup ) +
}

```

name

This required attribute specifies the control's name.

size

If you omit this attribute, you will get a drop-down menu. To create a scrollable list, specify how many lines to display. If there are too many choices to display at once, the control will have a scrollbar.

multiple

By default, the user will be able to select only one of the choices. If you specify the `multiple='multiple'` attribute, the user will be able to select any number of them, or none at all. The method for selecting multiple choices depends on the browser. For example, in Firefox, clicking any choice selects it; control-click adds a choice to the selected set, or removes it if it was already selected.

If a user selects more than one choice, the handler script will receive a list of choices, rather than a single choice, for that element. If the user selects no choices at all, the control will not be considered successful, and the handler script will not receive a name/value pair for that control.

tabindex

See Section 15.7, “The `tabindex` attribute: Specifying tab traversal order” (p. 57).

onchange, onfocus, onblur

See Section 16, “Event attributes” (p. 57).

Common.attrib

This control can carry any of the attributes described in Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

(option | optgroup)+

There are two ways to organize the options inside a `select` control. You can include a sequence of `option` elements; each defines one choice. You can also organize the options into groups: the content inside the `select` element is a sequence of `optgroup` elements, each of which has a sequence of `option` elements inside it.

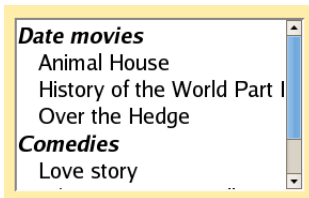
See Section 14.5, “The `option` element: One choice inside a `select` control” (p. 50) and Section 14.6, “The `optgroup` element: A group of choices inside a `select` control” (p. 51).

Here is an example of a single-level, drop-down `select` control with three mutually exclusive choices.



```
<select name='porridge-temp'>
  <option value='hot'>Too hot</option>
  <option value='cold'>Too cold</option>
  <option value='perfectamundo'>Just right</option>
</select>
```

Here's a more complex example. This control displays a list six lines high, and allows the user to select any number of different movies. The movies are grouped into genres.



```
<select name='movies' size='6' multiple='multiple'>
  <optgroup label='Date movies'>
    <option>Animal House</option>
    <option>History of the World Part I</option>
    <option>Over the Hedge</option>
  </optgroup>
  <optgroup label='Comedies'>
    <option>Love story</option>
    <option>When Harry met Sally</option>
    <option>On golden pond</option>
  </optgroup>
</select>
```

14.5. The option element: One choice inside a select control

Each option element inside a `select` control has this content model:

```
element option
{ attribute selected { 'selected' }?,
  attribute value { 'value' }?,
  Common.attrib,
  text
}
```

selected

If you don't specify this attribute on any of the choices in the `select` control, the first choice will be selected initially. If you want a different choice to be the default initial choice, add a `selected='selected'` attribute to that choice's `option` element.

value

If specified, the value of this attribute will be sent to the handler script for this choice. If you don't specify this attribute, the data sent to the handler will be the text content of the element.

For example, this choice would display as “Queen Elizabeth II”, but if selected, the string “QE2” would be sent to the handler:

```
<option value="QE2">Queen Elizabeth II</option>
```

Common.attrib

You can use any of the attributes from Section 15.3, “The common attributes: Common.attrib” (p. 55).

text

Place the text you want to display for this choice inside the element's tags.

14.6. The optgroup element: A group of choices inside a select control

Inside a `select` control, the `optgroup` element allows you to group options into categories. This can be useful when there are a lot of options. Instead of a sequence of `option` elements inside the `select` element, you use a sequence of `optgroup` elements, each of which in turn contains a sequence of `option` elements:

```
element optgroup
{ attribute label { text },
  Common.attrib,
  option+
}
```

label

This required attribute is the name of the category for this option group.

Common.attrib

Use any of the attributes described in Section 15.3, “The common attributes: Common.attrib” (p. 55).

option+

The content is a sequence of one or more `option` elements; see Section 14.5, “The option element: One choice inside a select control” (p. 50).

14.7. The textarea forms control: multiline text input

This element creates an area for displaying and allowing user entry of multiple lines of text. Its content model:

```
element textarea
{ attribute name { text },
  attribute rows { xsd:positiveInteger },
  attribute cols { xsd:positiveInteger },
  attribute onchange { text }?,
  attribute onselect { text }?,
  attribute onfocus { text }?,
  attribute onblur { text }?,
  Common.attrib,
```

```
text
}
```

name

This required attribute specifies the control's name.

rows

This attribute specifies how many lines of text are displayed; it is required.

cols

This attribute specifies the width of the displayed control in characters; it is required.

onchange, onselect, onfocus, onblur

See Section 16, "Event attributes" (p. 57).

Common.attrib

This element can carry any of the attributes from Section 15.3, "The common attributes: `Common.attrib`" (p. 55).

text

Any text you place inside this element will appear as the initial text inside the control.

Here is an example:

```
<textarea name='complaint' rows='5' cols='40'>
  Enter your complaint here.

  Then click 'Submit'.
</textarea>
```

14.8. The fieldset element: Adding structure to a form

If your form is large or complex, you may want to divide it up into multiple sections, perhaps also providing a descriptive label on each section. To do this, use a sequence of `fieldset` elements inside the `form` element. Here is the content model:

```
element fieldset
{ Common.attrib,
  legend,
  Flow.model
}
```

Common.attrib

This element can carry any of the attributes described in Section 15.3, "The common attributes: `Common.attrib`" (p. 55).

legend

The `legend` element contains the title for this field set. See below for its content model.

Flow.model

Inside the `fieldset` element you can use any combination of text, inline elements, and block elements.

Here is the content model for the `legend` element that titles each `fieldset` element:

```
element legend
{ Common.attrib,
```

```
Inline.model
}
```

Common.attrib

The usual attributes from Section 15.3, “The common attributes: `Common.attrib`” (p. 55).

Inline.model

The legend may consist of any mixture of text and inline elements; see Section 10, “Inline content: `Inline.model`” (p. 26).

Here's an example of the use of `fieldset`.

```
<form action="handler.cgi" method="post">
  <fieldset>
    <legend>Attitudes about fish</legend>
    ...form elements...
  </fieldset>
  <fieldset>
    <legend>Attitudes about birds</legend>
    ...form elements...
  </fieldset>
</form>
```

14.9. What makes a control successful?

When the user clicks on the *Submit* button of a form, name/value pairs are sent to the handler script for each successful control.

Here are the rules that define when a control is considered successful.

1. If a control is disabled, it cannot be successful.
2. Checkboxes that are set (on) are successful. Unset (off) checkboxes are not successful.
3. For a group of radiobuttons that have the same control name, only the one that is currently set is successful.

If you don't specify which of the radiobuttons in a group is initially set by default (with `checked='checked'`), the result is undefined. When you create a form with radiobuttons, always designate a default radiobutton.

4. For a `select` element, only the choice that is selected is considered successful.

14.10. Writing your form handler script

You can use many languages to write the form handler script that is run when the user clicks the *Submit* button on a form.

It is up to you to decide what to do with form data. You might e-mail it to somebody, or write it to a file, or add it to a database.

To write the handler script, you must know what controls are on the form, the name of each control, and what kind of data you expect.

There are three different protocols for transmitting the form data to the handler script. Use the `method` and `enctype` attributes on the `form` element to select one of these protocols:

- If you use `method='get'`, the name/value pairs from the form are encoded using *URL encoding*; see Section 14.11, “The URL encoding method for forms data” (p. 54). The URI of the document containing the form has a question mark (?) appended to it, followed by the encoded form data, and the resulting string is sent to the handler script.

Use this method only when the amount of form data is relatively small (less than 100 characters or so in encoded form), and when you don't care if the values from the form appear in the browser window.

- If you use `method='post'` and `enctype='application/x-www-form-urlencoded'`, the form values are encoded as a string, as described in Section 14.11, “The URL encoding method for forms data” (p. 54).
- If the volume of form data is quite large, or if you allow uploading of files (as described in Section 14.1.5, “The file control: Uploading user files” (p. 45), use `method='post'` and `enctype='multipart/form-data'`.

For the details of MIME multipart transmissions, see RFC 2045¹³.

Most of the popular scripting languages support retrieval of values from these protocols. For example, the Python language has a module called `cgi` that makes it trivial to retrieve form values transmitted by any of these three protocols.

14.11. The URL encoding method for forms data

When we say that the data from a form has been “URL-encoded”, it means that the data are reduced to a string of characters with this form:

- Each control's value is encoded as “*name=value*”.
- Name-value pairs are separated by ampersand (&) characters.
- Any character in the *value* part that is not a letter or digit is encoded as the three-character sequence “%xx” where xx is the hexadecimal code for the character in the ASCII character set.

For example, here is a small form with three elements.

```
<form action="plants.cgi" method="post">
  <div>Description: <input type='text' name='desc' /></div>
  <div>
    Sunlight:
    <div><input type='radio' name='sun' value='full' /> Full</div>
    <div>
      <input type='radio' name='sun' value='partial' />
      Partial sun
    </div>
    <div><input type='radio' name='sun' value='shade' /> Shade</div>
  </div>
  <div><input type='checkbox' name='toxic' value='yes' /> Toxic</div>
  <div><input type='submit' />
</form>
```

Suppose the user has typed “African violet” into the text field, set the “Shade” radiobutton, and set the “Toxic” checkbox. Three controls are considered successful, so three name/value pairs will be sent. In URL-encoded format, this would look like:

¹³ <http://www.ietf.org/rfc/rfc2045>

```
desc=African%40violet&sun=shade&toxic=yes
```

where the space is encoded using its hexadecimal value of "%40".

15. Standard attributes

This section describes some attributes that are used in many different places.

15.1. The `xml:lang` attribute

This attribute defines the language used in an element.

In general, codes can have one of two forms:

- `xml:lang="LL"`, where *LL* is a language code. These codes are defined by ISO 639¹⁴.
- `xml:lang="LL-CC"` where *LL* is a language code and *CC* is a country code. Country codes are defined by ISO 3166-1993¹⁵.

For example, use `xml:lang='en'` for English; use `xml:lang='en-us'` for United States English.

15.2. The `charset` attribute: Declaring a character set

This attribute is used to describe the character set of some resource. For character set names, see *IANA (Internet Assigned Numbers Authority) Official Names for Character Sets*¹⁶.

For example, here is a link to a page named `rowbazzle.html` that uses encoding ISO-8859-1:

```
<a href="rowbazzle.html" charset="ISO-8859-1">I go Pogo.</a>
```

15.3. The common attributes: `Common.attrib`

Wherever you see `Common.attrib` in an element's content model, that element can carry any of these attributes:

```
Common.attrib =  
  attribute id { xsd:ID }?,  
  attribute class { xsd:NMTOKENS }?,  
  attribute title { text }?,  
  attribute onclick { text }?,  
  attribute ondblclick { text }?,  
  attribute onkeydown { text }?,  
  attribute onkeypress { text }?,  
  attribute onkeyup { text }?,  
  attribute onmousedown { text }?,  
  attribute onmousemove { text }?,  
  attribute onmouseover { text }?,  
  attribute onmouseout { text }?,  
  attribute onmouseup { text }?
```

¹⁴ <http://www.oasis-open.org/cover/iso639a.html>

¹⁵ <http://www.oasis-open.org/cover/country3166.html>

¹⁶ <http://www.iana.org/assignments/character-sets>

- Section 15.4, “The `id` attribute: Assigning a unique identifier to an element” (p. 56).
- Section 15.5, “The `class` attribute: Declaring an element's CSS class” (p. 56).
- Section 15.6, “The `title` attribute: Titling an element” (p. 56).
- For all the attributes whose names start with `on-`, see Section 16, “Event attributes” (p. 57).

15.4. The `id` attribute: Assigning a unique identifier to an element

You can attach a unique identifier to any element on your page by giving that element an `id` attribute. The value must conform to the general XML rules for identifiers: the first character must be a letter, and any additional characters can be letters, digits, underbars (`_`), hyphens (`-`), or periods (`.`).

For example, suppose you are using a CSS stylesheet and you have a paragraph that contains a capsule biography of Frank Zappa. You can give this paragraph a unique identifier of “fz” by using the start tag “`<p id='fz'>`”. Then you can specify the appearance of this specific paragraph using the CSS selector “`p#fz`”.

The value of each `id` attribute in a document must be unique. You may not have two elements in the same document, for example, with attribute “`id='g14.7'`”.

15.5. The `class` attribute: Declaring an element's CSS class

Sometimes you want to change the appearance of certain elements on your page using a CSS stylesheet (see Section 5, “Separating content and presentation with CSS” (p. 7)). To do this, invent a name for those elements; this name must conform to the rules for names discussed in Section 6.2, “The ID data-type” (p. 8). Then, add a `class` attribute to those elements.

For example, suppose several paragraphs on your pages are important warnings, and you want to print them in larger type with a red border. To do this, add an attribute “`class='warning'`” to each paragraph. Then, your CSS stylesheet would use the selector `p.warning` with a rule that specifies larger type and a red border.

Any number of elements can have the same value of their `class` attributes, unlike `id` values which must be unique within a document (see Section 15.4, “The `id` attribute: Assigning a unique identifier to an element” (p. 56)).

An element can actually belong to several classes. In that case, use a `class` attribute whose value is a list of its classes separated by spaces. For example, the start-tag `<ul class="info Kyzyl Feynman"` would mean that the `ul` element is a member of three different classes.

15.6. The `title` attribute: Titling an element

You can attach a descriptive title to any element using a `title` attribute.

- Some browsers may display the value of this attribute as a “tool tip” when the mouse is over that element.
- The `title` attribute has specific meanings in these elements: Section 8.4, “The `link` element: Related documents” (p. 14), and the `abbr` and `acronym` elements, Section 10, “Inline content: `Inline.model`” (p. 26).

15.7. The `tabindex` attribute: Specifying tab traversal order

When the user presses a key on the keyboard, that key's character is routed to a specific element. That element is said to have the *focus*.

When a user presses the *tab* key, the “focus” moves through some of the elements on the page. If you don't otherwise specify, focus moves through the hyperlinks and form elements in the order you've specified them. If the focus is currently on a hyperlink, pressing the *Enter* key has the same effect as clicking the mouse on the link.

You can specify that elements are traversed in a different order by the *tab* key using the `tabindex` attributes of the links and form elements. Assign a different positive number to each element's `tabindex` attribute, and focus will traverse those elements from the lowest-numbered to the highest, and then through all the elements that don't have a `tabindex` attribute.

16. Event attributes

You can set up your document to run a script when various events occur, such as mouse clicks or keys struck on the keyboard. To do this, attach one of these *intrinsic event attributes* to an element; the value of the attribute is an expression in your scripting language that can handle that event.

To use these attributes, you must declare the scripting language in a `meta` element inside the page's `head` element. Here is the general form of this declaration:

```
<meta http-equiv='Content-Script-Type' content='T' />
```

where *T* is the MIME type of the scripting language. For example, the MIME type of JavaScript is `content='text/javascript'`. See Section 6.7, “MIME types: Defining a resource's format” (p. 10).

Here are the intrinsic event attributes, and the things that can happen to an element with that attribute that will cause the script to be run.

<code>onblur</code>	When an element loses focus. See the <code>onfocus</code> event below for a discussion of focus. We're not talking about actual blurriness here; it's just a play on words. If your eyes lose focus, what you see is blurry, right?
<code>onchange</code>	When a control on a form changes its value. To be precise, this happens only when the user moves focus onto the control, changes the value of an <code>input</code> , <code>textarea</code> , or <code>select</code> element, and then moves focus out of the control. See Section 14, “Forms: The form element” (p. 41).
<code>onclick</code>	When the user clicks the mouse button over the element.
<code>ondblclick</code>	When the user double-clicks the mouse on the element.
<code>onfocus</code>	At any given time, some element is said to have focus; this means that any keyboard input from the user is sent to that element. The <code>onfocus</code> event on an element happens when the focus is moved to that element. This can happen due to use of the <i>tab</i> key, or because the user clicked on it.
<code>onkeydown</code>	When a user presses a key; applies to the element with focus. See <code>onfocus</code> for a discussion of focus.
<code>onkeypress</code>	When a user presses and then releases a key on the keyboard; applies to the element that has focus. See <code>onfocus</code> , above, for a discussion of focus.
<code>onkeyup</code>	When a user releases a key after pressing it. Compare <code>onkeydown</code> and <code>onkeypress</code> .
<code>onload</code>	When a <code>body</code> element has finished loading.

<code>onmousedown</code>	When the mouse button is depressed over the element.
<code>onmousemove</code>	When the mouse is moved within an element. Not every position will be reported; the faster the mouse is moved, the fewer <code>onmousemove</code> events will be generated.
<code>onmouseout</code>	When the mouse is moved out of an element.
<code>onmouseover</code>	When the mouse is moved onto the element.
<code>onmouseup</code>	When the mouse button is released over the element.
<code>onreset</code>	When a user clicks on the reset button of a form; see Section 14, “Forms: The form element” (p. 41).
<code>onselect</code>	When a user selects some text, as in a <code>textarea</code> element or an <code>input</code> text field. See Section 14, “Forms: The form element” (p. 41).
<code>onsubmit</code>	When a user submits a form; see Section 14, “Forms: The form element” (p. 41).
<code>onunload</code>	When a <code>body</code> element has been removed from a browser window.

17. Legacy and unrecommended elements

This document does not discuss a number of elements. They fall into two classes:

- **Deprecated elements.** These elements are relics of the earlier days of HTML, and there are better and more modern techniques to replace them.
- **Unrecommended elements:** parts of XHTML that, in the author's opinion, represent bad practices and techniques.

If you need to use any of these features, refer to the XHTML 1.1 standard¹⁷.

17.1. Deprecated features

These features are officially part of XHTML, but may disappear in future versions.

br

This element meant “line break here.” Instead, use a `div` element to put each line into its own block.

tt, i, b, strike

These presentational elements were used to get specific font presentations: typewriter type, italic, boldface, and strikeout type. Use structural elements instead, such as `code`, `em`, or `strong`, or use `span` elements that are linked to CSS rules as described in Section 5, “Separating content and presentation with CSS” (p. 7).

align and valign attributes

These attributes could be used to change the horizontal or vertical alignment of headings, paragraphs, and other elements. Because it is good practice to separate structure from presentation, alignment should be handled in a stylesheet. See Section 5, “Separating content and presentation with CSS” (p. 7).

17.2. Features that are not recommended

These features are part of XHTML, but the author does not recommend their use.

¹⁷ <http://www.w3.org/TR/xhtml11/>

frame

Frame sets allow you to divide your page into blocks, each of which operates like an independent page.

However, you cannot link to one frame of a page. Also, the function of the *Back* and *Forward* buttons in your browser is not well-defined with respect to changes in the content of frames.

If you want to divide your page up into blocks, use CSS for the layout, and make up multiple pages for the different contents. That way, each state can have its own URI.

map

The `map` element is used to set up a clickable image map. Within this map, you can set up different “hot spots” so that any mouse click in such a spot acts like a link.

The principal problem with image maps is accessibility. Blind users cannot see the image map. Use regular hypertext links or the `button` element in a form.

