XHTML, step by step

Move beyond tag soup to create sites that truly separate content from presentation

Skill Level: Introductory

Uche Ogbuji (uche.ogbuji@fourthought.com)
Principal Consultant
Fourthought Inc.

06 Sep 2005

Get started working with Extensible Hypertext Markup Language. XHTML is a language based on HTML, but expressed in well-formed XML. However, XHTML has more to offer than just regularizing tags and characters -- XHTML can alter the way you approach Web design. This tutorial gives step-by-step instruction for developers familiar with HTML who want to learn how to use XHTML in practical Web sites.

Section 1. Tutorial introduction

Introducing XHTML

This tutorial offers step-by-step instruction for developers familiar with HTML who want to learn how to use XHTML in practical Web sites. XHTML is more than just an XML-compliant variation. It also opens up the first step in advancing from tag soup Web sites to sites that properly separate content from presentation, and thus save time and money in maintenance. The focus of this tutorial will be XHTML 1.1, with the occasional mention of XHTML 2.0. The lessons are built around examples that readers can view and experiment with in their favorite browser.

Who should take this tutorial?

XHTML is important in all areas of Web design and management. It is broadly applicable, and yet relatively easy to pick up for those already familiar with HTML and XML. Web designers should take this tutorial to learn how to develop well-formed pages. Programmers should take this tutorial if they deal with any Web applications. Developers who are learning XML should take this tutorial to learn about one of the most important examples of XML.

Prerequisites

This tutorial assumes knowledge of XML, XML namespaces, and HTML. If you aren't familiar with XML, take the developerWorks tutorial "Introduction to XML." If you aren't familiar with HTML, a good place to start is "Dave Raggett's introduction to HTML." If you need to learn about XML namespaces, read the article "Plan to use XML namespaces, Part 1."

I highly recommend that you try out the examples. They only require a Web browser that supports XHTML 1.1 -- and most current Web browsers do support this standard. When giving browser output examples, I show screenshots of Firefox 1.0.4 on Fedora Core Linux. Firefox is a popular Web browser available on Windows, Mac OS X, Linux, and other platforms. It is based on Mozilla's rendering engine.

About the examples in this tutorial

In this tutorial you will see many examples of XHTML files. You can find all the files shown here in the zip file, x-xhtml-tutorial-files.zip. In this package, all files start with a prefix indicating the section that discusses them and the order of examples within the section. For example, the names of files from the first example in the third section start with "eg_3_1".

Files that end with .xhtml are XHTML. A few of the files have more specialized extensions such as .xsl for XSLT transform files.

I do take care to further list the example files in each panel and show how each relates to the other, so if you follow along with the tutorial you should be able to locate and experiment with the examples easily enough.

Section 2. Anatomy of an XHTML Web page

The flavors of XHTML

When the W3C decided to create a well-formed XML version of HTML, it started with HTML 4.01. XHTML 1.0 is pretty much a direct port of the HTML 4.01 structure to XML syntax. However, the W3C actually came up with *three* HTML 4.01 DTDs because it wanted to steer HTML toward cleaner semantics for content rather than presentation. This means that some elements are deprecated in favor of stylesheet rules, and the DTD variants reflect the author's decision whether or not to use such elements. These DTDs are:

- **Strict:** The preferred DTD, excluding attributes and elements that are deprecated as being too presentational.
- **Transitional:** The Strict DTD with deprecated presentation elements added in.
- **Frameset:** A slight variation on the Transitional DTD for documents that use frames.

XHTML 1.0 includes XML equivalents for all three DTDs. The W3C then developed a system to describe and extend XHTML as a set of DTD **modules**. This specification is called Modularization of XHTML, and is the basis of XHTML specifications after version 1.0. XHTML 1.1, the focus of this tutorial, is basically XHTML 1.0 Strict broken nto a set of modules. In terms of actual usage, you'll see only minor changes from XHTML 1.0 Strict.

The W3C is currently working on XHTML 2.0, which offers more significant changes and is in many ways a full language redesign. You should certainly keep an eye on XHTML 2.0, but it is still in development; this tutorial will focus on the latest released version of XHTML, which is 1.1.

The XML declaration

XML supports a declaration for any document. It's technically optional, but it's good practice to always include one. One reason is that certain defaults determine information that you do not specify in the declaration, but these defaults often conflict with some aspect of your file.

An XML declaration takes the following form:

```
<?xml version
                    opt. encoding
```

```
                    opt. standalone?>
```

The three key bits of the declaration are called **pseudo-attributes**, because they look syntactically similar to XML attributes. If you include an encoding declaration it must follow the version, and if you include a standalone declaration it must be the last pseudo-attribute. It so happens that it never makes sense to use the standalone declaration in an XHTML document, so I shall not discuss it further. You will use "1.0" as the version, so the only part you'll ever really change in XHTML is the encoding.

I suggest you use UTF-8 or UTF-16 for XHTML documents, but this requires that you be very careful with the configuration of the Web server or other system that will host the document. If it is not clear at this low level that UTF-8 or UTF-16 is the encoding, too many browsers will ignore the XML declaration and assume ISO-8859-1, which can cause corrupted text.

A full example of an XML declaration for XHTML is:

```
<?xml version="1.0" encoding="utf-8"?>
```

This declaration appears at the beginning of the file -- although if you are using UTF-16, you will need what is known as a **byte-order mark** even before the declaration. (The byte order mark is a special character, not visible to people, but set aside for indicating the structure of byte sequences in a particular machine.)

Document type declarations

XHTML is defined in a **document type definition (DTD)**, and you must refer to this DTD in a construct just after the XML declaration called the **document type declaration**. In this tutorial, I focus on XHTML 1.1, which means that in all examples the document type declaration will look as follows:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
    "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
```

If you do not have such a declaration, you do not have a valid XHTML 1.1 document. `-//W3C//DTD XHTML 1.1//EN` is the public identifier, which you can omit; however, if you do include it, you must use this precise value. `http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd` is the system identifier. You cannot omit it, but you can change the URL to point to any copy of the W3C's

XHTML 1.1 DTDs. In most cases, an XML parser will look to retrieve the DTD specified at the system identifier, which can cause problems if you're offline. The usual solution for this is an XML catalog, which allows you to redirect requests for entities to customized locations. Most Web browsers and other tools that support XHTML have such catalogs built in and don't bother with the generally slow process of retrieval from that URL. You can also instruct many XML parsers not to retrieve external parameter entities, which suppresses the DTD URL request.

One other reason not to overlook the document type declaration is that Web browsers usually interpret the absence of such a declaration as a directive to render in **quirks** mode, which is designed to accommodate the tag soup of HTML legacy and non-standard browser behavior. By avoiding quirks mode, you'll find the HTML authoring and quality assurance process much less painful.

Main elements and namespace

Elements and attributes in XHTML 1.1 are very similar to those in the more familiar HTML, with the added restrictions that come with well-formed XML. I'll expand more on these restrictions throughout this tutorial, but one very important new thing to remember -- apart from concerns of XML well-formedness -- is that *elements and attributes in XHTML 1.1 are always in lower case*. They are never capitalized. The top-level element is `html` and never `HTML`.

In addition, XHTML elements are in an XML namespace. Usually you declare this namespace once on the top-level element, and let it apply to the other elements within. The following snippet demonstrates this:

```
<html xmlns="http://www.w3.org/1999/xhtml">
...
</html>
```

Of course you would replace the `...` with your own XHTML content. The namespace `http://www.w3.org/1999/xhtml` is the same for all released versions of XHTML, including the three DTDs of XHTML 1.0 and XHTML 1.1. This may change in the emerging XHTML 2.0 spec. In this example -- and in most cases where XHTML serves as a stand-alone document -- the default namespace is employed, but remember that prefixed forms are perfectly valid. The following snippet is the same as the one above, in XHTML terms:

```
<ht:html xmlns:ht="http://www.w3.org/1999/xhtml">
...
</ht:html>
```

A complete example

So far I've shown only snippets of XHTML. The following listing is the first complete example, taken from the XHTML 1.1 specification.

Listing 1. Complete XHTML example

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>Virtual Library</title>
  </head>
  <body>
    <p>Moved to <a href="http://vlib.org/">vlib.org</a>.</p>
  </body>
</html>
```

If you view this file in a browser, you should see a display similar to that in Figure 1.

Figure 1. A complete XHTML example

Notice the title bar of the window where "Virtual Library" comes from the contents of the `title` element. This demonstrates that you want to pay as much attention to the metadata in the document head as to the content in the body.

Section 3. Understand the ground rules

Avoid obsolete elements

If you're accustomed to working with HTML 3.2 or HTML 4.01 Transitional (also called *loose*), you might have to watch out for elements that are not allowed in XHTML 1.1 because they are too presentational. If you're not quite sure the flavor of HTML you have been using, it's best to assume this warning applies to you. The following is an informal list of elements that you can't use any more with XHTML 1.1.

- `applet`
- `basefont`
- `center`
- `dir`
- `font`
- `iframe`
- `isindex`
- `menu`
- `noframes`
- `s`
- `strike`
- `u`

The `font` element is the one you're most likely to miss. I'll touch on how to replace it later in the tutorial. Some attributes on valid elements are also off limits, but it's not easy to package all these details for your review. As an example, there is no longer an `align` attribute on the `div` element. Again, more on this later.

Use an end tag for all elements

Each opening tag must have a matching closing tag. You must no longer use markup such as `<p>hello<b>world</p>`. To make this well formed, you need to

close the bold tag as follows: helloworld .

This means that you have to pay special attention to empty elements such as br and hr. You might be accustomed to just writing <hr>, but now you have to use proper empty element syntax, <hr/> (notice the added slash). To avoid confusion, I do not recommend the other proper syntax for empty elements, <hr></hr>. For a while, it was a recommended workaround to insert a space before the empty element's slash -- <hr /> -- but this is no longer really necessary. As long as you properly include the document type declaration (thus avoiding quirks mode), most Web browsers will render the element correctly.

Think of elements as containers of other markup, rather than boundaries between markup. If you keep this in mind, you will be less tempted to write:

Listing 2. Incorrect XHTML (end tags)

```
<blockquote>
  Twinkle twinkle little star<p>
  How I wonder what you are<p>
  Up above the world so high<p>
  Like a diamond in the sky<p>
</blockquote>
```

Remember that each paragraph element is a container, and write:

Listing 3. Correct XHTML (end tags)

```
<blockquote>
  <p>Twinkle twinkle little star</p>
  <p>How I wonder what you are</p>
  <p>Up above the world so high</p>
  <p>Like a diamond in the sky</p>
</blockquote>
```

Nest elements properly

Make sure that elements are properly nested. Rather than write:

Listing 4. Incorrect XHTML (nested elements)

```
<p>
  <em><strong>Hello world</em></strong>
</p>
```

you must write:

Listing 5. Correct XHTML (nested elements)

```
<p>
  <em><strong>Hello world</strong></em>
</p>
```

The `strong` element is now completely nested within `em` rather than overlapping. Notice that I used elements `em` and `strong` rather than the more commonly known `i` and `b`. The latter two are also supported in XHTML 1.1, but you should consider using the former two instead because they have more content-specific semantics.

Use proper form for all attributes

Do not leave attribute values unquoted in XHTML. With HTML, you might have omitted the quotes if the value was a valid number, but XHTML allows no exceptions. Rather than `<table border=1>`, write `<table border="1">`.

Do not leave off attribute values. In HTML some attributes serve effectively as flags, so that you could write `<input disabled>`, whereas with XHTML you repeat the attribute name as the attribute value in these cases: `<input disabled="disabled">`. Again, don't forget that all attribute names are in lower case, which is reflected in such attribute values.

Watch out for XML's white space normalization rules for attributes. This basically means that you cannot expect to use whitespace very meaningfully in attributes except as simple, atomic delimiters. So an attribute value of `" a b c "` will end up being treated as if you'd written `"a b c"`.

Use id attributes

XHTML 1.1 does not permit the `name` attribute, which is generally used to set anchors for fragment identifiers; instead, use the the `id` attribute. So rather than writing:

```
<a name='foo'>Weigh anchor here</a>
```

Write the following:

```
<a id='foo'>Weigh anchor here</a>
```

Use the xml:lang attribute

XHTML 1.1 does not permit the `lang` attribute, which can be placed on any HTML element to specify the natural language of its contents. Use the `xml:lang` attribute instead of `lang`. Rather than write:

```
<p lang='fr'>Mon ami</p>
<p lang='en'>My friend</p>
```

Write the following:

```
<p xml:lang='fr'>Mon ami</p>
<p xml:lang='en'>My friend</p>
```

Remember that the `xml` prefix is reserved for the namespace `http://www.w3.org/XML/1998/namespace`, which you need not explicitly declare.

I presented an example of `xml:lang` earlier in "A complete example."

Section 4. Replace common HTML idioms

Use CDATA sections

You're probably familiar with the suggested practice of using comments to wrap the body of script elements in HTML:

Listing 6. Wrap script elements in comments

```
<script language="JavaScript">
<!--
var msg = "Hello World!";
document.write(msg);
//-->
</script>
```

This practice includes several layers of escaping and caution. For example the `//` on the second to last line is a JavaScript comment marker that prevents fragile JavaScript engines from being confused by the closing XML comment; this comment in turn prevents fragile browsers from choking on the script contents. You can generally avoid this elaborate game by using XML's general-purpose escaping construct: the CDATA section. In XHTML, the example looks like this:

Listing 7. Wrap script in a CDATA section

```
<script language="JavaScript">
<![CDATA[
var msg = "Hello World!";
document.write(msg);
]]>
</script>
```

You can leave left angle brackets, ampersands, and similar characters unescaped in CDATA sections. Any browser that can handle properly declared XHTML 1.1 can handle the semantics of elements such as `script`. CDATA sections have one gotcha, however -- the sequence `]]>` cannot appear within the body of a CDATA section. Fortunately, this is a rare sequence -- but it can appear in JavaScript code, especially when dealing with array types.

Listing 8. Example of the sequence "]]>" appearing in JavaScript code

```
if (i[j[0]]>1)
{
  //do something
}
```

In the example in Listing 8, the workaround is to add a space between the right square bracket characters ( ]) or before the greater than sign (>).

You should also get in the habit of using CDATA sections to wrap the body of `style` elements.

Understand inline style to set font and text details

With version 1.1, XHTML no longer includes the `font` element. If you used this element to set font and other text styles, you should now control such matters with CSS. The simplest way to do so is with inline style. Rather than writing:

```
<p><font color='blue' size='6'>Hello world</font></p>
```

You could write the following (although I don't recommend it because it's deprecated):

```
<p style='color: blue; font-size: x-large;'>Hello world</p>
```

Notice that the `size` attribute becomes the `font-size` CSS property, and that the value 6 becomes the clearer `x-large`. The old font size numbers translate to the following CSS `font-size` values:

- `1` becomes `xx-small`
- `2` becomes `x-small`
- `3` becomes `small`
- `4` becomes `medium`
- `5` becomes `large`
- `6` becomes `x-large`
- `7` becomes `xx-large`

HTML's `font` is an inline element, but this does not limit how you can use the above technique. You can place a `style` attribute on any element, including inline elements. If you don't have any other inline element on which to make style distinctions, you can use `span`, as follows:

```
<p>Hello <span style='color: blue; font-size: x-large;'>world</span></p>
```

In this example, the specified style only applies to the text "world."

Again, the `style` attribute is deprecated because it mixes presentation with content. It is legal to use in XHTML 1.1, but is not encouraged and you will have to use proper stylesheets in future versions of XHTML, as I discuss next.

Use stylesheets to set font and text details

CSS stylesheets are a more modular approach to applying style in XHTML (and HTML) than `style` attributes, which is why the latter have been deprecated. One

way to incorporate a stylesheet into an HTML document is to include it in a `style` element in the document's head. The document in Listing 9 shows how to do this in a way that corresponds to the snippet in Understand inline style to set font and text details.

Listing 9. Include a stylesheet in a style element

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>Hello document</title>
    <style type="text/css">
<![CDATA[
p {
    color: blue;
    font-size: x-large;
}
]]>
  </style>
  </head>
  <body>
    <p>Hello world</p>
  </body>
</html>
```

Notice the use of the CDATA section for the body of the `style` element, as discussed in "Use CDATA sections."

If you view this file in a browser, you should see a display similar to that in Figure 2.

Figure 2. Use stylesheets to set font and text details

Use the XML stylesheet processing instruction

In HTML documents, when you want to have the stylesheet external to the document you use a `link` element to specify the location of the external stylesheet resource. The following snippet has an example of such a link.

Listing 10. Specify an external stylesheet in HTML with a link element

```
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
  <head>
    <title>Hello document</title>
    <link rel="stylesheet" type="text/css" href="hello.css"/>
  </head>
```

In XHTML 1.1, you instead use the XML processing instruction (PI) for associating stylesheets with documents. I covered this PI in earlier XML/CSS tutorials on developerWorks (see Resources). The snippet in Listing 11 is equivalent to that in Listing 10, but using a PI instead.

Listing 11. Associate stylesheets with documents using a PI

```
<?xml-stylesheet type="text/css" href="hello.css"?>
<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en" >
  <head>
    <title>Hello document</title>
  </head>
```

Use style to align content

HTML's attributes for alignment of block-level elements are another good example of where you have to get used to stylesheets as the proper way to control presentation. For example, the `div` element no longer has an `align` attribute in XHTML 1.1. So the following is not legal in XHTML 1.1:

```
<div align="center">Hello world</div>
```

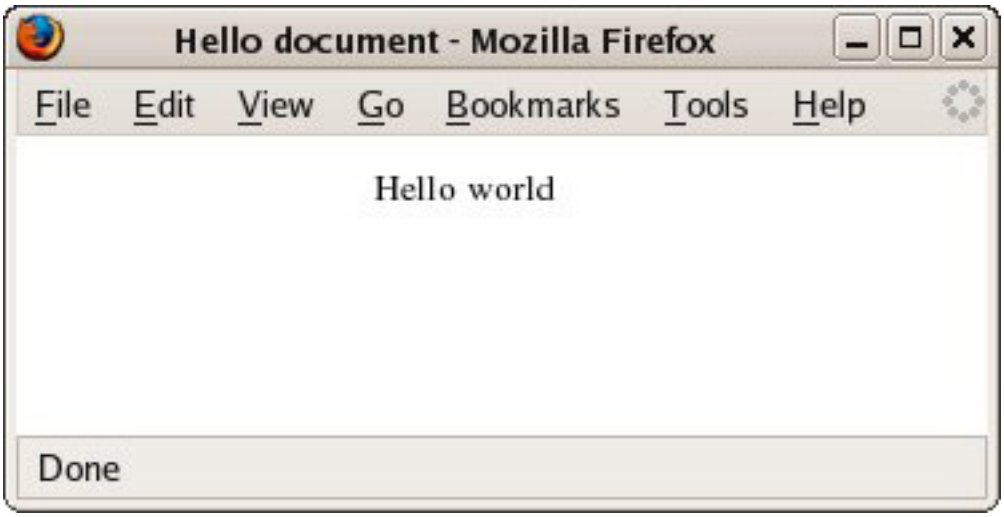
You can use an internal or external stylesheet for this. The example in Listing 12 (eg_4_5.xhtml in the download file) uses an internal stylesheet.

Listing 12. Align content with an internal stylesheet

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>Hello document</title>
    <style type="text/css">
<![CDATA[
div {
    text-align: center;
}
]]>
  </style>
  </head>
  <body>
    <div>Hello world</div>
  </body>
</html>
```

Notice that the `align` attribute becomes the `text-align` CSS property. If you view this file in a browser, you should see a display similar to that in Figure 3.

Figure 3. Align content with an internal stylesheet



Use style for table presentation

HTML's tables allow a host of attributes for controlling presentation, including the `align` attribute discussed in the Use style to align content. These have all been moved to CSS stylesheet properties in different ways. The following example of a typical HTML table start tag is not legal in XHTML 1.1:

```
<table border="0" cellpadding="5" cellspacing="5" width="50%">
```

All these attributes are obsolete in XHTML 1.1, and you must use stylesheet rules instead. Listing 13 (eg_4_6.xhtml in the download file) uses an internal stylesheet.

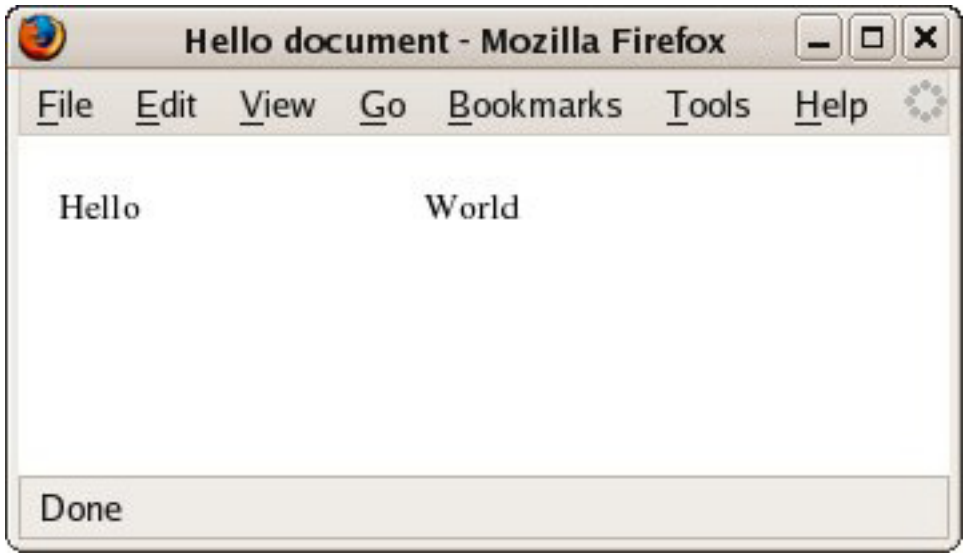
Listing 13. Use an internal stylesheet for table presentation

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>Hello document</title>
    <style type="text/css">
<![CDATA[
table {
    border: None;
    width: 50%;
}
table td {
    padding: 5px;
    spacing: 5px;
}
]]>
  </style>
```

```
  </head>
  <body>
    <table>
      <tbody>
        <tr>
          <td>Hello</td>
          <td>World</td>
        </tr>
      </tbody>
    </table>
  </body>
</html>
```

These attributes become more flexible when moved to a CSS stylesheet. With the `border` CSS property, you aren't limited to a simple thickness property. You can set the color and form of the border line as well. In all cases you gain flexibility over units. You can see how I specified pixel units for the `padding` and `spacing` properties. If you view this file in a browser, you should see a display similar to that in Figure 4.

Figure 4. Use an internal stylesheet for table presentation



Use ins and del to express edited content

A common way to visually express that content has been edited in HTML is to mark removed content with strikethrough elements, `s` or `strike`, and then to use a variety of presentation cues to show content that has been newly inserted. The resulting display is similar to that of a modern word processor with *show changes* or a similar option enabled. The strikethrough element is gone from XHTML 1.1, but instead you have available a pair of elements that express directly that content has been inserted or deleted. The following example is no longer legal:

```
<p>Hello <s>moon</s><b>world</b></p>
```

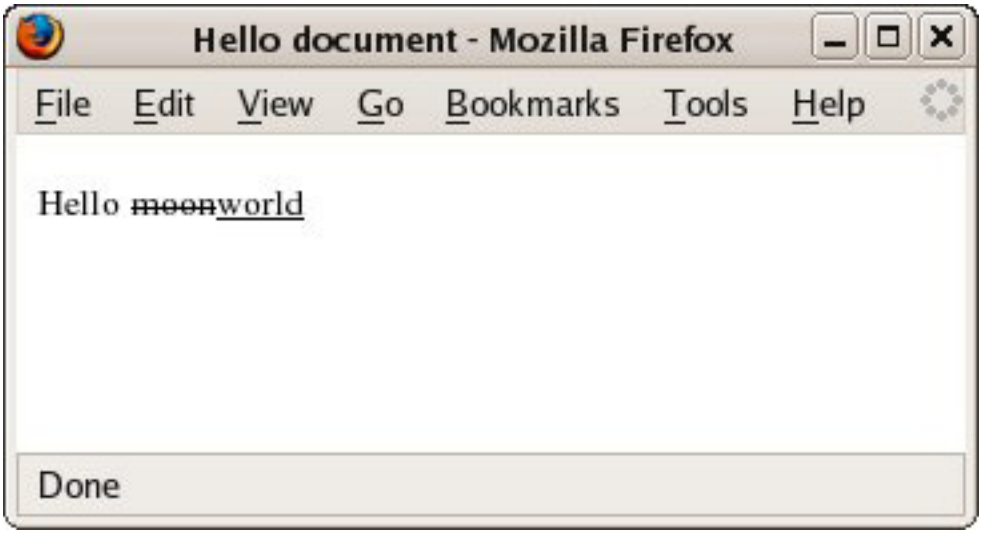
The `s` element is outright invalid; the `b` element is valid, but not encouraged (the `strong` tag is the usual replacement). Listing 14 (eg_4_7_1.xhtml in the download file) uses the preferred elements `ins` and `del`:

Listing 14. Express edited content with ins and del

```
<?xml version="1.0" encoding="UTF-8"?>
<!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>Hello document</title>
  </head>
  <body>
    <p>Hello <del>moon</del><ins>world</ins></p>
  </body>
</html>
```

If you view this file in a browser, you should see a display similar to that in Figure 5.

Figure 5. Express edited content with ins and del



Notice that Firefox's default rendering for the inserted text is underscore, rather than bold, which I used in the first snippet. You can override this in the CSS with a rule such as:

Listing 15. Override the underscore in Firefox

```
```

```
ins {
    text-decoration: none;
    font-weight: bold;
}
```

The first rule removes the underline effect and the second specifies boldface. Try out the example file eg_4_7_2.xhtml in the download file, to see how this affects the display.

Other updated idioms, in brief

Fonts and alignment are just a couple of the presentation details that you have to move from specialized attributes into stylesheets. The following are some other brief examples of common HTML idioms and their XHTML+CSS equivalents.

Rather than `<body bgcolor="white">`, lose the `bgcolor` attribute and instead use the CSS property `background-color: white;`. Likewise, replace `text` with the CSS property `color`, and replace `marginwidth` and `marginheight` with `margin`.

Do not write u for underlining; instead use a `span` associated with the CSS property `text-decoration: underline`. (Amusing note: the property `text-decoration: blink` has the same effect as the infamous, nonstandard element `blink`, but considering the popular response to this element -- users complain that blinking is an annoying distraction on a Web page -- I don't recommend that you actually use this CSS property.)

Rather than `<td valign="middle">`, lose the `valign` attribute and instead use a CSS property `vertical-align: middle`.

Rather than suppress the border that some browsers place around images with `<img border="0" ...>`, go with the CSS property `border` which I already discussed in the context of tables.

In the real world, you need more than one style for each element type, and you will have to learn more about CSS selectors in order to write effective stylesheets. You should learn about class, ID, descendant, sibling, and other sorts of CSS selectors. I cover some of these in my tutorials on using CSS with XML (but not necessarily XHTML) documents (see Resources).

Section 5. Some practical considerations

Do not use text/html with XHTML 1.1

An **Internet Media Type (IMT)**, also known as a MIME type, is designed to communicate to a program the correct interpretation of some resource. The well-known IMT for HTML documents is `text/html`. There are several important reasons not to use this IMT with XHTML, but they all boil down to the fact that an XHTML 1.1 document is a different beast from an HTML document (as you have no doubt noticed in this tutorial), and you'll suffer nothing but quirks and confusion if you don't make that clear.

The discussion on the best IMT for XHTML has been a long and winding one, but it has settled firmly on `application/xhtml+xml`. You should use this IMT, but you might have to work through some nuances (see Resources) to accommodate Web browsers that don't support it perfectly.

You might not need to directly set media type unless you run a Web server, but this practical suggestion might affect you in subtle ways. For one thing, consider using the .xml or .xhtml file extension for XHTML files, rather than the .html extension. Many programs try to guess the IMT of a document from its file extension, and will report the IMT of documents with the .html file extension as `text/html`. You should also keep the correct IMT in mind if you use the `link` element to express inter-document relationships -- particularly if you use the `type` attribute.

Use the W3C XHTML validator

You'll find a lot of rules and a lot of nuances in getting XHTML right. If XHTML is your first foray into Web design, these are no more arcane and complex than the rules for effective use of HTML. But if you do have experience with HTML (as most of you do), you might frequently have to catch yourself from sliding into bad habits. A good way to do this is to use a validator to check your XHTML pages.

The best known validator is the W3C's online tool (see Resources). The W3C actually has online validators for many of their specifications, including almost all flavors of HTML and XHTML. These have all been combined into one simple form in which you can specify a URL or upload a file. The validator checks the file for a document type declaration that indicates which specification to validate against. If the page does *not* validate, then you get to try again, optionally specifying the HTML

or XML version, encoding, or other parameters. This way you can more easily home in on basic problems that trigger a cascade of validity problems.

And yes, I did run all the XHTML examples in this tutorial through the validator.

Use the XML declaration for character encoding

Most Web browsers today look at the HTTP headers as their primary way to determine the character encoding for the Web page. You should certainly continue to configure Web servers and the like to provide this information. You should also make sure that this information is present in the XML declaration of the document itself -- and of course, you should be sure that the encoding specified in the XML declaration is consistent with that specified in the HTTP headers.

In HTML documents, you might have overridden the character information in yet another way: in document head metadata (using the `meta` element with `http-equiv` attribute). The following snippet includes an example of such a declaration.

```
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
```

You no longer need to use this in XHTML, and indeed the W3C says that you should not do so.

Use XSLT to provide standby conversion to HTML

I encourage you to move away from legacy HTML and towards XHTML 1.1. Most browsers these days can handle XHTML with relatively few work-arounds, and support for it gets better with each browser release. Nevertheless, at some point you may want to render your XHTML content as classic HTML. If that happens, you'll be glad you decided to start with XHTML. In general, it's very hard to convert from HTML to XHTML but very easy to go the other way, in effect because XHTML is more regular and structured than HTML. Probably the quickest way to convert from XHTML to HTML is through XSLT.

Because XHTML 1.1 is based on XHTML 1.0 Strict, which is based on HTML 4.01 Strict, most elements and attributes are the same across these formats. This means that just by rendering the same elements in an XHTML document, but using HTML conventions, you've already achieved most of the conversion. XSLT does provide an HTML output method that produces an HTML result (version 4.0 by default, but you can specify a different version), and a technique called an **identity transform**

echoes the input elements, attributes, and text to the output. You can use such a transform as a basis for conversion from XHTML to HTML. The following listing (xhtml2html.xsl in the download file) is based on the identity transform, but also strips elements of the XHTML namespace and converts `xml:lang` attributes to `lang`, and `id` attributes to `id` and `name` pairs. These are the only content model changes that are required in most cases.

Listing 16. Identity transform and conversions (xhtml2html.xsl)

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

  <xsl:output method="html" version="4.01" encoding="iso-8859-1"
              doctype-public="-//W3C//DTD HTML 4.01//EN"
              doctype-system="http://www.w3.org/TR/html4/strict.dtd"/>

  <xsl:template match="*" priority="1">
    <xsl:element name="{name()}">
      <xsl:apply-templates select="@*|node()"/>
    </xsl:element>
  </xsl:template>

  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="@xml:lang">
    <xsl:attribute name="lang">
      <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:template>

  <xsl:template match="@id">
    <xsl:attribute name="id">
      <xsl:value-of select="."/>
    </xsl:attribute>
    <xsl:attribute name="name">
      <xsl:value-of select="."/>
    </xsl:attribute>
  </xsl:template>
</xsl:stylesheet>
```

You need not fully understand this XSLT in order to use it. Just pass it to your favorite XSLT processor with the XHTML document that you want to convert as the source document. (Bear in mind that most modern Web browsers support XSLT.) The result should be quite serviceable HTML.

Embed XHTML in other vocabularies

You do not have a full, valid XHTML document unless it starts with the document

type declaration and a top-level `html` element, but you can still reuse snippets of XHTML in other XML vocabularies that allow extensibility by embedding elements in a foreign namespace. In this case, you can just pick any XHTML block element such as `div`, `table`, or `ol` and insert it into the other XML vocabulary, with the XHTML namespace declared. (When embedding, it's useful to adopt a prefix so it's clear that the elements are not part of the host vocabulary.) Listing 17 is a perfectly valid XSLT document, with a block at the top that contains some documentation information using embedded XHTML.

Listing 17. XSLT document with embedded XHTML

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
>

  <html:div xmlns:html="http://www.w3.org/1999/xhtml" xml:lang="en">
   This is the XSLT identity transform, described in
   <html:a href="http://www.w3.org/TR/xslt#copying">section 7.5</html:a>
   of the XSLT 1.0 specification.
  </html:div>

  <xsl:template match="@*|node()">
    <xsl:copy>
      <xsl:apply-templates select="@*|node()"/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Section 6. Wrap up

Summary

In this tutorial, you have learned how to write XHTML, including how to express many of the things in HTML that you might be familiar with. In particular you have learned about:

- The various flavors of HTML and XHTML
- The XML declaration
- The XHTML 1.1 document type declaration

- The XHTML namespace
- The basic element structure of XHTML
- Proper use of start and end tags in XML and thus XHTML
- Proper element nesting in XML and thus XHTML
- Proper form for attributes in XML and thus XHTML
- The idea of separating content from presentation in XHTML
- Elements from HTML that are disallowed in XHTML
- Attributes on XHTML elements that were allowed in HTML, but are disallowed in XHTML
- Expressing XHTML element IDs to provide anchors for URL fragments
- Expressing content language in XHTML elements
- Escaping text within XHTML script and style elements
- Using CSS stylesheets to express presentation details
- Specifying external CSS stylesheets for use in XHTML
- The proper MIME type to use with XHTML documents
- Proper ways to express the character encoding of XHTML documents
- Using the W3C XHTML validator
- Embedding XHTML in other XML vocabularies

This is quite a collection of topics, but it is merely a springboard for you to become an XHTML guru through further practice.

Downloads

Description	Name	Size	Download method
Code samples for XHTML tutorial	x-xhtml-tutorial-files.zip	4 KB	HTTP

Information about download methods

Resources

Learn

- Not familiar with XML? Start with the many helpful resources in the developerWorks "New to XML" page, especially the tutorial "Introduction to XML" by Doug Tidwell (August 2002).
- Learn about XML namespaces in the article "Plan to use XML namespaces, Part 1" by David Marston (developerWorks, April 2004) and learn more about how to use namespaces effectively in the articles "Plan to use XML namespaces, Part 2" by David Marston (developerWorks, April 2004) and "Principles of XML design: Use XML namespaces with care" by Uche Ogbuji (developerWorks, April 2004).
- Learn or revise HTML starting with Dave Raggett's introduction to HTML. For more thorough instruction, get a good book. A recommended title is "HTML & XHTML: The Definitive Guide, Fifth Edition," by Chuck Musciano and Bill Kennedy (O'Reilly and Associates, 2002).
- Learn more about the XML declaration in the article "Always use an XML declaration" by Uche Ogbuji (developerWorks, April 2004).
- Bookmark the definitive reference on XHTML 1.1, the W3C spec. Keep in touch with developments and emerging resources on HTML and XHTML on the W3C HTML home page.
- Find useful discussion of the changes in XHTML 1.1 in the Weblog entry "What You Should Know about XHTML 1.1," by Wayne Burkett.
- Get the basics of CSS and CSS with XML. See the tutorial "Use Cascading Stylesheets to display XML, Part 1" (developerWorks, November 2004) -- including the links in Resources -- to get started. This tutorial introduces the use of CSS to style XML in browsers.
- Continue with "Use Cascading Stylesheets to display XML, Part 2" (February 2005), which covers advanced topics for the use of CSS to style XML in browsers. If you are familiar with XSLT, consider going through Part 3 (June 2005) which discusses XSLT techniques for working with CSS for HTML or XML output.
- Learn the right way to use Internet Media Types with XHTML. The semi-official specification is the W3C's "XHTML Media Types" (Note, August 2002). A more informal discussion is "The Road to XHTML 2.0: MIME Types", by Mark Pilgrim.
- Learn about XML Catalogs from Norman Walsh's article "XML Entity and URI Resolvers."

Get products and technologies

- Check your documents with the free, on-line W3C Markup Validation Service.

About the author

Uche Ogbuji
Uche Ogbuji is a consultant and co-founder of Fourthought Inc., a software vendor and consultancy specializing in XML solutions for enterprise knowledge management. Fourthought develops 4Suite, an open source platform for XML, RDF, and knowledge-management applications. Mr. Ogbuji is also a lead developer of the Versa RDF query language. He is a computer engineer and writer born in Nigeria, living and working in Boulder, Colorado, USA. You can find more about Mr. Ogbuji at his Weblog Copia, or contact him at uche.ogbuji@fourthought.com.