

Display XML with Cascading Stylesheets, Part 3: Use Cascading Stylesheets to display XML

Combine XSLT and CSS to process XML

Skill Level: Intermediate

[Uche Ogbuji \(uche.ogbuji@fourthought.com\)](mailto:uche.ogbuji@fourthought.com)
Principal consultant
Fourthought Inc.

20 Jun 2005

In parts 1 and 2 of this tutorial series, Uche Ogbuji has shown how to use Cascading Stylesheets (CSS) to display XML in browsers, presenting basic and advanced techniques. Although some people see XSLT and CSS as opposing technologies, they are actually very complementary. There are many XML rendering tasks that CSS cannot handle, and is not designed to handle. XSLT can be used for many such tasks, and can even be used to manage the CSS that is still used to fine-tune the presentation. This tutorial covers techniques for using XSLT to process XML in association with CSS.

Section 1. Tutorial introduction

CSS versus XSLT

In the previous tutorials in this series ([Part 1](#) and [Part 2](#)), Uche Ogbuji showed how to use Cascading Stylesheets (CSS) to display XML in browsers, presenting basic techniques and advanced topics. Some discussions present XSLT and CSS as opposing technologies for presenting XML, but I find little reason to view them this way. Each one offers far more to complement than to counter the other.

This cooperation has many manifestations, but the most common is using XSLT to

generate CSS and include it in the output. XSLT provides more algorithmic power than CSS, so these two technologies form a very rich combination. This tutorial builds on its predecessors to explore this combination.

Who should take this tutorial?

Much XML ultimately finds its way to Web browsers in one way or another. CSS is a great way to make XML presentable in browsers. CSS is best known as the recommended method for specifying the presentation of HTML, but recent versions also support XML; in fact, several browser vendors have embraced it as the best-supported means of displaying XML. CSS is also used in other XML vocabularies such as SVG and XForms.

XSLT is the most widely used means of manipulating XML. It can also be used for display presentation, but CSS is more often used for final browser display. XSLT is a very different approach -- more complex, but more powerful -- and it's supported by many of the browsers that also support CSS.

Anyone who works with XML should take this tutorial. Even if CSS and XSLT don't cover your needs for production Web publishing, they are great tools for general processing, debugging, and experimentation. They offer rich interaction with other XML technologies and you'll likely run into CSS and XSLT even when you least expect them.

Prerequisites

This tutorial assumes knowledge of XML, CSS, XPath, and XSLT. If you aren't familiar with these technologies, I recommend that you first take whichever of the following tutorials suits your needs:

- ["Introduction to XML"](#)
- ["Use Cascading Stylesheets to display XML, Part 1"](#) (introduces the use of CSS to style XML in browsers)
- ["Use Cascading Stylesheets to display XML, Part 2"](#) (covers advanced topics for the use of CSS to style XML in browsers)
- ["Get started with XPath"](#)
- ["Create multi-purpose Web content with XSLT"](#)

I highly recommend that you try out the examples in this tutorial. Some are best checked out with a stand-alone XSLT processor. I use [4Suite](#) 1.0b1. Some require a Web browser that supports XML, XSLT 1.0, and CSS Level 2 or better. When

showing 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, which is a very CSS-compliant browser.

About the XML/CSS examples in this tutorial

In this tutorial you will see many examples of CSS files. All the files used in this tutorial are in the file `x-xmlcss3-tutorial-files.zip` (see [Downloads](#)). In this package, all files start with a prefix indicating what section they are covered in and what order of examples within the section. For example, files from the first example in the third section are named starting with "eg_3_1".

Files ending with `.css` are Cascading Stylesheets, files ending with `.xsl` are XSLT, and those ending in `.xml` are sample XML documents. The latter are to be processed, usually using XSLT of the same prefix. The various XML, CSS, and XSLT files do not always match. A few of the files use more specialized extensions, such as `.svg` for Scalable Vector Graphics.

I do take care to further list the example files in each panel and 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. Basics of using XSLT to generate HTML and CSS

Use XSLT to generate HTML with CSS style attributes

Generating HTML from XML is probably the most common use of XSLT. In the HTML elements you create, you can include CSS within `style` attributes, as supported by the HTML 4.01 spec. The following XML example (`eg_2_1.xml` in `x-xmlcss3-tutorial-files.zip`, see [Downloads](#)) is a list of club members with an attribute for the date when each member joined the club.

```
<?xml version='1.0' encoding='utf-8'?>
<club-members>
  <member joined='2004-04-19'>Mark Askew</member>
  <member joined='2004-11-25'>Chikezie Emereuwa</member>
  <member joined='2005-02-05'>Hideo Oguchi</member>
  <member joined='2005-02-20'>Peter Schneier</member>
</club-members>
```

The following XSLT (eg_2_1.xsl in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) renders the XML for browser display.

```
<?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"/>
  <xsl:strip-space elements="*" />

  <xsl:template match="club-members">
    <html>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="member">
    <!-- Literal output element with literal CSS style attribute -->
    <div style="color: blue;">
      <!-- handle the text children (using XSLT default templates) -->
      <xsl:apply-templates/>
    </div>
  </xsl:template>
</xsl:stylesheet>
```

This XSLT uses a pretty straightforward push style to get the job done. Push style involves heavy use of XSLT templates. The `xsl:output` instruction is important because it specifies that the output should be in HTML (*not* XHTML). You can use other attributes for this instruction besides `method`; make sure you are familiar with these as a way to fine-tune the output. The `xsl:strip-space` instruction is helpful in that it tells the processor to eliminate whitespace in the source document that has no more significance than for formatting. This can help to clean up the HTML output.

Stand-alone XSLT processing

One way to do the transform from [Use XSLT to generate HTML with CSS style attributes](#) is to use a stand-alone processor. For example, using 4Suite's command-line XSLT processor on UNIX, you would run the following command:

```
$ 4xslt eg_2_1.xml eg_2_1.xsl
<html>
  <body>
    <div style="color: blue;">Mark Askew</div>
    <div style="color: blue;">Chikezie Emereuwa</div>
    <div style="color: blue;">Hideo Oguchi</div>
    <div style="color: blue;">Peter Schneier</div>
  </body>
</html>
```

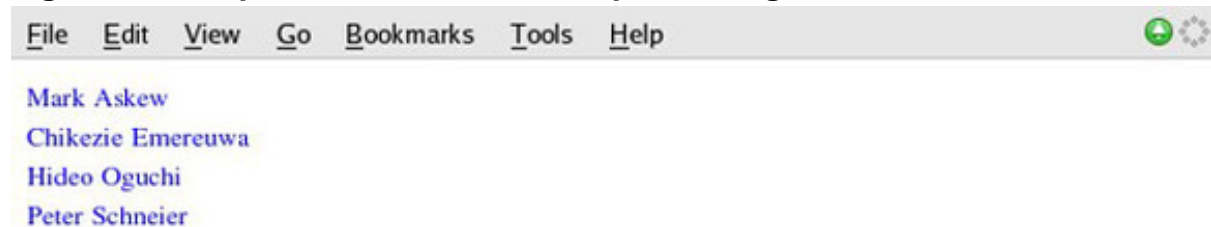
You can see the output printed on the console. As expected, it's plain HTML. You

can send the output to a file in order to view it with a browser:

```
$ 4xslt -o eg_2_1.html eg_2_1.xml eg_2_1.xsl
```

The image below shows how eg_2_1.html appears in the browser.

Figure 1. Example of stand-alone XSLT processing



Done

XSLT through the browser

You can also take a more direct approach to applying the XSLT in the examples so far. The following XML example (eg_2_3.xml in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) is the same as in [Use XSLT to generate HTML with CSS style attributes](#), but with a processing instruction (PI) for applying the XSLT transform (the second line of the listing).

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/xml" href="eg_2_1.xsl"?>
<club-members>
  <member joined='2004-04-19'>Mark Askew</member>
  <member joined='2004-11-25'>Chikezie Emereuwa</member>
```

```
<member joined='2005-02-05'>Hideo Oguchi</member>
<member joined='2005-02-20'>Peter Schneier</member>
</club-members>
```

The result of viewing this XML file in the browser is the same as the image in [Stand-alone XSLT processing](#). The PI is similar to the ones you're already familiar with, for specifying a CSS stylesheet. The main difference is the `type` attribute, which is `text/xml` rather than `text/css`. Unfortunately, this `type` attribute has generated quite a bit of confusion because some tools, including the Internet Explorer Web browser, look for an erroneous value of `text/xsl`. Don't use this incorrect MIME type, if you can help it.

You can also use these PIs transparently in some stand-alone XSLT processors. With the PI, you need not specify the XSLT file if you use 4XSLT:

```
$ 4xslt -o eg_2_3.html eg_2_3.xml
```

From now on, you should be able to decide whether to use a stand-alone processor or your Web browser.

Provide XSLT and CSS as alternative stylesheets

Not all browsers support XSLT, so if you use it as your primary means of presenting XML, consider also providing fall-back CSS (and depending on the reach of your Web site, perhaps pre-generated fall-back HTML). You can place multiple stylesheet PIs in your source XML, as I discussed in the last tutorial, but these do not just have to be differentiated by media. For example, you can place one PI for an XSLT stylesheet and one for a CSS stylesheet. The following XML example (`eg_2_4.xml` in `x-xmlcss3-tutorial-files.zip`, see [Downloads](#)) is the same as in [XSLT through the browser](#), but with an additional PI to apply a CSS transform (third line of the listing).

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/xml" href="eg_2_1.xsl"?>
<?xml-stylesheet type="text/css" href="eg_2_4.css"?>
<club-members>
  <member joined='2004-04-19'>Mark Askew</member>
  <member joined='2004-11-25'>Chikezie Emereuwa</member>
  <member joined='2005-02-05'>Hideo Oguchi</member>
  <member joined='2005-02-20'>Peter Schneier</member>
</club-members>
```

The following CSS is `eg_2_4.css` as referenced in `x-xmlcss3-tutorial-files.zip` (see [Downloads](#)).

```
* { display: inherit; }
club-members { display: block; }
member {
```

```
color: blue;
}
```

The result of viewing this XML file in the browser is the same as the image in [Stand-alone XSLT processing](#).

Create CSS style attributes with the xsl:attribute instruction

In section 2, panel 1 of the [last tutorial](#), I used sample XML just like you saw in [XSLT through the browser](#) to demonstrate how to use CSS partial attribute selectors to highlight in boldface the members who joined in 2005. Performing the same task in XSLT is a much more natural task of string manipulation. I'll present the same example, which works from the same XML list of club members used in the previous panels.

The following XSLT (eg_2_5.xsl in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) highlights in boldface those members who joined in 2005, maintaining the blue text color in all cases.

```
<?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"/>
  <xsl:strip-space elements="*" />

  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="club-members">
    <html>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="member">
    <div style="color: blue;">
      <xsl:if test="substring-before(@joined, '-') = 2005">
        <xsl:attribute name="style">
          <xsl:text>color: blue; font-weight: bold;</xsl:text>
        </xsl:attribute>
      </xsl:if>
      <xsl:apply-templates/>
    </div>
  </xsl:template>

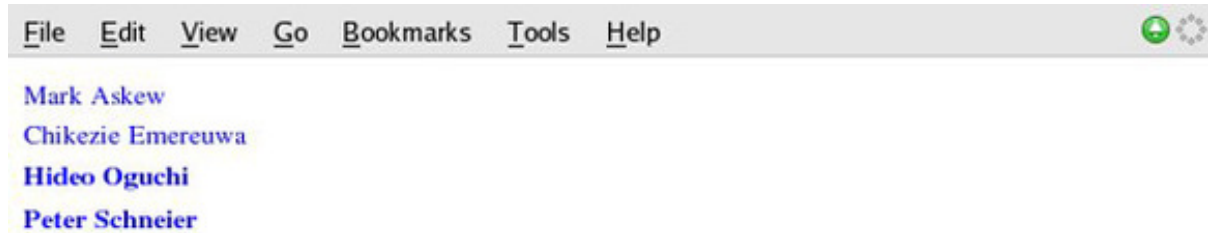
</xsl:stylesheet>
```

The `xsl:if` element uses the XPath `substring-before(@joined, '-') = 2005` to determine dates in 2005, using the fact that the date is in ISO-8601 form. I could also have used an XPath such as `contains(@joined, 2005)`. The style attribute added by the `xsl:attribute` instruction has the same name as the one

in the literal element `div`, so it overrides the earlier attribute. I must also repeat the `color: blue;` property. The `xsl:text` instruction is used to avoid extraneous whitespace in the output attributes.

The XML example `eg_2_5.xml` in `x-xmlcss3-tutorial-files.zip` (see [Downloads](#)) is the same as `eg_2_3.xml`, but with an updated stylesheet PI. The image below shows the output in the browser.

Figure 2. Example of how to create CSS style attributes with the `xsl:attribute` instruction



XSLT named attribute sets

When you have a lot of CSS that you want to apply in style elements, it can become cumbersome. If you need to make a change to the style, you could end up with a lot of instructions to tweak. Of course, using a full CSS stylesheet is one solution to this (which I'll cover later), but even when generating `style` attributes manually, you can modularize things a bit by using **XSLT named attribute sets**. The following XSLT (`eg_2_6.xsl` in `x-xmlcss3-tutorial-files.zip`, see [Downloads](#)) is functionally equivalent

to that in [Create CSS style attributes with the xsl:attribute instruction](#), and results in the same output.

```
<?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"/>
  <xsl:strip-space elements="*" />

  <xsl:attribute-set name="basic-member">
    <xsl:attribute name="style">
      <xsl:text>font-weight: bold;</xsl:text>
    </xsl:attribute>
  </xsl:attribute-set>

  <xsl:attribute-set name="member-2005">
    <xsl:attribute name="style">
      <xsl:text>color: blue; font-weight: bold;</xsl:text>
    </xsl:attribute>
  </xsl:attribute-set>

  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="club-members">
    <html>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="member">
    <xsl:choose>
      <xsl:when test="substring-before(@joined, '-') = 2005">
        <div xsl:use-attribute-sets="member-2005">
          <xsl:apply-templates/>
        </div>
      </xsl:when>
      <xsl:otherwise>
        <div xsl:use-attribute-sets="basic-member">
          <xsl:apply-templates/>
        </div>
      </xsl:otherwise>
    </xsl:choose>
  </xsl:template>
</xsl:stylesheet>
```

The `xsl:attribute-set` instructions at the top define the attributes that you can reuse by invoking `xsl:attribute-set` with the name of the set you want. In this example, each attribute set specifies a single attribute but you can have as many as you'd like. I changed the logic of the `member` template to use `xsl:choose`, which makes the logic a bit clearer. The source XML is `eg_2_6.xml` in `x-xmlcss3-tutorial-files.zip` (see [Downloads](#)), and again it's the same as the XML I've been using, but with an updated stylesheet PI.

Dynamic generation of properties using attribute value

templates

Sometimes you want to use information computed from the XML source in properties in generated styles. This usually occurs in more complex situations, but here's a simple (if somewhat contrived) example: Say you want to list members in a fancy way in the example XML you've been working with -- using increasing shades of grey depending on how late in the year each member joined. The following XSLT (eg_2_7.xsl in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) does the trick:

```
<?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"/>
  <xsl:strip-space elements="*" />

  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>

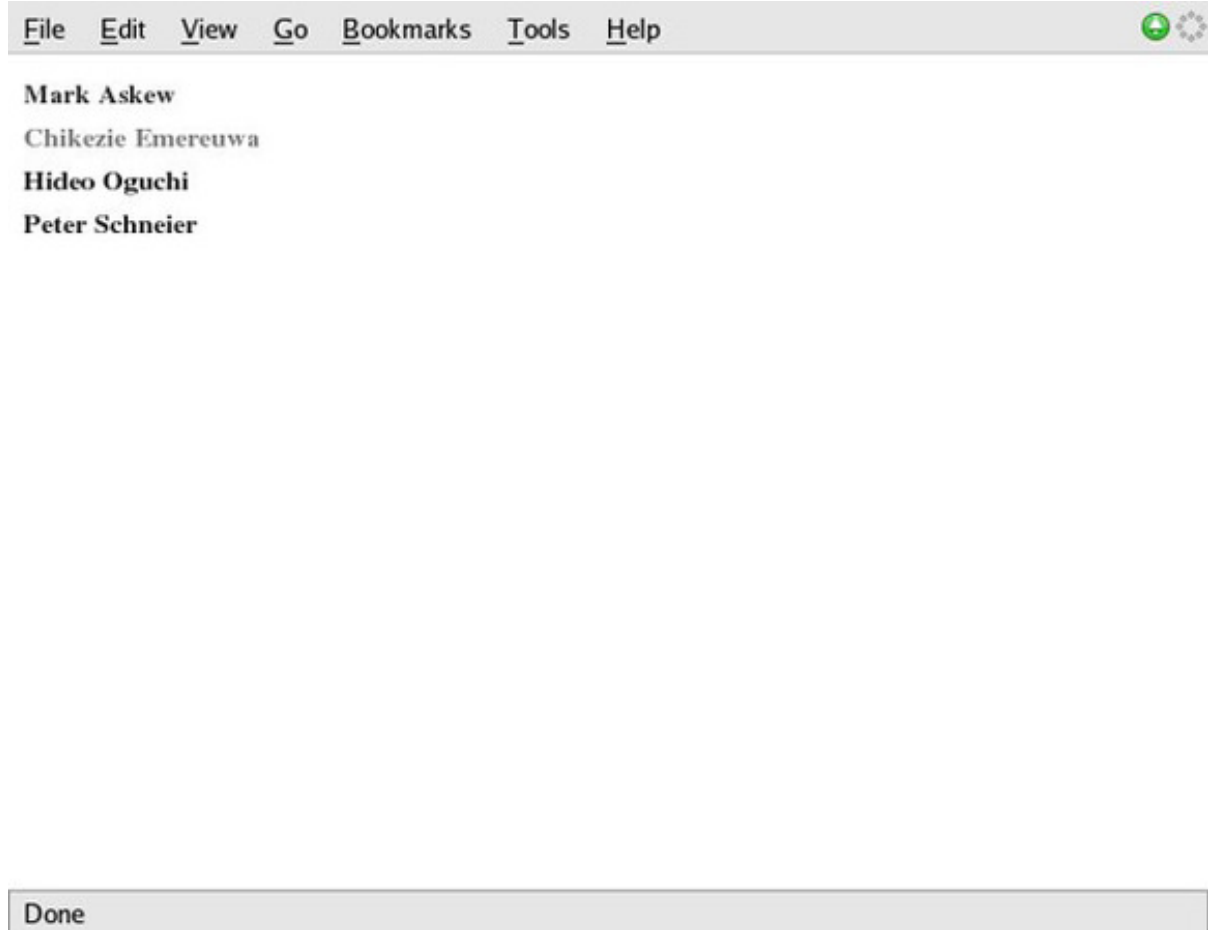
  <xsl:template match="club-members">
    <html>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

  <xsl:template match="member">
    <!-- Extract the month from the date string
         (the part between the dashes -->
    <xsl:variable name="month"
      select="substring-before(substring-after(@joined, '-'), '-')"/>
    <!-- Scale up the month so that you can see the difference in shade
         better. Note: this is fragile logic purely for this contrived
         example. For one thing, it will break if the month is "01" -->
    <xsl:variable name="shade" select="7*$month"/>
    <!-- Use the shade within the AVT in the style property -->
    <div style="font-weight: bold; color: #{$shade}{$shade}{$shade};">
      <xsl:apply-templates/>
    </div>
  </xsl:template>

</xsl:stylesheet>
```

The key bit here is the literal attribute `style="font-weight: bold; color: #{$shade}{$shade}{$shade};"`. This is an attribute value template (AVT), which allows you to specify parts of the attribute in line as XPath expressions. In this case, I use references to the `shade` variable, which I've computed based on the month value in the `joined` attribute. I also use boldface in order to make the shading differences clearer. You might want to run this example in a stand-alone processor, so you can see the resulting HTML and understand the result of the logic. The source XML is `eg_2_7.xml` in `x-xmlcss3-tutorial-files.zip` (see [Downloads](#)), and again it's the same as the XML I've been using, but with an updated stylesheet PI. The image below shows the output in the browser.

Figure 3. Example of dynamic generation of properties using attribute value templates



Dynamic generation of properties using position()

You can also use the computational power of XSLT to control CSS properties. The following XML example (eg_2_8.xml in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) is a skeleton of a document with nested sections.

```
<?xml version="1.0" encoding="utf-8"?>
<?xml-stylesheet type="text/xml" href="eg_2_7.xsl"?>
<doc>
  <section>
    <para>This is section 1</para>
    <section>
      <para>This is section 1.1</para>
      <section>
        <para>This is section 1.1.1</para>
      </section>
    </section>
  </section>
  <section>
    <para>This is section 2</para>
  </section>
</doc>
```

```
<para>This is section 2.1</para>
</section>
<section>
  <para>This is section 2.2</para>
</section>
</section>
</doc>
```

The following XSLT (eg_2_8.xsl in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) increases the font size of paragraphs based on their relative position in the document. Again, it's somewhat contrived to show the generation of style properties using dynamic XSLT features, but it is a simple enough example for this tutorial.

```
<?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"/>
  <xsl:strip-space elements="*" />

  <xsl:template match="/">
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="doc">
    <html>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

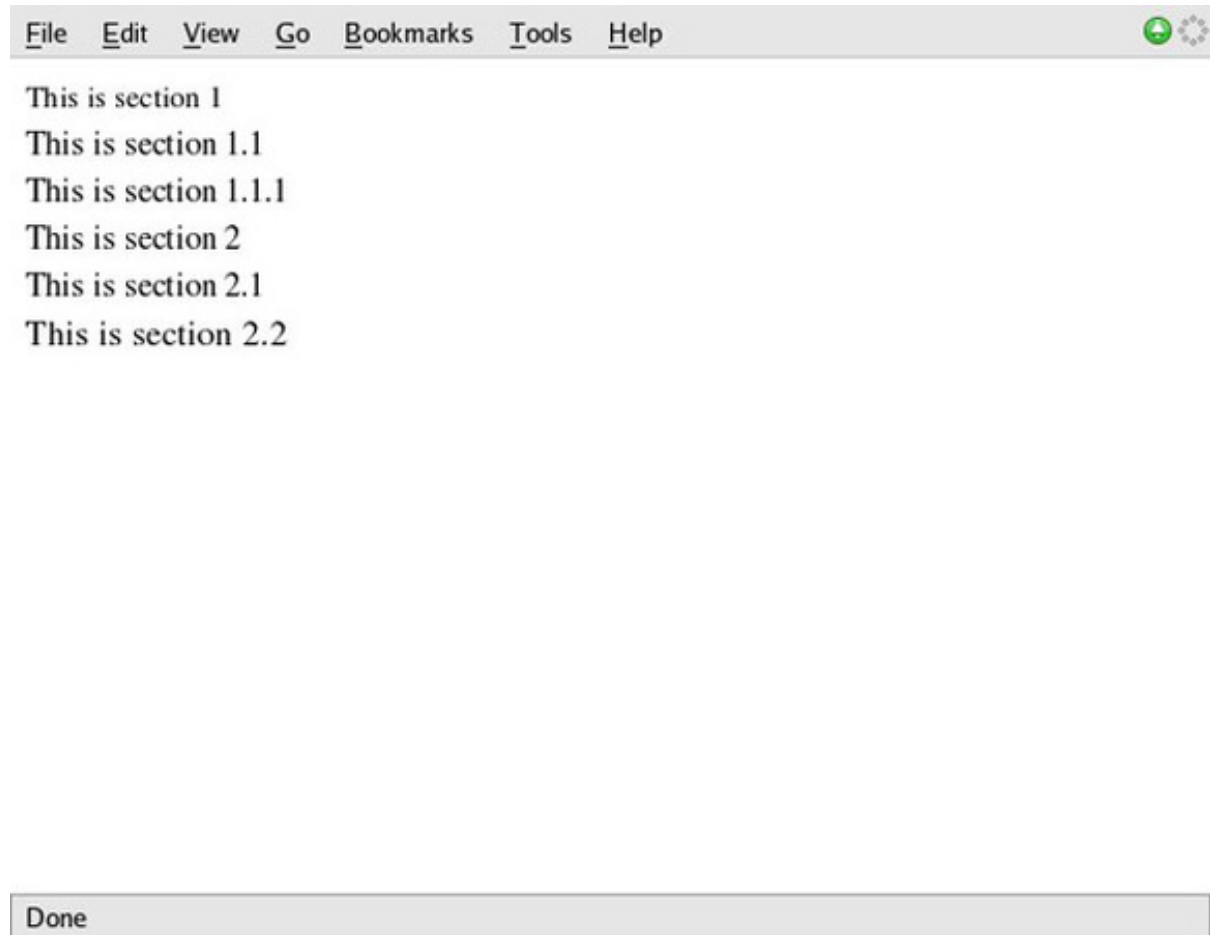
  <xsl:template match="section">
    <xsl:apply-templates>
      <xsl:with-param name="spos" select="position()" />
    </xsl:apply-templates>
  </xsl:template>

  <xsl:template match="para">
    <xsl:param name="spos" />
    <div style="font-size: 1.{$spos}0%;">
      <xsl:apply-templates/>
    </div>
  </xsl:template>

</xsl:stylesheet>
```

The `section` template keeps track of the position of the section within the context node list by passing it on as a parameter when templates are applied to child nodes. In the `para` template, this value is used in an AVT. The image below shows the output in the browser.

Figure 4. Example of dynamic generation of properties of properties using `position()`



Section 3. Using XSLT to generate HTML using full CSS stylesheets

Use XSLT to generate an in-line CSS stylesheet

CSS stylesheets are a more modular approach to applying style to HTML than `style` attributes. One way of incorporating a stylesheet into HTML output is to include it in a `style` element in the document's head. The following XSLT example (eg_3_1.xsl in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) operates on the same XML example (eg_3_1.xml in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) as in [XSLT through the browser](#), but with an updated stylesheet PI.

```
<?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"/>
<xsl:strip-space elements="*" />

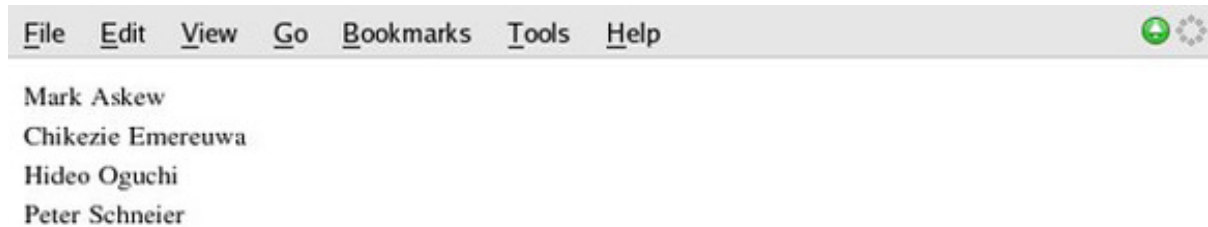
<xsl:template match="club-members">
  <html>
    <head>
      <style type="text/css">
        <xsl:comment>
div {
  color: blue;
}
        </xsl:comment>
      </style>
    </head>
    <body>
      <xsl:apply-templates/>
    </body>
  </html>
</xsl:template>

<xsl:template match="member">
  <div>
    <xsl:apply-templates/>
  </div>
</xsl:template>

</xsl:stylesheet>
```

The CSS is now a complete (although small) stylesheet that appears directly in the document header. It's in the body of a `style` element according to the HTML spec, and it's placed in a comment using the `xsl:comment` output instruction. This is a recommended precaution for increased browser compatibility. The resulting CSS stylesheet now applies to the entire HTML document, including the `div` elements generated for each member. The result of viewing this XML file in the browser should be the same as in [Stand-alone XSLT processing](#), but in this case there is a problem with current versions of Firefox. If you go straight to the XML file in the browser, you get the following, incorrect display:

Figure 5. Example of how to generate an in-line CSS stylesheet with XSLT



Done

If instead you generate the resulting HTML from a stand-alone processor (eg_3_1.html in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) and view this directly with Firefox, you get the expected display, with blue text.

Use HTML class attributes for more specific CSS selection

In [Create CSS style attributes with the xsl:attribute instruction](#), I demonstrated how to use XSLT to differentiate the CSS style attributes, in that case to highlight in boldface members who joined in 2005. You can allow for such specific treatment in a somewhat different way when generating HTML and a CSS stylesheet. The trick is to use the HTML `class` attribute, which can then be used in your CSS selectors.

The following XSLT (eg_3_2.xsl in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) highlights in boldface those members who joined in 2005, maintaining the blue text color in all cases.

```
<?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" />
<xsl:strip-space elements="*" />

<xsl:template match="club-members">
  <html>
    <head>
      <style type="text/css">
        <xsl:comment>


The xsl:if element uses the XPath you saw earlier, substring-before(@joined, '-') = 2005. This time a class attribute is added if the date matches, and in the CSS a selector div.new is used to specify div elements with that particular class attribute.



The result of processing the XML example (eg_3_2.xml in x-xmlcss3-tutorial-files.zip, see Downloads) and viewing the resulting HTML file in the browser is the same output as in Create CSS style attributes with the xsl:attribute instruction.



## Use HTML ID attributes for more specific CSS selection



In section 2, panels 4 and 5 of the previous tutorial, I discussed CSS ID selectors. You can also use these when generating HTML and CSS from XML. The following XSLT (eg_3_3.xsl in x-xmlcss3-tutorial-files.zip, see Downloads) highlights a couple of the members in an individual way.



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



Use Cascading Stylesheets to display XML  
© Copyright IBM Corporation 2005. All rights reserved.



Trademarks  
Page 16 of 30


```

```
<xsl:output method="html" />
<xsl:strip-space elements="*" />

<xsl:template match="club-members">
  <html>
    <head>
      <style type="text/css">
        <xsl:comment>


This time an expression in an AVT extracts the surname from each entry and uses it as an ID for the div. The CSS stylesheet then uses ID selection for a couple of the IDs. This is again a contrived example -- the code to extract the surname is not reliable. For example, a name such as "John Fitzgerald Kennedy" would cause problems because the resulting ID would contain an illegal space. Also, the very idea of the example probably creates too close a binding between the content and the presentation, by embedding such content specifics as the surname in the CSS.



The result of processing the XML example (eg_3_3.xml in x-xmlcss3-tutorial-files.zip, see Downloads) and viewing the resulting HTML file in the browser follows:



Figure 6. Example of how to use HTML class attributes for more specific CSS selection

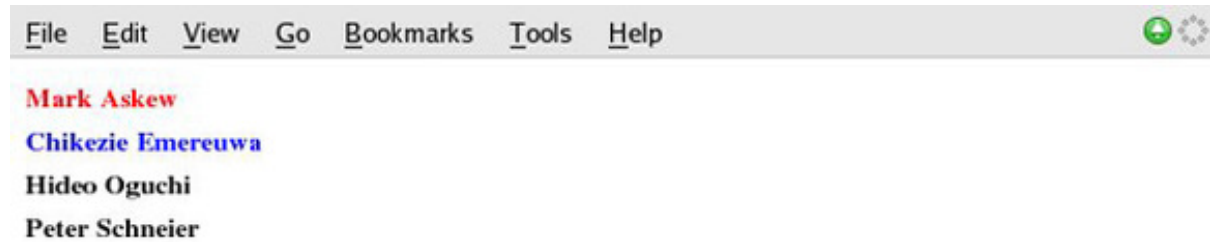


Use Cascading Stylesheets to display XML  
© Copyright IBM Corporation 2005. All rights reserved.



Trademarks  
Page 17 of 30


```



Done

Render parts of the CSS with dynamic techniques

If you need to set up some aspects of the CSS based on dynamically computed details, you can use XSLT's output facilities as needed. If so, you should probably move the code to a named template for clarity and modularity. The following XSLT (eg_3_4.xml in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) highlights a couple of the members in an individual way.

```
<?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"/>
  <xsl:strip-space elements="*" />

  <xsl:template match="club-members">
    <html>
      <head>
        <xsl:call-template name="style"/>
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </template>
</xsl:stylesheet>
```

```

    </html>
  </xsl:template>

  <xsl:template match="member">
    <div>
      <xsl:apply-templates/>
    </div>
  </xsl:template>

  <xsl:template name="style">
    <style type="text/css">
      <xsl:comment>
        <xsl:if test="not(/club-members/member)">
<!-- Make the background red if the document is empty -->
body {
  background-color: red;
}
      </xsl:if>
div {
  color: blue;
}
      </xsl:comment>
    </style>
  </xsl:template>
</xsl:stylesheet>

```

The CSS generation instructions are now moved to a named template, which `call-template` invokes. A CSS rule to paint the background red is only generated if no member elements are present, according to the XPath expression `not(/club-members/member)`. An example of such an empty file is `eg_3_4_1.xml` in `x-xmlcss3-tutorial-files.zip` (see [Downloads](#)); `eg_3_4.xml` is based on the examples you have used thus far. You can use these to explore the effect on the result, whether or not the document contains members.

Reference an external CSS stylesheet

Many CSS files are loaded from a separate file. To load these files, you can add a stylesheet link instruction to the header of the HTML document. The following XSLT (`eg_3_5.xsl`) does just that, referring to `eg_3_5.css` in `x-xmlcss3-tutorial-files.zip` (see [Downloads](#)).

```

<?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"/>
  <xsl:strip-space elements="*" />

  <xsl:template match="club-members">
    <html>
      <head>
        <link rel="stylesheet" href="eg_3_5.css"
          type="text/css" media="screen" />
      </head>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
  </xsl:template>

```

```
</html>
</xsl:template>

<xsl:template match="member">
  <div>
    <xsl:apply-templates/>
  </div>
</xsl:template>
</xsl:stylesheet>
```

Again, you can use AVTs when generating any of the literal attributes in the `link` element. You might, for example, use one to dynamically generate the file specified in `href`. You can also have multiple stylesheet links, as discussed in the [previous tutorial](#) (section 3, panels 4 and 5).

Generate a separate CSS document using EXSLT's document output extension

If you're willing to venture beyond the boundaries of XSLT 1.0 itself, you can generate a CSS stylesheet in a separate file. XSLT has a nice extension mechanism with which you can define additional capabilities. EXSLT is a standard set of such extensions defined in an implementation-agnostic way, and supported by many processors. One EXSLT extension generates a new output document. The following XSLT (eg_3_6.xsl in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) uses this to create a CSS file, eg_3_6.css.

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

  <xsl:output method="html"/>
  <xsl:strip-space elements="*" />

  <xsl:template match="club-members">
    <html>
      <body>
        <xsl:apply-templates/>
      </body>
    </html>
    <!-- Now generate the stylesheet in a separate document -->
    <xsl:call-template name="style"/>
  </xsl:template>

  <xsl:template match="member">
    <div>
      <xsl:apply-templates/>
    </div>
  </xsl:template>

  <xsl:template name="style">
    <exsl:document href="eg_3_6.css" method="text">
  div {
    color: blue;
  }
  </exsl:document>
  </xsl:template>
```

```
}
  </exsl:document>
</xsl:template>

</xsl:stylesheet>
```

Remember to declare the EXSLT extension namespace, and to put the prefix in `extension-element-prefixes`. If you forget to do this, you'll either get a namespace error or the processor will mistake the extension instruction for a plain literal result element, which will be lumped into the HTML output as is. The `exsl:document` takes the same attributes as `xsl:output` and uses these output parameters to control writing to a new resource (file), given by the `href` attribute. In this case I use the `text` output method, which is suitable for CSS output since it doesn't use SGML-like mark-up. If the body of the extension includes any XSLT instructions, they are applied as usual, and you can thus compute output dynamically.

It probably only makes sense to test this XSLT with a stand-alone processor. For one thing, Firefox doesn't support EXSLT extensions. For another, most browsers have good security reasons to avoid extensions that create files in this way.

Section 4. Using XSLT to generate XML with CSS

Use XSLT to generate a CSS PI

As I discussed in the [previous tutorial](#), you can use CSS with a lot more than HTML. Similarly, XSLT can be used to generate a lot more than HTML (in an earlier panel I used it to generate a separate CSS file). You can use XSLT to create CSS in association with other output XML formats. The most general case is to use XSLT to set up the situation that I have been describing throughout this tutorial series: XML files that use CSS for browser display. The following XSLT (eg_4_1.xsl in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) is basically a tool that strips away the dates from the member files I've been using for most examples.

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

  <xsl:output method="xml" indent="yes"/>
  <xsl:strip-space elements="*" />

  <xsl:template match="/">
    <xsl:processing-instruction
      name="xml-stylesheet">
      <xsl:text>type="text/css" href="eg_4_1.css"</xsl:text>
```

```
    </xsl:processing-instruction>
    <xsl:apply-templates/>
  </xsl:template>

  <xsl:template match="*">
    <xsl:copy>
      <xsl:apply-templates/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Except for the template with `match="/"`, this XSLT code is a very general purpose attribute stripper. The template with `match="*"` will copy over any element and its contents without copying the attributes. The part of this code that's of most interest is `xsl:processing-instruction`, which allows you to create an XML PI from XSLT. The code should make sense if you consider the following output from that instruction:

```
<?xml-stylesheet type="text/css" href="eg_4_1.css"?>
```

A warning about the XML declaration

One of the most common sources of confusion is the XML declaration, which I've often heard people call the "XML processing instruction." This is incorrect. The XML declaration uses similar syntax to a PI, but it is not a PI, and you cannot create it with the `xsl:processing-instruction`. You can control generation of the XML PI by using the `xsl:output` element. If you use the XML output method, a PI is generated for you by default. You can turn this off by specifying `omit-xml-declaration="yes"` in the output element. You can set the character encoding for the output -- and thus the encoding specified in the declaration -- in the `encoding` attribute. You can set the standalone declaration in the `standalone` attribute.

While I'm discussing `xsl:output`, I might as well touch on some other useful matters that are not always so well known to XSLT users. You can also generate a document type declaration by using the `doctype-public` and `doctype-system` attributes. In most XSLT in this tutorial, you have seen the HTML output method, and in such cases indentation is added by default so that the output is pretty printed. In the examples in this section, the output method is generally XML, which has no such pretty-printing by default. You can override either default by manually specifying the `indent` attribute. You probably won't need to use the `media-type` attribute except in the most advanced cases. I shall cover another very important attribute in the next section, [Advanced notes](#).

Generate SVG with style

In the [previous tutorial](#) (section 5, panel 2), I discussed Scalable Vector Graphics (SVG), a sophisticated vector graphics format in XML. SVG uses CSS to specify many aspects of style in graphics elements, from text fonts to shape colors. You can link to an external CSS stylesheet from SVG, or you can embed the entire stylesheet within the SVG document. The following XSLT (eg_4_3.xsl in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) is a very unsophisticated example that generates an SVG example like one from the last tutorial.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:svg="http://www.w3.org/2000/svg"
  xmlns="http://www.w3.org/2000/svg">

  <xsl:output method="xml" indent="yes"
    doctype-public="-//W3C//DTD SVG 1.1//EN"
    doctype-system="http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd"
    cdata-section-elements="svg:style"/>
  <xsl:strip-space elements="*" />

  <xsl:template match="/">
    <svg version="1.1"
      width="10cm" height="5cm" viewBox="0 0 1000 500">
      <defs>
        <xsl:call-template name="style" />
      </defs>
      <rect x="200" y="100" width="600" height="300" />
    </svg>
  </xsl:template>

  <xsl:template name="style">
    <style type="text/css">
      rect {
        fill: red;
        stroke: blue;
        stroke-width: 3
      }
    </style>
  </xsl:template>

</xsl:stylesheet>
```

The XSLT is constructed so as to ignore the XML source document; you can pick any well-formed XML document you like as the source. It uses some of the output parameters discussed in [A warning about the XML declaration](#). It also uses one I have not yet covered, `cdata-section-elements`. This attribute takes a space-delimited list of qualified names for elements that are to be generated with their text children wrapped in a CDATA section. The SVG namespace declaration is repeated in order to serve this need: One declaration is for the `svg` prefix, which is used in the `cdata-section-elements` attribute; the other is a default namespace declaration so that the output can also use the default namespace, and avoid prefixes. The `cdata-section-elements` attribute specifies that the CSS stylesheets in SVG `style` elements will be in CDATA sections, as recommended. The rest of the XSLT is fairly straightforward. The file `eg_4_3.svg` in `x-xmlcss3-tutorial-files.zip` (see [Downloads](#)) is the output I got using 4XSLT.

Section 5. Advanced notes

CSS high characters

In the [first tutorial of this series](#) (section 4, panel 3) I discussed the different mechanisms for escaping characters in HTML, XML, and CSS. HTML and XML have similar methods for escaping special characters, but CSS uses a different mechanism. For example, to unmistakably express a new line in XML or HTML you would use the character entity `
` (or `
`). In CSS, you would use the escape sequence `\A`. This is not something you often need to worry about. The uncommon case of generated content (also discussed in the first tutorial) is the most likely situation in which you'll need to worry about character escaping.

For the most part, you should be able to just emit the needed characters directly, without worrying about escaping. CSS is a Unicode application, as is XML (and thus XSLT). The following XSLT snippet generates an opening quotation mark, which can be a direct part of output CSS:

```
<xsl:text>&#x201C;</xsl:text>
```

The resulting character is the equivalent of the following character, expressed as a CSS escape:

```
\201C
```

If you do decide that you want to turn characters in the XML source or the XSLT into actual CSS character escapes, one possible approach is through lookup tables (see [Resources](#)).

Set up JavaScript in the output

In the [previous tutorial](#) (section 4, panels 4, 5, and 6), I showed examples of manipulating CSS properties with JavaScript. If you are using XSLT to generate HTML with CSS, you can rely on broader browser support because that combination has less gray area than JavaScript with XML content. The following XSLT (eg_5_2.xsl in x-xmlcss3-tutorial-files.zip, see [Downloads](#)) demonstrates a neat trick in which members are shown in blue text, but the text turns to red if you run your mouse over any entry.

```
<?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"/>
  <xsl:strip-space elements="*" />

  <xsl:template match="club-members">
    <html>
      <head>
        <xsl:call-template name="style" />
      </head>
      <body>
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>

  <xsl:template match="member">
    <div onmouseover="this.style.color='red';"
      onmouseout="this.style.color='blue';">
      <xsl:apply-templates />
    </div>
  </xsl:template>

  <xsl:template name="style">
    <style type="text/css">
      <xsl:comment>
div {
  color: blue;
}
      </xsl:comment>
    </style>
  </xsl:template>
</xsl:stylesheet>
```

The key is in the JavaScript invocation attributes added to the `div` elements. The `onmouseover` attribute specifies JavaScript to be executed when the user's mouse enters the box area of the element; `onmouseout` specifies JavaScript to be executed when the user's mouse leaves the box area of the element. The JavaScript simply uses `this.style` to access the CSS information for the DOM object that represents the current element, and you can go ahead and manipulate the properties as you will. Remember that if you need CSS properties whose names are not proper JavaScript identifiers, you should use DOM functions such as `setProperty` (discussed in the previous tutorial).

If you process the XML example (`eg_5_2.xml` in `x-xmlcss3-tutorial-files.zip`, see [Downloads](#)) and view the resulting HTML file (`eg_5_2.html`), you'll see that the initial display in the browser is the same as in [Stand-alone XSLT processing](#). Try it yourself and run the mouse over the text to see the color roll-over effect.

Manipulate class with JavaScript

You can move some of the style specifics from the dynamic manipulation in [Set up JavaScript in the output](#) into the stylesheet. The following XSLT (`eg_5_3.xsl` in

x-xmlcss3-tutorial-files.zip, see [Downloads](#)) demonstrates the same neat trick, where the members are in blue text, but the text turns to red if you run your mouse over any entry. This time it takes a somewhat different approach.

```
<?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" />
  <xsl:strip-space elements="*" />

  <xsl:template match="club-members">
    <html>
      <head>
        <xsl:call-template name="style" />
      </head>
      <body>
        <xsl:apply-templates />
      </body>
    </html>
  </xsl:template>

  <xsl:template match="member">
    <div class="nohover" onmouseover="this.className='onhover';"
      onmouseout="this.className='nohover';">
      <xsl:apply-templates />
    </div>
  </xsl:template>

  <xsl:template name="style">
    <style type="text/css">
      <xsl:comment>
div.nohover {
  color: blue;
}
div.onhover {
  color: red;
}
      </xsl:comment>
    </style>
  </xsl:template>

</xsl:stylesheet>
```

This time rather than manipulate the style properties directly, you manipulate the class. That way you can combine all sorts of properties in the class, take advantage of CSS cascade, and so forth. You also avoid complications like CSS property names that are not proper JavaScript identifiers. This is generally a much cleaner solution. I set up two classes in the stylesheet: `nohover` for when the mouse is not over the element, and `onhover` for when it is. Remember to initialize the `div` to the former class in the XSLT literal element; then in the `onmouseover` and `onmouseout`, the JavaScript sets the class of the element rather than individual CSS properties.

Because of accessibility concerns, be sure not to rely too heavily on such roll-over effects. Use them for enhancement of information that is also expressed conventionally.

Section 6. Wrap up

Summary

In this tutorial, you learned how to work with XSLT and CSS, picking up many useful techniques, including how to:

- Use stand-alone XSLT processors to generate HTML with CSS
- Generate HTML with CSS directly within XSLT-capable browsers
- Generate HTML with CSS in style attributes
- Use attribute sets to generate groups of related attributes together
- Use XSLT computational features to dynamically generate CSS details
- Generate HTML with CSS in an in-line stylesheet
- Generate HTML with CSS in an external stylesheet, including the use of EXSLT to create such an external document from XSLT
- Generate HTML that uses `class` or `id` attributes as hooks for the CSS
- Maintain modularity in code that generates CSS
- Generate XML declarations and stylesheet PIs
- Work with the CSS character model
- Set up dynamic JavaScript manipulation of CSS in HTML that's generated through XSLT

This is a wide range of topics, but it merely scratches the surface of what you can achieve with XML, XSLT and CSS using current technology. I strongly encourage you to push the boundaries through a lot of practice and experimentation, using the example code presented here as a starting point.

Downloads

Description	Name	Size	Download method
Sample code	x-xmlcss3-tutorial-files.zip	19KB	HTTP

[Information about download methods](#)

Resources

- If you're unfamiliar with XML, start with the many helpful resources in the developerWorks [New to XML](#) page, including Doug Tidwell's popular [Introduction to XML](#) tutorial (August 2002).
- Learn the basics of CSS and CSS with XML. See the first tutorial in this series [Use Cascading Stylesheets to display XML, Part 1](#), including the Resources panels of that tutorial, to get started. This tutorial introduces the use of CSS to style XML in browsers (November 2004).
- Continue with [Use Cascading Stylesheets to display XML, Part 2](#), which covers advanced topics for the use of CSS to style XML in browsers (February 2005).
- If you're unfamiliar with XSLT, take a look at [Get started with XPath](#) by Bertrand Portier (May 2004), move on to [Investigating XSLT: The XML transformation language](#) by LindaMay Patterson (August 2001), and finish up your introductory tour with [Create multi-purpose Web content with XSLT](#) by Nicholas Chase (March 2003).
- While you're at it, bookmark the W3C's [XSL reference page](#).
- Also see [Adding a touch of style](#), a great introduction to CSS by Dave Raggett of the W3C. The W3C staff (Bert Bos to be specific) also wrote [How to add style to XML](#), a sketchy introduction, but with some useful examples.
- Follow along by running the tutorial examples yourself. The author tested the CSS in this tutorial with [Firefox](#), a popular and free Web browser that's available for Windows, Mac OS X, Linux, and other platforms. Firefox is based on Mozilla's rendering engine, which has always been regarded as a very CSS-compliant browser. He used [4Suite 1.0b1](#) as the stand-alone XSLT processor.
- Bookmark and refer to the formal CSS 2 specification, [Cascading Style Sheets, level 2: CSS2 Specification](#). Many parts of the specification are clear and readable.
- Keep an eye on developments in [CSS 3](#), the next version of CSS. Many browsers already experimentally implement features proposed on CSS 3 working drafts. Read [CSS 3 Selectors](#) by Russell Dyer for a good overview of the selectors (though CSS 3 is more than just new selectors). But before CSS 3 is complete, you can expect to see [CSS 2.1](#), a minor update.
- Review [Associating Style Sheets with XML documents Version 1.0](#) (W3C Recommendation, June 1999), which is the most authoritative document that describes how to specify style for XML documents.
- For an overview of push versus pull XSLT techniques, see [this article](#).
- To get started with EXSLT, see Uche Ogbuji's article [EXSLT by example](#)

(developerWorks, February 2003). Particularly relevant for this tutorial is [The exsl:document element](#).

- Take a closer look at Scalable Vector Graphics (SVG) with the developerWorks tutorials [Introduction to Scalable Vector Graphics](#) (March 2004) and [Interactive, dynamic Scalable Vector Graphics](#) (June 2003), both by Nicholas Chase.
- For more on the XSLT lookup tables approach, see Uche Ogbuji's tip [Packaging XSLT lookup tables as EXSLT functions](#) (developerWorks, January 2005), and work backwards through resources posted there.
- Refer to [Document Object Model \(DOM\) Level 2 Style Specification](#) (W3C Recommendation, November 2000), which details how CSS can be processed using DOM (for example, through JavaScript in a browser). Mozilla has excellent DOM and CSS references. See, for example, the [Gecko DOM Reference](#) (Gecko is Mozilla's rendering engine). And in case you run into problems, bookmark the [Mozilla XSLT FAQ](#).
- Use the [W3C CSS Validation Service](#), "a free service that checks Cascading Style Sheets (CSS) in (X)HTML documents or standalone for conformance to W3C recommendations."
- [Browse for books](#) on these and other technical topics.
- Find more XML and Web development resources on the developerWorks [XML](#) and [Web architecture](#) zones.
- Finally, find out how you can become an [IBM Certified Developer in XML and related technologies](#).

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 out more about Mr. Ogbuji at his Weblog [Copia](#), or contact him at uche.ogbuji@fourthought.com.