

Display XML with Cascading Stylesheets, Part 2: Use Cascading Stylesheets to display XML

Advanced techniques to present XML in Web browsers

Skill Level: Introductory

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

25 Feb 2005

In a previous tutorial, Uche Ogbuji showed how to use Cascading Stylesheets (CSS) to display XML in browsers, presenting basic techniques. However, as you get familiar with CSS, you'll find numerous tricks, traps, and nuances. This tutorial builds on the basics in the earlier one to cover intermediate and advanced topics like differences in compliance and rendering details across browsers, handling of different media such as aural (speech) and print, and interaction with other browser and XML technologies such as XSLT and JavaScript.

Section 1. Introduction

The many nuances of CSS

In [a previous tutorial](#), Uche Ogbuji showed how to use Cascading Stylesheets (CSS) to display XML in browsers, presenting basic techniques. However, as anyone who has tried to master CSS in the world of HTML and JavaScript can attest, you'll find numerous tricks, traps, and nuances. The same is true when using CSS with XML. You have to watch out for:

- Differences in compliance and rendering details across browsers

- Handling of different media such as aural (speech) and print
- Interaction with other browser and XML technologies such as XSLT and JavaScript

You may also be interested in matters such as:

- The use of CSS embedded in important XML vocabularies such as SVG
- Linking, forms, and other user interface features relating to browser styling

This tutorial builds on the basics in the earlier one to cover these intermediate and advanced topics.

Who should take this tutorial?

A lot of XML eventually finds its way to Web browsers in one way or another, and 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, and have been embraced by browser vendors as the best-supported means of displaying XML. CSS is also used in other XML vocabularies such as SVG and XForms.

Anyone who works with XML should take this tutorial. Even if CSS doesn't cover your needs for production Web publishing, it is a great tool for debugging and experimentation. It also has rich interaction with other XML technologies, and you are likely to run into CSS in many unexpected cases.

Prerequisites

This tutorial assumes knowledge of XML, and some basic knowledge of CSS. If you aren't familiar with XML, I recommend that you first take the "[Introduction to XML](#)" tutorial here on developerWorks.

You should also be familiar with the use of CSS to style XML in browsers. To learn this, take the previous tutorial, "[Use Cascading Stylesheets to display XML](#)".

I recommend, but do not require, familiarity with XML Namespaces and XSLT. If you aren't familiar with XSLT, you might want to take the tutorial "[Create multi-purpose Web content with XSLT](#) tutorial."

I highly recommend that you try out the examples in this tutorial. To do so you need a Web browser that supports XML and CSS Level 2 or better. Where applicable, I present the output of examples using Firefox 1.0 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 known to be a very CSS-compliant browser.

About the XML/CSS examples in this tutorial

In this tutorial you will see many examples of CSS files. You can find all the files used in this tutorial in the file, x-xmlcss2-tutorial-files.zip (see [Downloads](#)). In this package, all files start with a prefix indicating the section that covers them and the order the examples appear within the section. For example, files from the first example in the third section are named starting with "eg_3_1".

Files with the .css extension are Cascading Stylesheets, and files with the .xml extension are sample XML documents that you can display using the CSS file with the same prefix, if one is present. Some of the XML files are not associated with a matching CSS file; for example, they may demonstrate embedded CSS. A few of the files use more specialized extensions such as .svg for SVG files.

I do list the example files in each panel, so you can easily locate and experiment with the examples as you take this tutorial.

Section 2. Intermediate CSS selectors and rules in action

Partial attribute selectors

In the [previous tutorial](#), you learned about attribute selectors -- for example, `author[member='yes']` selects elements with a `member` attribute value of `yes`. A useful twist on this is the ability to select parts of attribute values. Take the following XML (`eg_2_1.xml` in `x-xmlcss2-tutorial-files.zip`, see [Downloads](#)), a list of club members with an attribute for the date when each member joined the club.

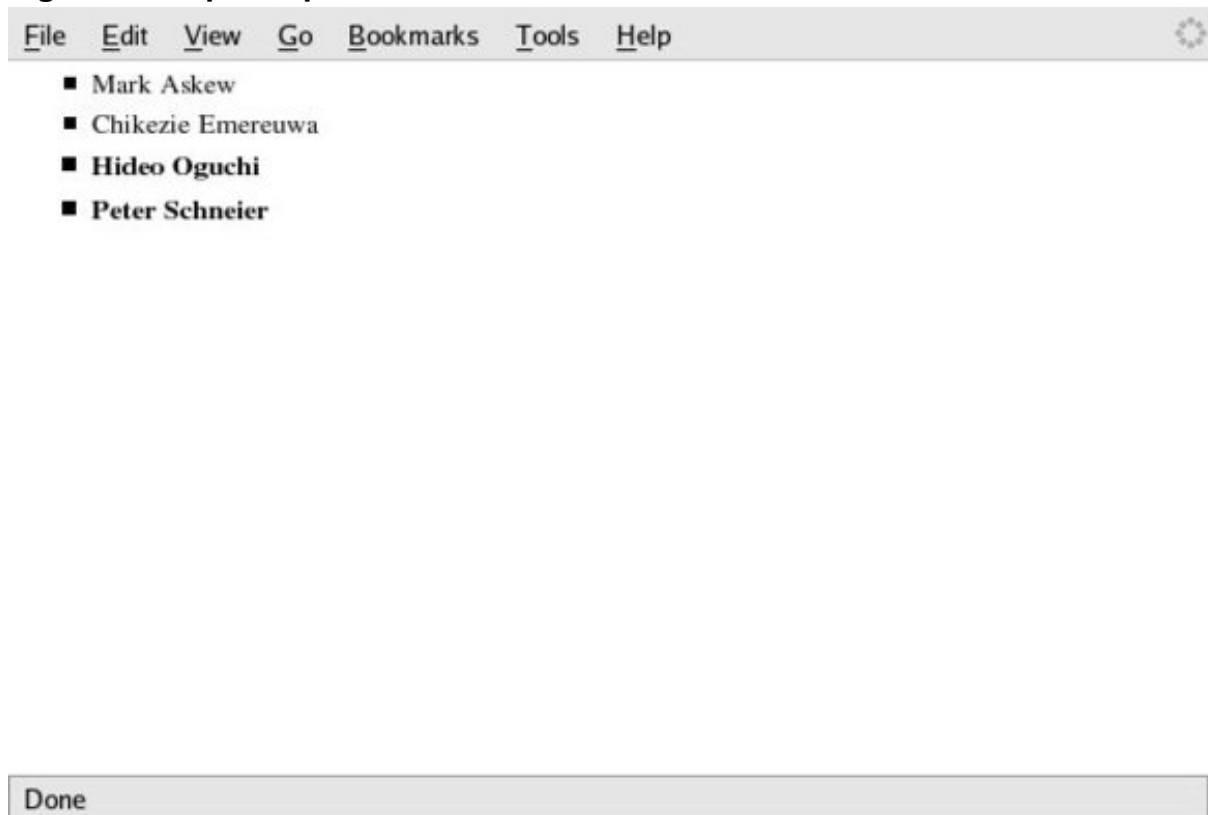
```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/css" href="eg_2_1.css"?>
<club-members>
  <member joined='2004-04-19'>Mark Askew</member>
  <member joined='2004-11-25'>Chikezie Emereuwa</member>
  <member joined='2005-01-05'>Hideo Oguchi</member>
  <member joined='2005-01-20'>Peter Schneier</member>
</club-members>
```

The following CSS (eg_2_1.css in the file, x-xmlcss2-tutorial-files.zip, see [Downloads](#)) highlights in boldface members who joined in 2005.

```
* { display: inherit; }
club-members { display: block; }
member {
  display: list-item;
  list-style: square outside;
  margin-left: 3em;
}
member[joined="2005"] { font-weight: bold; }
```

The `member` selector establishes that all members are rendered as lists, and in addition the selector `member[joined="2005"]` matches all attributes where 2005 is the initial portion, delimited by a hyphen. The image below shows the output.

Figure 1. Output of partial attribute selectors



Select by language code

Unfortunately, the form of attribute selector just covered is very limited. It can only be used to select the first item in a hyphen-delimited attribute value. So

`member[joined]="2005"]` matches 2005-01-05 but not 01-05-2005 nor 2005 01 05. Even with such a limitation, this technique proved useful in [Partial attribute selectors](#) because the dates in the sample XML are expressed using ISO 8601 representation; I happened to be focusing on the year, which is the first part of dates in this representation.

This form of attribute selector is so severely limited because it was developed to support selection by language specified using ISO 639 language codes. As an example of such codes, `en-US` is a code for the United States variant of English, `en-GB` is a code for the British variant of English, and `en` is a code for any English variant. This means that you can use CSS to put a border around any `section` element with an `xml:lang` specifying that it is in English. Here's an example:

```
section[xml:lang]="en" { border: thin red solid; }
```

Select by presence of a word in attribute values

Another form of attribute selector checks for the presence of a particular word. It is especially effective when used with attributes of DTD type `NMTOKENS` or the equivalent. This type is a whitespace-delimited list of tokens (words, essentially). As an example, the following XML (`eg_2_3.xml` in the file `x-xmlcss2-tutorial-files.zip`, see [Downloads](#)) is a list of fashion designers. In this example, an attribute gives the locations of boutiques (incidental information so it's okay from an XML design point of view to place the locations in an attribute).

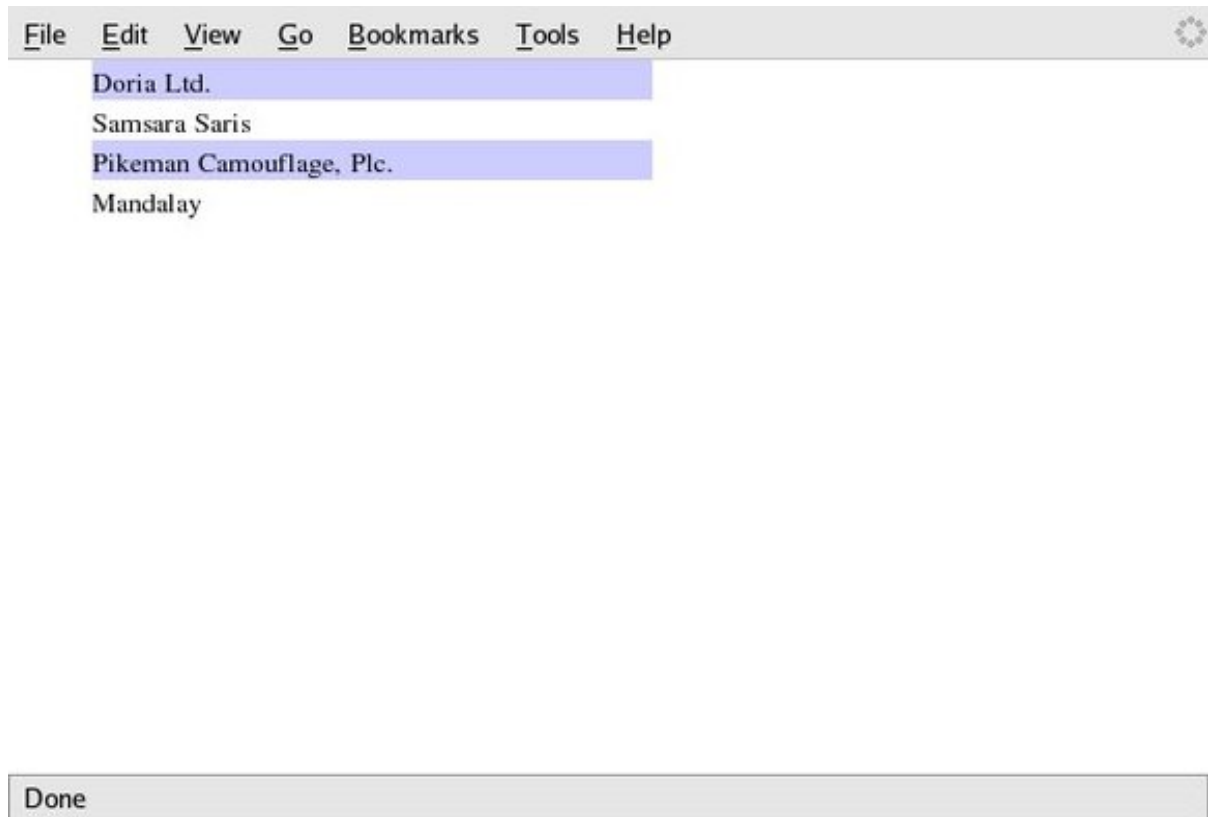
```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/css" href="eg_2_3.css"?>
<designers>
  <designer locations="Paris London Manhattan">Doria Ltd.</designer>
  <designer locations="Mumbai Chennai Jakarta">Samsara Saris</designer>
  <designer locations="London Moscow">Pikeman Camouflage,
Plc.</designer>
  <designer locations="Milan Paris Tokyo">Mandalay</designer>
</designers>
```

The following CSS (`eg_2_3.css` in the file `x-xmlcss2-tutorial-files.zip`, see [Downloads](#)) gives a sky blue background to designers who have boutiques in London.

```
* { display: inherit; }
designers { display: block; }
designer {
  margin-left: 3em;
  width: 20em;
}
designer[locations~="London"] { background-color: #ccccff; }
```

The selector `designer[locations~="London"]` matches all attributes where `London` appears as a space-delimited word. You can see in the example that this also works if the target word happens to be at the start of the attribute value. A key limitation with this sort of selector is that you can't look for values with spaces in them, such as `locations~=["New York"]`. The image below shows the output.

Figure 2. Output for selector based on presence of a particular word



ID selectors and XML

As the CSS 2.1 spec puts it: *Document languages may contain attributes that are declared to be of type ID. What makes attributes of type ID special is that no two such attributes can have the same value; whatever the document language, an ID attribute can be used to uniquely identify its element. In HTML all ID attributes are named "id"; XML applications may name ID attributes differently, but the same restriction applies.* The easiest way available today to create IDs for XML is through a DTD. Going back to the membership list example, the following XML (`eg_2_4.xml` in the file `x-xmlcss2-tutorial-files.zip`, see [Downloads](#) gives each member a unique identifier in the `tag` attribute.

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/css" href="eg_2_4.css"?>
<!DOCTYPE club-members [
  <!ELEMENT club-members (member*)>
```

```
<!ELEMENT member (#PCDATA)>
<!ATTLIST member tag ID #REQUIRED>
]>
<club-members>
  <member tag='ma1'>Mark Askew</member>
  <member tag='ce1'>Chikezie Emereuwa</member>
  <member tag='ho1'>Hideo Oguchi</member>
  <member tag='ps1'>Peter Schneier</member>
  <member tag='ce2'>Carlos Ezra</member>
</club-members>
```

The following CSS (eg_2_4.css in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) uses a different list bullet to display the club founder, "Chikezie Emereuwa".

```
* { display: inherit; }

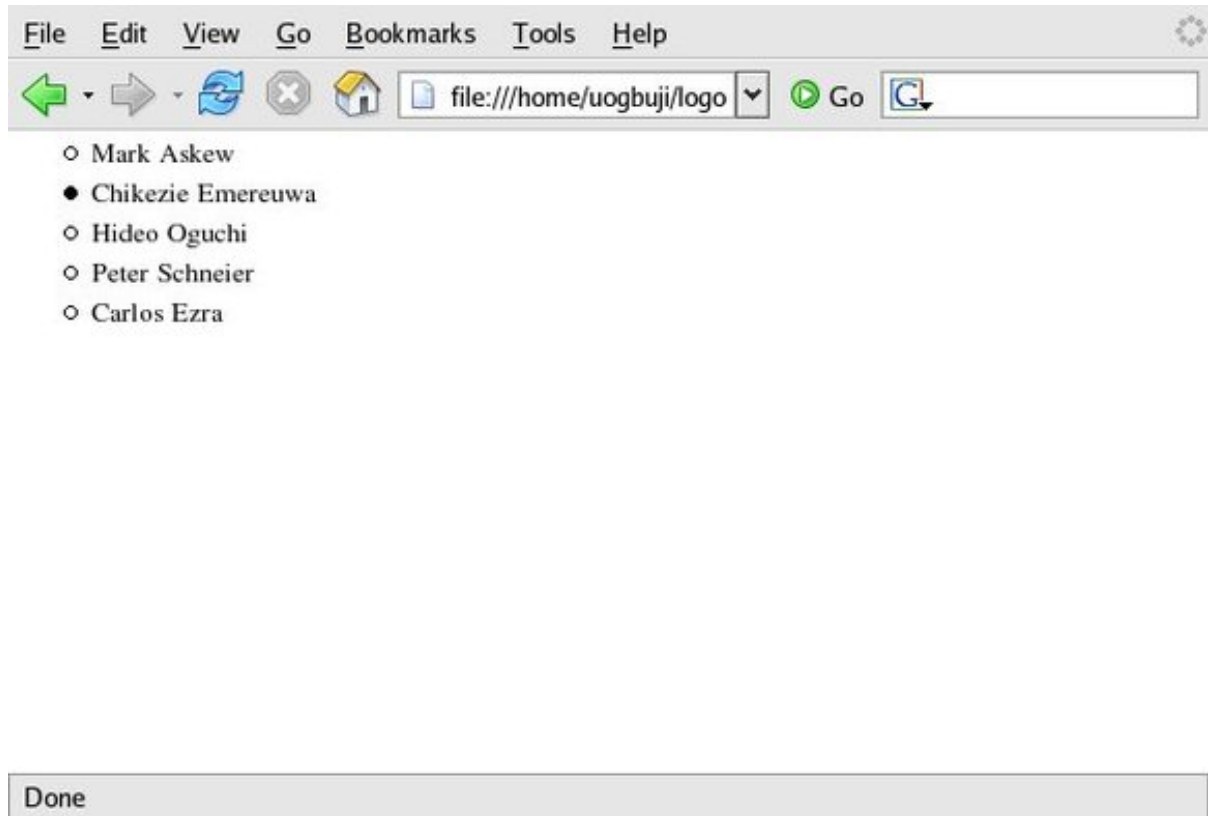
club-members { display: block; }

member {
  display: list-item;
  list-style: circle outside;
  margin-left: 3em;
}

member#ce1 { list-style: disc outside; }
```

The selector `member#ce1` grabs the given ID attribute value. Because of the semantics of ID attributes, you know it's only going to match one element. The image below shows the output.

Figure 3. Output based on specific ID attribute values



Coming soon? xml:id

DTDs have always been one of the spotty areas in browser support of XML. For example, Mozilla-based browsers such as Firefox still don't support the external subset of DTDs; this is one place where many XML vocabularies such as XHTML, DocBook, and MathML declare ID-type attributes. Because of these quirks and many other issues, the W3C has started working on a system of ID attributes that does not require DTDs or other schema declarations. It prescribes a special attribute, `xml:id` (names starting with "xml" are reserved by the XML 1.0 specification). This attribute would always have unique identifier semantics.

Currently, no CSS implementations support `xml:id`, which is understandable because it's still just a Candidate Recommendation. I expect because of the simplicity of the specification, many implementations will emerge fairly quickly, so it's worth being familiar with the concept. The following XML (eg_2_5.xml in the [filex-xmlcss2-tutorial-files.zip](#), see [Downloads](#)) is the membership list example from [ID selectors and XML](#), but using `xml:id`.

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/css" href="eg_2_4.css"?>
<club-members>
  <member xml:id='mal'>Mark Askew</member>
  <member xml:id='cel'>Chikezie Emereuwa</member>
```

```
<member xml:id='ho1'>Hideo Oguchi</member>  
<member xml:id='ps1'>Peter Schneier</member>  
<member xml:id='ce2'>Carlos Ezra</member>  
</club-members>
```

You do not need to change the CSS from [ID selectors and XML](#) (eg_2_4.css in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) for it to recognize the new ID syntax. CSS is not involved in how the document language determines what is an attribute of type ID. For example, HTML and XHTML merely say that any attribute named `id` is of type ID. As `xml:id` solidifies, it will be important in many areas besides CSS, including JavaScript, XSLT, and XInclude.

Section 3. CSS and media types

CSS goes beyond the GUI Web browser

In the [previous tutorial](#) I covered a bit of the broader spectrum of CSS. It's not just about the classic GUI Web browsers that most people use. CSS is designed to accommodate several media, to support accessibility for the disabled, to repurpose content to multiple forms, and for overall flexibility. The specification itself offers a clear summary of CSS's approach to multiple media:

One of the most important features of stylesheets is that they specify how a document is to be presented on different media: on the screen, on paper, with a speech synthesizer, with a Braille device, etc.

Certain CSS properties are only designed for certain media (e.g., the 'page-break-before' property only applies to paged media). On occasion, however, stylesheets for different media types may share a property, but require different values for that property. For example, the 'font-size' property is useful both for screen and print media. The two media types are different enough to require different values for the common property; a document will typically need a larger font on a computer screen than on paper. Therefore, it is necessary to express that a stylesheet, or a section of a stylesheet, applies to certain media types.

In this section you'll learn more about how CSS can apply to diverse media. I did not include sample screen shots in this section because it deals with media that can't be easily demonstrated in this tutorial format.

Use imports to specify media-specific styles

One way to specify rules for multiple media is to create multiple CSS files, each for a different medium, and import them into a master CSS. You may recall the following sample XML (eg_3_2.xml in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) from the [previous tutorial](#). It's a sketched-out format for published papers.

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/css" href="eg_3_2_master.css"?>
<paper>
  <prologue>
    <title>Faster than light travel</title>
    <subtitle>From fantasy to reality</subtitle>
    <author member='yes' e-mail="cemereuwa@nasa.gov">Chikezie Emereuwa</author>
    <author member='no' e-mail="okey.agu@navy.mil">Okechukwu Agu</author>
    <main-contact e-mail='cemereuwa@nasa.gov' />
    <abstract>All laws are meant to be broken, and according to our analysis,
    all the required antecedent technologies are in place for us to break
    the famous universal speed limit that derives from the Special Theory
    of Relativity.</abstract>
  </prologue>
  <section>
    <para>The work of Oguchi, Hu, Schneider et. al. already establishes
    the equivalent states of each elementary particle when traveling
    at superoptical velocities. The work of Baraka, Smith and Hossemi
    establish the effect of an omega hadron field in eliminating
    relativistic effects upon ordinary matter.</para>
  </section>
  <section>
    <para>We can construct an omega hadron field drive in a module that fits
    in the International Space Station high energy experiments wing,
    allowing us to demonstrate <abbreviation><short>FTL</short><long>faster
    than light</long></abbreviation> travel over distances of up
    to 12,000 meters.</para>
    <para>By September of 2020, we expect superoptical drives to burst
    from science fiction into the world of fact.</para>
  </section>
</paper>
```

The following CSS (eg_3_2_master.css in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) is the main stylesheet specified by the document. Its only purpose is to import media-specific stylesheets.

```
@import url("eg_3_2_paged.css") print;

* {display: inherit;}

paper {display: block;}

long {display: none;}

abbreviation {display: inline;}

title {
  text-align: left;
  font: small-caps 175% sans-serif;
}

subtitle {
  text-align: left;
  font: 150% sans-serif;
}

abstract { font-size: small; font-style: italic;}
```

```
author:before {
  content: 'Author: ';
  font-size: x-small;
  font-weight: lighter;
}

author {
  font-size: x-small;
  font-weight: lighter;
}
```

The `print` keyword in the `@import` rule instructs the processor to load the target stylesheet conditionally. The stylesheet is loaded only if the media type defined for the user agent matches a medium specified in the import. You can specify multiple media for an import by separating multiple media specifiers with commas. The special specifier `all` means that the import is always loaded, regardless of medium. The rest of the master stylesheet includes all the rules that are suitable for screen display. These rules will naturally also apply to print.

The following CSS (eg_3_2_paged.css in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) contains the print-medium-specific rules. It only has a single rule, to ensure a page break before the main paper content.

```
/* Basically create a separate title/cover page */
prologue { page-break-after: always }
```

Use the media rule to specify media-specific styles

You can also specify rules for different media within a single stylesheet file using the `@media` rule. The best way to understand this rule is by example. The sample XML is the same as in [Use imports to specify media-specific styles](#), but with a stylesheet processing instruction that loads the following CSS instead (eg_3_3.css in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)).

```
@media print {
  /* Basically create a separate title/cover page */
  prologue { page-break-after: always }
}

* {display: inherit;}

paper {display: block;}

long {display: none;}

abbreviation {display: inline;}

title {
  text-align: left;
  font: small-caps 175% sans-serif;
}

subtitle {
  text-align: left;
```

```

    font: 150% sans-serif;
  }
  abstract { font-size: small; font-style: italic;}

  author:before {
    content: 'Author: ';
    font-size: x-small;
    font-weight: lighter;
  }

  author {
    font-size: x-small;
    font-weight: lighter;
  }

```

The section within the @media rule is ignored if the user agent medium is not print.

Use the stylesheet processing instruction to specify media-specific styles

You might wish to specify media-specific styles right in the source document. CSS allows the document language to offer some means of specifying stylesheets according to media type. In the case of XML, the `media` pseudo-attribute of the `xml-stylesheet` provides processing instruction. This is useful because XML allows you to associate any number of external stylesheets with a document. The following XML document (eg_3_4.xml in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) is the example from [Use imports to specify media-specific styles](#), modified to show media selection in the processing instruction.

```

<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/css" href="eg_3_4_master.css"?>
<?xml-stylesheet type="text/css" href="eg_3_4_paged.css" media="print"?>
<paper>
  <prologue>
    <title>Faster than light travel</title>
    <subtitle>From fantasy to reality</subtitle>
    <author member='yes' e-mail="cemereuwa@nasa.gov">Chikezie Emereuwa</author>
    <author member='no' e-mail="okey.agu@navy.mil">Okechukwu Agu</author>
    <main-contact e-mail='cemereuwa@nasa.gov' />
    <abstract>All laws are meant to be broken, and according to our analysis,
    all the required antecedent technologies are in place for us to break
    the famous universal speed limit that derives from the Special Theory
    of Relativity.</abstract>
  </prologue>
  <section>
    <para>The work of Oguchi, Hu, Schneider et. al. already establishes
    the equivalent states of each elementary particle when traveling
    at superoptical velocities. The work of Baraka, Smith and Hossemi establish
    the effect of an omega hadron field in eliminating relativistic effects
    upon ordinary matter.</para>
  </section>
  <section>
    <para>We can construct an omega hadron field drive in a module that fits
    in the International Space Station high energy experiments wing, allowing us
    to demonstrate lt;abbreviation><short>FTL</short><long>faster
    than light</long></abbreviation> travel over distances of up

```

```
to 12,000 meters.</para>
<para>By September of 2020, we expect superoptical drives to burst
from science fiction into the world of fact.</para>
</section>
</paper>
```

Notice the pseudo-attribute `media="print"` in the second processing instruction. This instructs the user agent to load this stylesheet only if it is rendering for print medium. On the other hand, `eg_3_4_master.css` is always loaded regardless of medium. Note: It is called a *pseudo-attribute* because even though it has similar syntax to attributes in element tags, it is not technically recognized as an attribute by any XML processor (it is recognized as text, from which the user agent parses the pseudo-attribute). `href` and `type` are also pseudo-attributes.

`eg_3_4_master.css` and `eg_3_4_paged.css` are the same as their counterparts in [Use imports to specify media-specific styles](#) except that the former does not have an `@import` rule.

Alternate styles

Besides permitting readers to tailor style to presentation medium, you might want to provide them with alternate styles for a document. You might have a standard style, one for color-blind readers, one for readers who like highlighting by color, and one for readers who like highlighting by font effects. The `xml-stylesheet` processing instruction supports this through the `alternative` and `title` pseudo-attributes. Basically, three modes for selecting a stylesheet are based on the value of these attributes:

- A **persistent** stylesheet does not have a `title` pseudo-attribute, and it either has no `alternative` pseudo-attribute or one with the value `no`. A persistent stylesheet is always loaded in addition to any other stylesheets that might be loaded according to medium or reader preference.
- A **preferred** stylesheet has a `title` pseudo-attribute, and it either has no `alternative` pseudo-attribute or one with the value `no`. A preferred stylesheet is loaded by default, but the reader can opt not to load it by deselecting its title using whatever mechanism the user agent provides.
- An **alternate** stylesheet has a `title` pseudo-attribute and an `alternative` pseudo-attribute with the value `yes`. An alternate stylesheet is not loaded by default, but the reader can opt to load it by selecting its title using whatever mechanism the user agent provides.

As an example, see the following set of stylesheet processing instructions:

```
<!-- persistent -->
```

```
<?xml-stylesheet type="text/css" href="basics.css"?>
<!-- preferred -->
<?xml-stylesheet type="text/css" href="regular.css"
  title="Style for most readers"?>
<!-- alternative -->
<?xml-stylesheet type="text/css" href="compact.css"
  title="Compact style" alternate="yes"?>
```

In this example, the rules in `basics.css` always apply. By default the rules in `regular.css` apply, but the reader can turn them off. By default the rules in `compact.css` do not apply, but the reader can turn them on. Firefox allows users some measure of selection from stylesheets in the **Page Style** option in the **View** menu.

Section 4. Manage behaviors with XML and CSS

Extend CSS to support linking?

Unfortunately, the challenge of adding links to an XML document has become something of a disaster. This can be a make-or-break consideration for whether XML/CSS works for you, and yet just about every user agent seems to have tackled the problem in a different way. But since it is such an important matter, I shall cover the topic as best as I can.

I'll start with the most CSS-like approach. The Opera Web browser defines a set of proprietary CSS extensions for XML linking: `-set-link-source` and `-use-link-source`. Using these extension rules you can specify attributes or content that contains link targets. The following XML document (`eg_4_1.xml` in the file `x-xmlcss2-tutorial-files.zip`, see [Downloads](#)) contains a list of fashion designers with (bogus) links for each corporate home page.

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/css" href="eg_4_1.css"?>
<designers>
  <designer homepage="http://doria.co.uk">Doria Ltd.</designer>
  <designer
    homepage="http://samsara.biz">Samsara Saris</designer>
  <designer
    homepage="http://pcp.co.uk">Pikeman Camouflage, Plc.</designer>
  <designer homepage="http://mandalay.co.jp">Mandalay</designer>
</designers>
```

The following CSS (`eg_4_1.css` in the file `x-xmlcss2-tutorial-files.zip`, see [Downloads](#)) uses the Opera CSS extensions to render each designer element as a link to the home page.

```

* { display: inherit; }

designers { display: block; }

designer {
  margin-left: 3em;
  width: 20em;
  /* use the browser convention of blue text, underlined */
  color: blue;
  text-decoration: underline;
  /* Actually specify the link data */
  -set-link-source: attr(homepage);
  -use-link-source: current;
}

```

Besides the usual highlighting to indicate the link, the first interesting rule in this CSS is `-set-link-source`; it sets either the attribute from which the link URL is to be taken, or `content()`, to indicate that the element content has the link URL. The second property, `-use-link-source` is a bit of a head-scratcher, but suffice it to say that you will almost always use this property in a rule with a value of `current` to accompany `-use-link-source`.

This stylesheet will only have the desired effect in the Opera Web browser.

Use XLink for linking XML documents

Mozilla-based Web browsers such as Firefox support XLink, the W3C's generic framework for expressing links in XML documents. Unfortunately, XLink has been a controversial spec with spotty support in browsers. One of the browsers that does support XLink is Mozilla. The easiest way to render links from XML in Mozilla actually has nothing to do with CSS: You just use XLink in the source document.

The following XML document (`eg_4_2.xml` in the file `x-xmlcss2-tutorial-files.zip`, see [Downloads](#)) is a similar list of fashion designers, but this time using XLink. I touched up the document structure a bit so you can see the contrast between links and non-link text.

```

<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/css" href="eg_4_2.css"?>
<designers xmlns:xl="http://www.w3.org/1999/xlink">
  <blurb>
    <designer xl:type="simple"
              xl:href="http://doria.co.uk">
      >Doria Ltd.</designer> of London
    </blurb>
  <blurb>
    <designer xl:type="simple"
              xl:href="http://samsara.biz">
      >Samsara Saris</designer> of Mumbai
    </blurb>
  <blurb>
    <designer xl:type="simple"
              xl:href="http://pcp.co.uk">
      >Pikeman Camouflage, Plc.</designer> of London
    </blurb>

```

```
<blurb>
  <designer xl:type="simple"
            xl:href="http://mandalay.co.jp"
            >Mandalay</designer> of Tokyo
</blurb>
</designers>
```

The following CSS (eg_4_2.css in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) provides the basic style for document display, and has nothing to do with the linking.

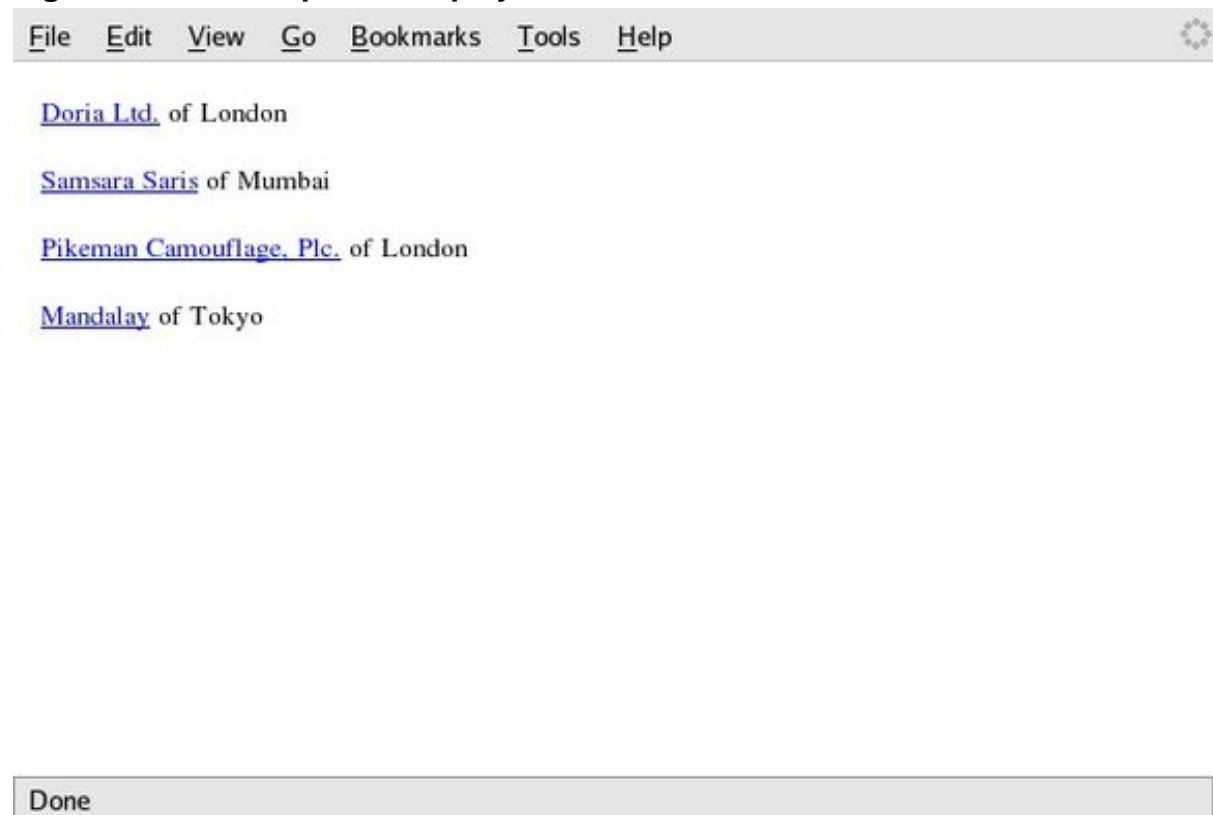
```
* { display: inherit; }
designers { display: block; }

blurb {
  margin: 1em;
  width: 20em;
}

designer {
  display: inline;
}
```

The image below shows the result. Firefox does display the links, using the correct URLs. It applies the popular convention of blue text, underlined automatically.

Figure 4. XLink output as displayed in Firefox



Style XLinks in Firefox

As you discovered in [Use XLink for linking XML documents](#), Firefox interprets links expressed using XLink readily enough, but it unimaginatively defaults the display to the conventional blue text, underlined. You can always modify the CSS if you prefer a different display. However, Firefox does make this a bit tricky. My first attempt at customizing the link style in the designer blurbs example was adding the following to the CSS:

```
*[xl:type] {
  text-decoration: none;
  color: green;
  border: thin violet solid;
}
```

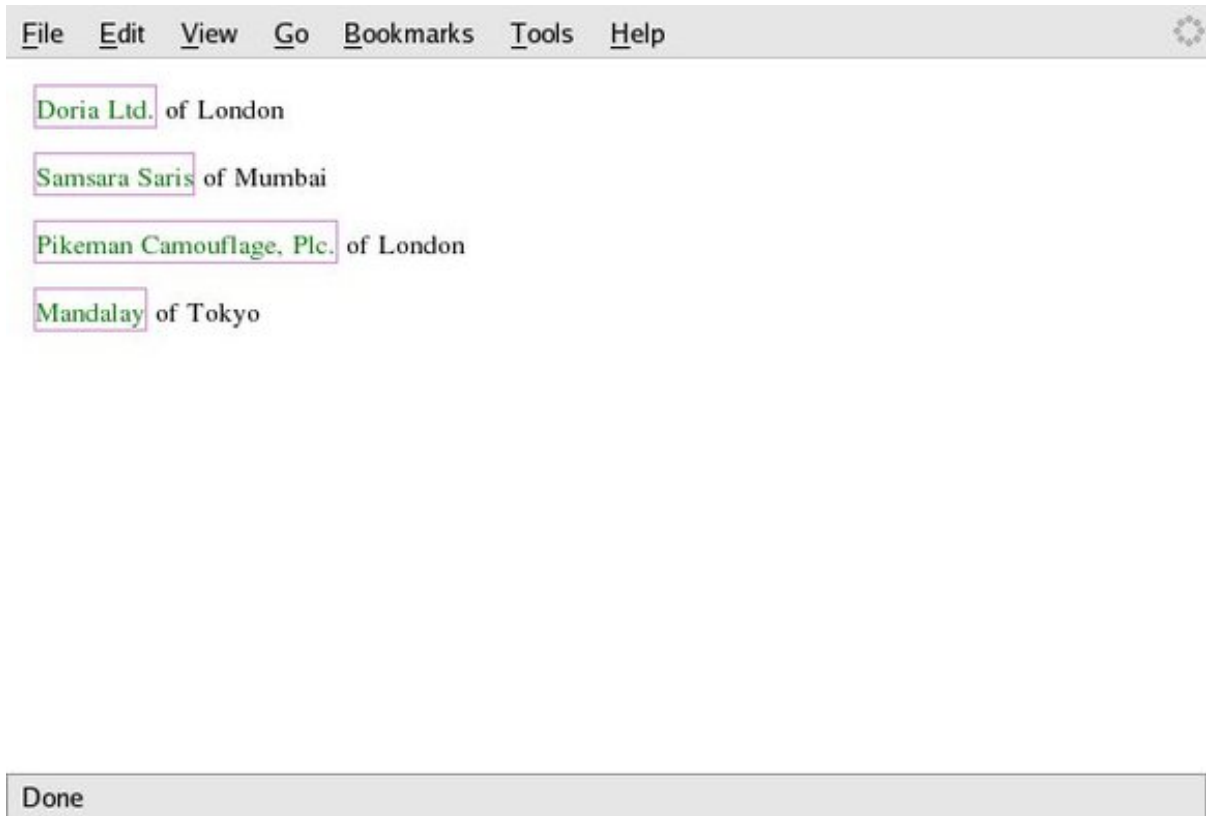
This should have selected all elements with an `xl:type` attribute (such as `designer`). The `text-decoration: none;` should have undone the underlining, and the rest of the rules should have imposed my own quirky style for the links.

I tried several variations on this, including selectors such as `xl:type="simple"`, with no luck. I only got it to work by stuffing the style for the links into the rule for `designer`. This limitation does reduce the modularity of how you can style links. The sample XML (eg_4_3.xml in the file `x-xmlcss2-tutorial-files.zip`, see [Downloads](#)) is the same as in [Use XLink for linking XML documents](#), but with a stylesheet processing instruction that loads the following CSS instead (eg_4_3.css in the file `x-xmlcss2-tutorial-files.zip`, see [Downloads](#)).

```
* { display: inherit; }
designers { display: block; }
blurb {
  margin: 1em;
  width: 20em;
}
designer {
  display: inline;
  text-decoration: none;
  color: green;
  border: thin violet solid;
}
```

The image below shows the resulting display.

Figure 5. Output of styled XLinks in Firefox



Use JavaScript and DOM for linking XML documents

A third approach to linking depends on dynamic scripting. It should work in most Web browsers, but is more complex than the last two approaches that I described. As a start, it will work in any browser that looks for elements within an XML document that are in the XHTML namespace and then use XHTML semantics for these elements alone. Such behavior is not required by any industry-wide specification. The trick, which I learned from an article by Scott Andrew LePera, is to use scripting to dynamically create link elements.

The following XML document (eg_4_4.xml in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) contains mostly the same list of fashion designers with links to home pages, but look at the bottom of the document for the key difference.

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet type="text/css" href="eg_4_4.css"?>
<designers>
  <blurb>
    <designer homepage="http://doria.co.uk">Doria Ltd.</designer>
      of London
  </blurb>
  <blurb>
    <designer homepage="http://samsara.biz">Samsara Saris</designer>    of Mumbai
  </blurb>
</designers>
```

```

    <designer homepage="http://pcp.co.uk">Pikeman Camouflage, Plc.</designer>
  of London
</blurb>
<blurb>
  <designer homepage="http://mandalay.co.jp">Mandalay</designer>
  of Tokyo
</blurb>
<xhtml:script xmlns:xhtml="http://www.w3.org/1999/xhtml"
  src="eg_4_4.js"
  type="text/javascript"/>
</designers>

```

The following JavaScript (eg_4_4.js in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) is the script that the XHTML `script` tag refers to. It basically removes the `designer` elements and replaces them with XHTML link elements (`<a href="..."`). Firefox runs the script as it's encountered in the source document, which is why you place it after all the elements that need to be turned into links. Because I do not require knowledge of JavaScript or DOM for this tutorial, I have heavily commented each line so that you can still follow the logic if you aren't familiar with these technologies. Lines that start with `/"` are comments.

```

//Save the XHTML namespace for when you need it
var xhtmlns = "http://www.w3.org/1999/xhtml";
//get all elements named "blurb" in the document
//The first "" indicates no namespace for the element you're seeking
var blurbs = document.getElementsByTagNameNS("", "blurb");
//Loop over each element you found
for (var i=0; i < blurbs.length; i++)
{
  //retrieve the blurb element from the collection
  var blurb = blurbs[i];
  //Get the designer element within the blurb
  //Assumes only one designer element per blurb
  var designer = blurb.getElementsByTagNameNS("", "designer").item(0);
  //In DOM the text in designer is actually a text node object child
  //of blurb. The link title is the value of this text node
  //Assumes the text node is normalized
  var link_title = designer.firstChild.nodeValue;
  //Link URL is the homepage attribute of designer, in no namespace
  var link_url = designer.getAttributeNS("", "homepage");
  //Create a new XHTML namespace link element
  var xhtml_link = document.createElementNS(xhtmlns, "a");
  //Create a new text node with the link title
  var new_text_node = document.createTextNode(link_title);
  //Set the href attribute to the link URL
  xhtml_link.setAttributeNS("", "href", link_url);
  //Attach the text node with the link title to the XHTML link
  xhtml_link.appendChild(new_text_node);
  //Replace the designer element with the new XHTML link element
  blurb.replaceChild(xhtml_link, designer);
}

```

The following CSS (eg_4_4.css in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) is still required for basic style.

```

* { display: inherit; }

designers { display: block; }

blurb {

```

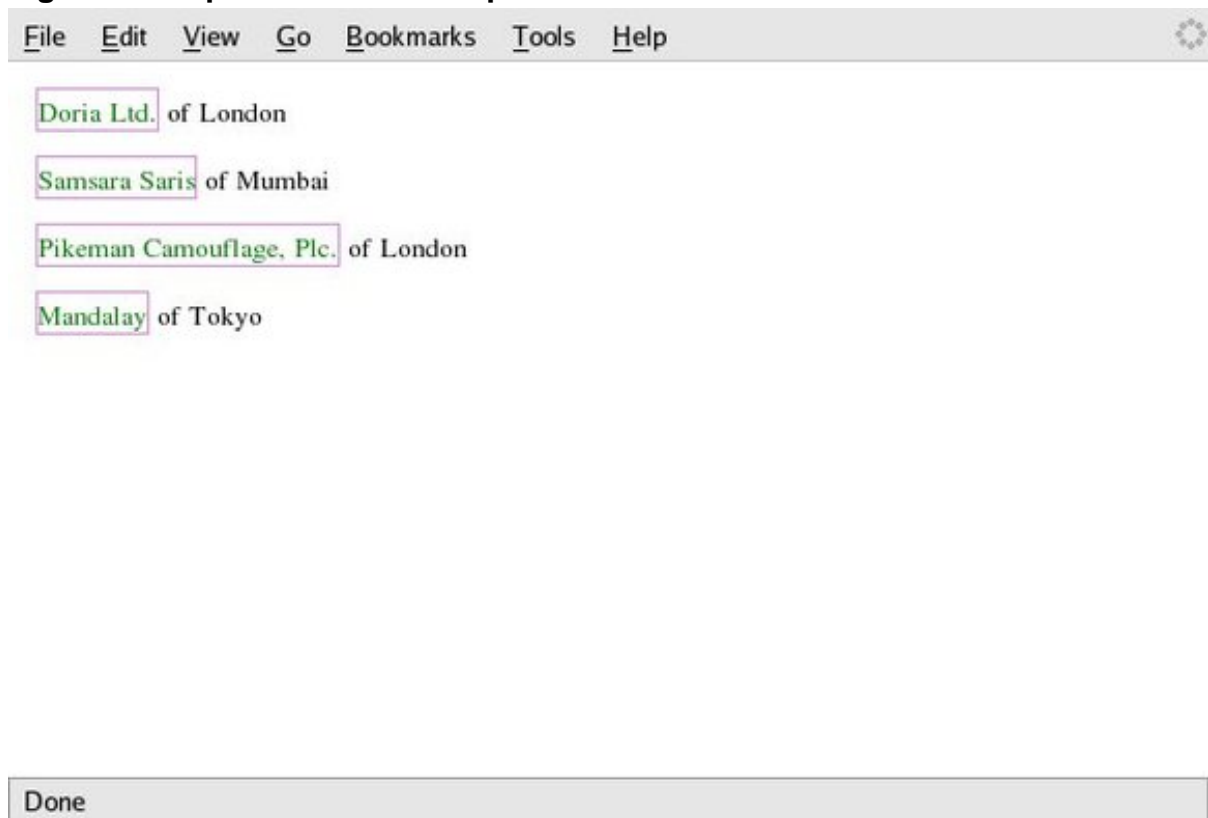
```
margin: 1em;
width: 20em;
}

a {
display: inline;
text-decoration: none;
color: green;
border: thin violet solid;
}
```

Notice that `designer` has no rule, since these elements are removed anyway. The CSS for styling the link is now in the `a` selector, which makes sense once the script has been applied. Firefox respects CSS rules even for dynamically created elements. The image below shows the resulting display. It is identical to the display from [Style XLinks in Firefox](#).

If you do know JavaScript and DOM, I have some advanced notes in the last two panels in this section. If you don't, please skip to the next section ([CSS in other XML vocabularies](#)).

Figure 6. Output when JavaScript and DOM link XML documents in Firefox



Style through JavaScript and DOM

This is a special, advanced section for those who are familiar with JavaScript and

DOM. If you aren't, please skip to the next section ([CSS in other XML vocabularies](#)).

In the example in [Use JavaScript and DOM for linking XML documents](#), I showed you how to apply style for generated XHTML links through a CSS stylesheet that was loaded by the source XML document. You can also apply style directly in JavaScript and DOM. The following JavaScript (eg_4_5.js in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) is an update of the script in the previous section, [CSS and media types](#). I added JavaScript commands to set the CSS style on the newly created XHTML link elements. See the comments for more details. I have removed the comments for JavaScript statements that appeared in [Use JavaScript and DOM for linking XML documents](#).

```
//For more explanatory comments on this code, see eg_4_4.js
var xhtmlns = "http://www.w3.org/1999/xhtml";
var blurbs = document.getElementsByTagNameNS("", "blurb");
for (var i=0; i < blurbs.length; i++)
{
    var blurb = blurbs[i];
    var designer = blurb.getElementsByTagNameNS("", "designer").item(0);
    var link_text = designer.firstChild.nodeValue;
    var link_text_node = designer.firstChild;
    var link_url = designer.getAttributeNS("", "homepage");
    var xhtml_link = document.createElementNS(xhtmlns, "a");
    xhtml_link.setAttributeNS("", "href", link_url);
    xhtml_link.appendChild(link_text_node);
    blurb.replaceChild(xhtml_link, designer);
    //ADDED SECTION
    //Set CSS style of xhtml_link
    //Setting display to "inline" is needed to override the
    //"*{ display: block}" rule from the CSS file
    xhtml_link.style.display = "inline";
    xhtml_link.style.color = "green";
    xhtml_link.style.border = "thin violet solid";
    //Notice that css property "text-decoration" can't be accessed as
    //easily as others because this is not a valid JavaScript identifier
    //We have to fall back to the setProperty() function, to which we
    //Pass in the property name, value, and priority ("important" is
    //often what you want for priority in cases of scripting)
    xhtml_link.style.setProperty("text-decoration", "none", "important");
}
}
```

The following CSS (eg_4_5.css in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) is still required for basic style, but it no longer tries to style the a element.

```
* { display: inherit; }

designers { display: block; }

blurb {
    margin: 1em;
    width: 20em;
}
```

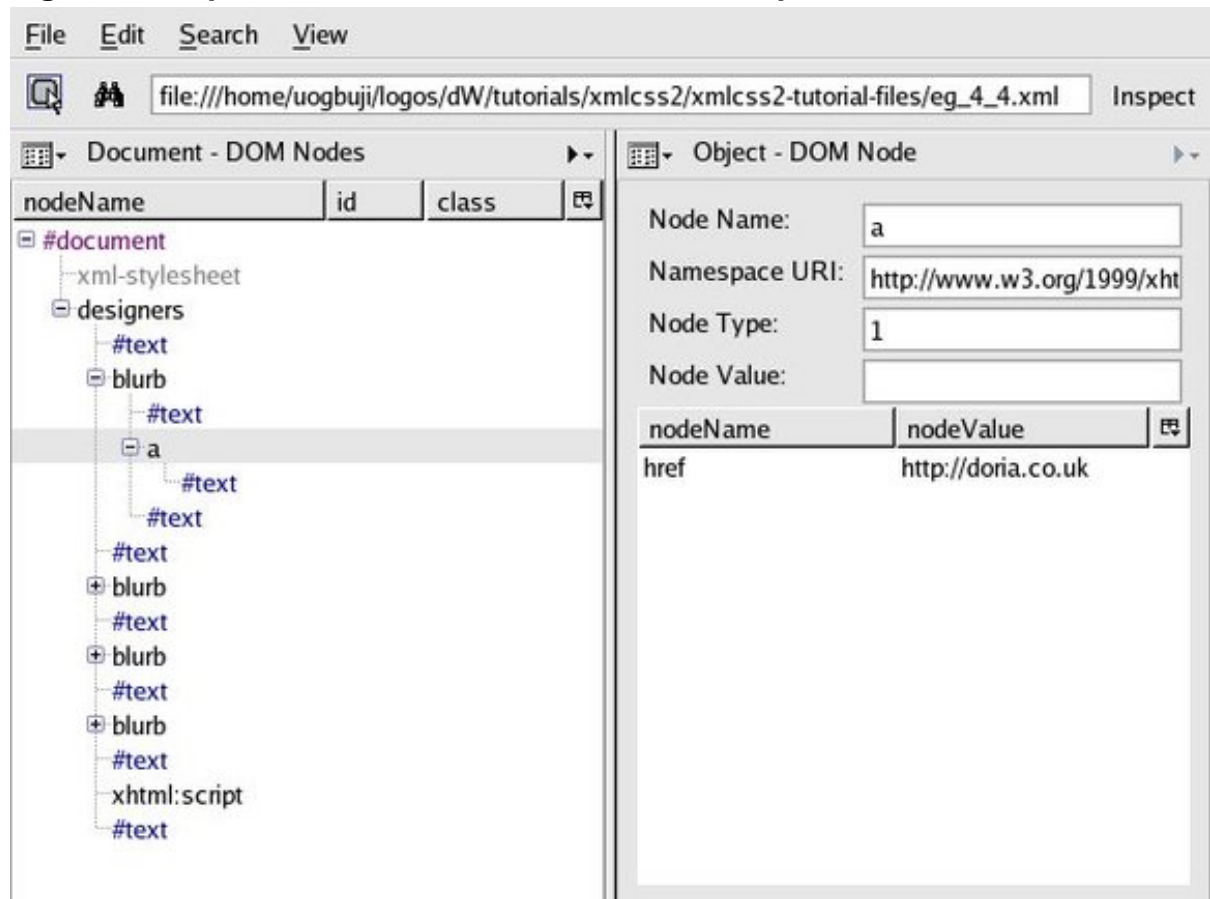
If you want to experiment, you'll find updated XML (eg_4_5.xml) in the file x-xmlcss2-tutorial-files.zip (see [Downloads](#)). The only difference is the reference to updated CSS and JavaScript files.

Fun with JavaScript and DOM

This is a special, advanced section for those who are familiar with JavaScript and DOM. If you aren't, please skip to the next section ([CSS in other XML vocabularies](#)).

Mozilla provides a couple of nice tools to help with development of such JavaScript and DOM trickery. One is the **DOM inspector** (in the **Tools** menu of Firefox 1.0). It provides a tree view of DOM nodes, with pretty much all the properties (from basic DOM, CSS, and elsewhere) that you might possibly care about. The following image is an example tree view corresponding to the browser output features in [Style through JavaScript and DOM](#). If you bring up the DOM Inspector and click on **Object -> DOM Node**, you will see other available views, including a comprehensive view of the CSS declarations in effect for the selected node.

Figure 7. Output in tree view created with JavaScript and DOM



The **JavaScript Console** (also in the **Tools** menu of Firefox 1.0) is important to have open while you're developing JavaScript code. It lists all errors and relevant information from the scripting session. Without it, many errors are invisible and very difficult to debug.

Section 5. CSS in other XML vocabularies

Express style wherever it needs to be expressed

\

Web browsers aren't the only sort of application that can present information to readers based on XML. Therefore CSS does not limit itself to Web browsers, or even Web-accessible user agents. The generic term "user agent" covers a wide variety of software, and with this in mind, many XML vocabularies provide for the use of CSS to express style. Some examples are:

- **SVG:** A very sophisticated vector graphics format in XML
- **XForms:** An XML language for user interface forms, similar to the forms elements in HTML and XHTML, but more powerful
- **Mozilla's XUL:** A user interface control language for the Mozilla Web browser

The first two are W3C Recommendations while XUL is proprietary to Mozilla. For this reason I focus on the first two, but I still think you may find XUL interesting because of the openness and flexibility of the Mozilla project.

Don't worry -- I don't require knowledge of any of these XML languages in this section; I provide examples that are simple enough for you to get the basic idea of the role of CSS.

In this section I also briefly compare CSS with XSL-FO (formatting objects), a popular XML vocabulary for specifying print-quality presentation.

CSS loaded from SVG

SVG is a remarkable language, not just for the amount of function and quality that the W3C was able to mold with XML (not always the most elegant syntax), but also for how well it has been adopted. One of the tricks that made SVG such a manageable spec was to off-load many stylistic concerns to CSS. This also helped CSS application developers to reuse existing CSS implementations, which facilitated adoption. That said, CSS support is not mandatory for SVG implementations, but most implementations do support CSS.

In CSS you can use the regular `xml-stylesheet` processing instruction to load a stylesheet, or you can embed CSS in a `style` element in the CSS namespace (if you do, I recommend wrapping the CSS in a CDATA section to avoid character escape problems). The following SVG (eg_5_2.svg in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) is based on an example in the CSS spec:

```
<?xml version='1.0' encoding='utf-8'?>
<?xml-stylesheet href="eg_5_2.css" type="text/css"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="5cm" viewBox="0 0 1000 500"
xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect x="200" y="100" width="600" height="300"/>
</svg>
```

The following CSS (eg_5_2.css in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) is loaded by the XML.

```
rect {
  fill: red;
  stroke: blue;
  stroke-width: 3
}
```

The output is a red rectangle with a blue border. The geometry of the rectangle is given in the SVG, by the `rect` element. The style of the rectangle, however, is given in the CSS. You'll notice the SVG document type declaration. As for the main element, `svg`, its main purpose is to establish the dimensions of the image's viewable area, and to declare the default namespace `http://www.w3.org/2000/svg`. Of course, you can use any prefix for this namespace, but if you do, don't forget to account for it by changing the element name selector in the CSS.

This approach provides the usual benefits of separating the basic graphics information from style.

CSS embedded in SVG

You can also embed the entire stylesheet within the CSS document, forfeiting some of the content/presentation separation benefits. The following SVG (eg_5_3_1.svg in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) is based on an example in the CSS spec:

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
"http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="5cm" viewBox="0 0 1000 500"
xmlns="http://www.w3.org/2000/svg" version="1.1">
  <defs>
```

```

<style type="text/css"><![CDATA[
  rect {
    fill: red;
    stroke: blue;
    stroke-width: 3
  }
]]></style>
</defs>
<rect x="200" y="100" width="600" height="300"/>
</svg>

```

The `style` element within `defs` expresses style for the entire document element.

A third option is to place the CSS rules inline in the CSS elements, similar to `style` attributes on arbitrary XHTML elements. The following SVG (eg_5_3_2.svg in the file x-xmlcss2-tutorial-files.zip, see [Downloads](#)) is also based on an example in the CSS spec:

```

<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.1//EN"
  "http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd">
<svg width="10cm" height="5cm" viewBox="0 0 1000 500"
  xmlns="http://www.w3.org/2000/svg" version="1.1">
  <rect x="200" y="100" width="600" height="300"
    style="fill:red; stroke:blue; stroke-width:3"/>
</svg>

```

The `style` attribute on the `rect` element now has the CSS rules for that element. You can also specify a class for an SVG element using a `class` attribute with similar semantics to XHTML's.

CSS with XForms

XML is intended to take you from the chaotic world of information management represented by HTML technologies to a new era of structured information management. Similarly, XForms is designed to take you from a tangled and confused world of Web forms to a next generation of forms that have:

- Flexible semantics that minimize the need for scripting
- A very clean data model
- Good separation of content and presentation

Once again, CSS is the engine behind the latter drive.

XForms works closely with one of the modules of CSS3, *CSS3 Basic User Interface Module* (W3C Candidate Recommendation, May 2004), which defines a whole raft of special pseudo-classes, pseudo-elements, and so on. These are special, virtual classes and elements that appear to be present depending on the properties or state

of various forms elements. The CSS processor then applies any specified rules as normal. The idea is to make assertions such as "If the user has entered invalid data in any field, highlight it with a red color" or "...play back an audio message warning of the error". The following CSS snippet covers the first example:

```
*:invalid { background-color: red; }
```

CSS versus XSL-FO

You might have heard of XSL Formatting Objects (XSL-FO), a presentation language defined in XML that's especially suited for print formatting. XSL-FO is an XML format that any user agent can use to render content to precise specifications given by the developer. It allows for page-perfect positioning and other formatting tasks. In this role, it has some significant overlap with the print media module for CSS. In fact, there have been several debates about whether people should use XSL-FO or CSS for formatting XML documents for print.

I have used both CSS and XSL-FO extensively to format XML and I don't really pick a side in this debate, but I thought it is worth giving a few practical notes on the matter, since it may affect your decision-making in how best to use CSS with XML.

In my opinion, CSS has the advantage in simplicity. First of all, XSL-FO is very verbose, and very complex. Some of this complexity comes from its significant formatting power, but to create XSL-FO, you need a very keen understanding of technical formatting arcana. The XSL-FO option also generally involves running the original XML through an XSLT processor to generate XSL-FO, which is then run through an XSL-FO processor to generate PDF, Postscript, TeX DVI, or another display format. This can be a complex chain of operations to get right.

The following is an example of the differences between the two formats alone. It's taken from the CSS versus XSL-FO wars, but I chose an example that I do think fairly illustrates the typical difference between the two. First the CSS example:

```
div.header {  
  margin-top: 8pt;  
  margin-bottom: 8pt;  
  font-size: 75%  
}
```

And the following is XSL-FO with similar semantics:

```
<fo:block space-before="8pt"  
          space-after="8pt"  
          font-size="75%">  
<xsl:apply-templates/>
```

```
</fo:block>
```

XSL-FO's advantage over CSS is mostly a matter of tools support. A wide range of tools work with XSL-FO, and they can create professional documents in many print-ready formats. CSS has tremendous tool support, but mostly for screen media. Support for the print media capabilities of CSS has in practice been very slow in coming.

Of course, in the end ecumenism has a way of winning out, and there are already several efforts to bridge the CSS and XSL-FO divide. A good example is the **CSSToXSLFO project**, which is developing a utility to convert CSS to XSL-FO. Using such a tool, you can take advantage of the simplicity of CSS *and* the tools support of XSL-FO.

Section 6. Wrap up

Summary

In this tutorial, you learned many techniques for using CSS to style XML. These included:

- More sophisticated element selection based on attributes, including selection by language code attributes
- Selection based on DTD IDs, or using the forthcoming `xml:id` specification
- Controlling CSS modules based on the media used for document rendering
- Selective loading of CSS from the XML document based on media and user preference
- Techniques for rendering links from XML documents, including CSS extensions, XLink, and JavaScript/DOM manipulation
- *Advanced users* -- Using JavaScript and DOM to manipulate CSS in elements from XML source documents, including useful tools and tricks to help with this
- Use of CSS in SVG and XForms

- How to choose between CSS and XSL-FO for rendering to print media

This wide field of topics merely scratches the surface of what you can achieve with XML and CSS using today's technology, and technology soon to come. This is just the beginning, and I strongly encourage you to push the boundaries through lots of practice and experimentation, using the example code I presented as a base.

Downloads

Description	Name	Size	Download method
Sample code	x-xmlcss2-tutorial-files.zip	14KB	HTTP

[Information about download methods](#)

Resources

Learn

- If you're unfamiliar with XML, start with the many helpful resources in the developerWorks "[New to XML](#)" page.
- Learn the basics of CSS and CSS with XML. See the earlier tutorial "[Use Cascading Stylesheets to display XML](#)" (developerWorks, November 2004). That tutorial's extensive resources panel is particularly useful.
- Browse through Simon St.Laurent's series exploring practical display of XML in major Web browsers. The articles are dated (some code details are obsolete), but still excellent:
 - "[On Display: XML Web Pages with Mozilla](#)"
 - "[On Display: XML Web Pages with Opera 4.0](#)"
 - "[On Display: XML Web Pages with Internet Explorer 5.x](#)"

Simon also has a very nice [Browser XML Display Support Chart](#), although it covers older browser versions.

- 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.
- Read "[Printing XML: Why CSS Is Better than XSL](#)" by Hakon Wium Lie and Michael Day for a biased but informative look at how CSS can be used to style XML documents for print, by comparison with XSL-FO. XSL expert Norm Walsh wrote a rebuttal "[Comparing CSS and XSL: A Reply...](#)" See "[CSS3 Paged Media Module](#)" (W3C Candidate Recommendation, February 2004) for all the details on CSS for print media.
- 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."
- Bookmark and refer to the formal CSS2 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 CSS3 working drafts. Read "[CSS 3 Selectors](#)" by Russell Dyer for a good overview of the selectors (though CSS3 is more than just new selectors). But before CSS3 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.

- Browse the short and sweet [xml:id](#) working draft from the W3C.
- Read "[Printing XML: Why CSS Is Better than XSL](#)" by Hakon Wium Lie and Michael Day for a biased but informative look at how to use CSS to style XML documents for print. See "[CSS3 Paged Media Module](#)" (W3C Candidate Recommendation, February 2004) for all the details on CSS for print media.
- If you're already familiar with SVG, pore through the details of where CSS fits in with SVG in section "[6 Styling](#)" of the latest SVG recommendation.
- See "[SVG and Typography](#)" by Fabio Arciniegas for a very interesting discussion of how CSS in SVG can be used to create effects with text.
- See "[CSS3 Basic User Interface Module](#)" (W3C Candidate Recommendation, May 2004) for the details on special CSS features that you can use with XForms and related specifications. Also of interest is "[Document Object Model \(DOM\) Level 2 Style Specification](#)" (W3C Recommendation, November 2000), which details how to process CSS using DOM (for example, through JavaScript in a browser).
- If you want to learn more about XForms, check out [XForms Essentials](#) by Micah Dubinko (O'Reilly and Associates). It is available online, but is well worth the purchase.
- Learn more about XSL-FO with IBM's own Doug Tidwell.
 - His developerWorks tutorial "[XSL Formatting Objects \(XSL-FO\) basics](#)" (February 2003) shows you the fundamentals of this powerful, flexible XML vocabulary for formatting data.
 - His follow-up tutorial, "[XSL-FO advanced techniques](#)" (February 2003), shows advanced techniques for working with XSL-FO for formatting data, such as handling lists and tables, and creating complex documents with multiple layouts.
 - Finally, his "[HTML to Formatting Objects \(FO\) conversion guide](#)" (February 2003) shows by example how to use XSLT templates to convert 45 commonly used HTML elements to formatting objects for easy transformation to PDF using XSLT.
- 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.
- If you're interested in learning more on XUL, Mozilla's rich user interface language (represented as an XML vocabulary), start with [this introduction](#).

- Check out [CSSToXSLFO](#), "a utility which can convert an XML document, together with a CSS2 stylesheet, into an XSLFO document, which can then be converted into PDF, PostScript, etc. with an XSLFO-processor."
- 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 resources on the [developerWorks XML zone](#).
- Finally, find out how you can become an [IBM Certified Developer in XML and related technologies](#).

Get products and technologies

- The author tested the CSS in this tutorial with [Firefox](#), a popular and free Web browser available on Windows, Mac OS X, Linux, and other platforms. Firefox is based on Mozilla's rendering engine, which historically is a very CSS-compliant browser. Also consider [Opera](#), a commercial (though inexpensive) browser implementing even more of CSS2 than Firefox. If you are an Opera user, look over the product document "[Web Specifications Supported in Opera 6](#)," which includes a description of Opera's CSS extensions for linking and other uses.

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 contact Mr. Ogbuji at uche.ogbuji@fourthought.com.