Using XSLT as a stylesheet for HTML

Jürgen M. Regel

March 13, 2012

As the name suggests, Extensible Stylesheet Language Transformations (XSLT) can be used as a stylesheet. Like Cascading Style Sheets (CSS), XSLT helps to separate style from content. You can simplify and enrich your XHTML documents with XSLT, which relieves the XHTML document of navigation. In this article, learn how to use XSLT as stylesheets, which can be executed on the server or in every modern Internet browser. Several examples show how to start developing your own XSLT stylesheets.

Introduction

When you hear the word stylesheet, you probably think of CSS stylesheets. XSLT stylesheets are typically used for XML transformations, such as mapping data between web services. Because XSLT is so well suited for this purpose, the `<xsl:transform>` alias of the top-level element `<stylesheet>` was created—though it's hardly ever used. The input structure of such an XSLT transformation is quite different from the output. Primarily, the namespaces are different.

Conversely, the input structure of an XSLT stylesheet is similar to the output structure, but simpler. Some tags are enriched, but most are simply copied without change to the output. Input and output namespaces are the same (HTML). The input document may also contain directives for the stylesheet (such as creating a footnote), which belong to another namespace and do not arrive in the output.

Frequently used abbreviations

- CSS: Cascading Style Sheets
- XHTML: Extensible Hypertext Markup Language
- XPath: XML Path Language
- XSLT: Extensible Stylesheet Language Transformations

In this article, learn how to use XSLT stylesheets to enrich your XHTML documents. Examples in the article show how to use directives, how to reference parts of other source documents, and how to navigate with links in a master document. Also, explore the difference between interpretation and compilation of your pages.

Limitations of CSS stylesheets

The XSLT stylesheet does not prevent you from using other techniques, such as JavaScript or CSS. CSS is suitable for fonts, emphasis, colors, margins, and so forth. It is not for combining information from different places, such as footnotes, modules, or generating a table of contents. This is where XSLT comes into play—it is completing, not replacing, CSS.
Examples of what XSLT can do

In practice, you might collect your XSLT code in a single file. For simplicity, each example in this article is in a separate XSLT file, with the exception of some code that is always required. Listing 1 shows the required code.

Listing 1. Required code (in samples/common.xml)

```xml
<s:stylesheet
xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://www.w3.org/1999/xhtml"
xmlns:a="http://sourceforge.net/projects/arbalo/"
xmlns:s="http://www.w3.org/1999/XSL/Transform"
exclude-result-prefixes="a h"
version="1.0"
>
  <s:template match="h:head">
    <s:copy>
      <s:apply-templates select="@*|node()"/>
      <meta
        http-equiv="content-type"
        content="text/html;charset=UTF-8" />
      <link
        href="common.css"
        rel="stylesheet"
        type="text/css" />
    </s:copy>
  </s:template>
  <s:template match="*">
    <s:copy>
      <s:copy-of select="@*"/>
      <s:apply-templates/>
    </s:copy>
  </s:template>
</s:stylesheet>
```

The namespace for XHTML is defined twice: default and h:. The default namespace is used for writing output XHTML tags where a namespace prefix should be avoided. h: is used in XPath expressions.

This article uses Version 1.0 of XSLT. At this time, most browsers cannot interpret XSLT 2.0. It might be a usable option, however, if your XSLT runs on the server. XSLT 2.0 also provides:

- XPATH 2.0 (if...then...else, and many built-in functions)
- Built-in and user written XPATH functions
- Grouping

In Listing 1:

- `<s:template match="head"/>` enriches the head section of the source document by adding a link to a CSS stylesheet. Even if UTF-8 is the default encoding in XML, some browsers need the content type to render it.
- `<s:template match="*"/>` is the default deep copy. In principle, everything is copied into the target document. If this template were missing, only the text content of the tags would be copied to the target document. Processing instruction nodes are not copied.
All other examples in this article are separate files that import the common.xsl.

**Enrichment**

With enrichment, a feature is added that is not explicitly requested in the source document. An example is the link to the CSS stylesheet in **Listing 1**. Try another example by adding a small arrow (^ v) to each internal link that indicates whether the target precedes or follows it. **Listing 2** shows the stylesheet.

**Listing 2. Stylesheet (in samples/linkUpDown.xsl)**

```xml
<s:stylesheet
   xmlns="http://www.w3.org/1999/xhtml"
   xmlns:h="http://www.w3.org/1999/xhtml"
   xmlns:s="http://www.w3.org/1999/XSL/Transform"
   version="1.0">
   <s:import href="common.xsl"/>
   <s:template match="h:a[starts-with(@href,'#')]">
     <s:copy>
       <s:copy-of select="@*"/>
       <s:variable name="name" select="substring-after(@href,'#')"/>
       <s:choose>
         <s:when test="preceding::h:a[@name=$name]">
           <s:text>^</s:text>
         </s:when>
         <s:when test="following::h:a[@name=$name]">
           <s:text>v</s:text>
         </s:when>
       </s:choose>
       <s:apply-templates/>
     </s:copy>
   </s:template>
</s:stylesheet>
```

At first, the common stylesheet in **Listing 2** is imported. The template matches internal links (starting with '#'). If the anchor that the link points to precedes the link, the link is enriched by an upward arrow (or by a downward arrow if the opposite is true).

`s:copy-of` and `s:apply-templates` ensure that nothing falls by the wayside.

**Listing 3 shows an example document, with internal links, that is enriched by the stylesheet in Listing 2.**

**Listing 3. Source document (in samples/linkUpDown.xml)**

```xml
<?xml-stylesheet href="linkUpDown.xsl" type="text/xsl"?>
<html xmlns="http://www.w3.org/1999/xhtml">
<head/>
<body>
  <a name="a"/>
  <p>This link goes <a href="vb">downward.</a></p>
  <br/>
  <p>Reduce the size of the window to verify the link really works.</p>
  <br/>
  <a name="b"/>
  <p>This link goes <a href="^a">upward.</a></p>
</body>
</html>
```
The target document looks the same, except for the entries in Listing 4.

Listing 4. Target document (in samples/linkUpDown.html)

```html
... <a href="#b">v downwards.</a> ...
... <a href="#a">^ upwards.</a> ...
```

Directives

You can intersperse the source document with directives that tell the stylesheet what to do. They belong to another namespace (prefix `a:` in the example) and are not copied to the target document.

In Listing 5, a footnote is created by the directive tag `a:ref` anywhere in the source document.

Listing 5. Stylesheet (in samples/footnote.xsl)

```xml
<s:stylesheet
 xmlns="http://www.w3.org/1999/xhtml"
 xmlns:a="http://sourceforge.net/projects/arbalo/
 xmlns:h="http://www.w3.org/1999/xhtml"
 xmlns:s="http://www.w3.org/1999/XSL/Transform"
 version="1.0"
>
 <s:import href="common.xsl"/>
 <s:template match="h:body">
   <s:copy>
     <s:apply-templates select="@*|node()"
     <!-- put the footnotes at the end
     if there is no a:references directive -->
     <s:if test="not(descendant::a:references)"
     <s:call-template name="references"/>
   </s:if>
 </s:copy>
 </s:template>
 <s:template match="a:ref">
   <s:variable
     name="number"
     select="count(preceding::a:ref) + 1"/>
   <a name="ref-{$number}"/>
   <a class="footnote" href="#reference-{$number}"
     <s:value-of select="concat('v ',{$number})"/>
   </a>
 </s:template>
 <!-- if a:reference is missing, assume it at the end of the body -->
 <s:template match="a:references" name="references">
   <hr/>
   <s:for-each select="/a:ref">
     <s:variable name="number" select="count(preceding::a:ref) + 1"/>
     <p>
       <a name="reference-{$number}"/>
       <a class="footnote" href="#ref-{$number}"
         <s:value-of select="concat(' ^',{$number})"/>
       </a>
       <s:apply-templates/>
     </p>
   </s:for-each>
 </s:template>
</s:stylesheet>
```
With the `a:references` directive in the source document, the template named `references` allocates the footnotes where the template matches the directive. If such a directive is missing, the first template matching `body` allocates the footnotes at the end of the `body` by calling the same template named `references`. In both cases, the contents of the footnotes are listed and upward links are generated and indicated by upward arrows.

The second template, which matches `a:ref`, creates the link to the footnote with a down arrow. The footnote is numbered. Its content is ignored here.

The `class="footnote"` attribute is resolved after the XSLT transformation by a CSS stylesheet, which is linked in the XSLT stylesheet `common.xsl`.

The source document in Listing 6 uses the `a:ref` directive to create footnotes.

**Listing 6. Source document (in samples/footnote.xml)**

```xml
<?xml-stylesheet href="footnote.xsl" type="text/xsl"?>
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:a="http://sourceforge.net/projects/arbalo/"
><head/>
<body>
<p>This example looks a little scientific
   <a:ref>
      From Latin <em>scientia</em>
   </a:ref>
   and academic
   <a:ref>From Greek akademia</a:ref>.
</p>
<p>Do you know why?
   <a:ref>
      It uses <em>footnotes</em>.
   </a:ref>
</p>
<p>Reduce size of window to verify links are generated.</p>
</body>
</html>
```

The target document contains the list of footnotes at the bottom, as in Listing 7.

**Listing 7. Target document (in samples/footnote.html)**

```html
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:h="http://www.w3.org/1999/xhtml"
xmlns:a="http://sourceforge.net/projects/arbalo/"
><head><link type="text/css" rel="stylesheet" href="common.css"/></head>
<body>
<p>This example looks a little scientific
   <a name="ref-1"/> <a href="#reference-1" class="footnote">v 1</a></p>
</body>
```

```xml
```
and academic.
<a name="ref-2"/><a href="#reference-2" class="footnote">v 2</a>&lt;/a>
</p>
&lt;p&gt;Do you know why?
<a name="ref-3"/><a href="#reference-3" class="footnote">v 3</a>&lt;/a&gt;
</p>
&lt;p&gt;Reduce size of window to verify links are generated.&lt;/p&gt;
<br/>
&lt;hr/&gt;
&lt;p&gt;&lt;a name="reference-1"&gt;&lt;a href="#ref-1" class="footnote"> ^1</a&gt; From Latin &lt;em&gt;scientia&lt;/em&gt;&lt;/a&gt;
</p>
&lt;p&gt;&lt;a name="reference-2"/&gt;
 &lt;a href="#ref-2" class="footnote"> ^2</a&gt;From Greek akademia&lt;/a&gt;
&lt;p&gt;&lt;a name="reference-3"/&gt;&lt;a href="#ref-3" class="footnote"> ^3</a&gt;&lt;/a&gt; It uses &lt;em&gt;footnotes&lt;/em&gt;.&lt;/p&gt;
&lt;/body&gt;
&lt;/html&gt;

Crossing the border of the source document

Parts of other source documents can be referenced, too. The a:include directive includes an element that may belong to another source document and transforms it, as in Listing 8.

**Listing 8. Stylesheet (in samples/include.xsl)**

```xml
<s:stylesheet
 xmlns="http://www.w3.org/1999/xhtml"
 xmlns:a="http://sourceforge.net/projects/arbalo/
 xmlns:s="http://www.w3.org/1999/XSL/Transform"
 version="1.0"
 >
 <s:import href="common.xsl"/>
 <s:template match="a:include">
  <s:choose>
   <s:when test="0!=string-length(@src)">
    <s:apply-templates
      select="document(@src)//*[id=current()]/@refid]"/>
   </s:when>
   <s:when test="not(@src) and //a:default[1]/@src">
    <s:apply-templates
      select="document((//a:default[1]/@src)//*[id=current()]/@refid]"/>
   </s:when>
   <s:when test="0=string-length(@src) or not((//a:default[1]/@src))">
    <s:apply-templates
      select="//*[id=current()]/@refid]"/>
   </s:when>
  </s:choose>
 </s:template>
</s:stylesheet>
```

An a:include directive in the source document refers to the id of a source element. The document that the element is included from may be named in an src attribute. If the attribute is missing, the src attribute of the a:default directive is used. If there is no src attribute anywhere, the same source document is used. Therefore, the id is referenced by refid to avoid infinite recursion.

The imported element may be a complex type and is transformed after inclusion (apply-templates). Listing 9, Listing 10, and Listing 11 show examples.
Listing 9. Source document (in samples/include.xml)

```xml
<?xml-stylesheet href="include.xsl" type="text/xsl"?>
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:a="http://sourceforge.net/projects/arbal0/">
<head>
  <a:default src="includedY.xml"/>
</head>
<body>
  <p>The following text is included:</p>
  <a:include refid="x" src="includedX.xml"/>
  <a:include refid="y1"/>
  <p id="i">double</p>
  <a:include refid="y2"/>
  <a:include refid="i" src=""/>
</body>
</html>
```

Listing 10. Part of source document (in samples/includeY.xml)

```xml
<h2 id="y2">I'm the <em>included</em> h2</h2>
<h1 id="y1">I'm the <em>included</em> h1</h1>
```

Listing 11. Target document (in samples/include.html)

```html
<body>
  <p>The following text is included:</p>
  <p id="x">I'm the <em>included</em> paragraph.</p>
  <h1 id="y1">I'm the <em>included</em> h1</h1>
  <p id="i">double</p>
  <h2 id="y2">I'm the <em>included</em> h2</h2>
  <p id="i">double</p>
</body>
```

Master document and navigation

Suppose you have a presentation consisting of several pages, with a master document that contains the page headings and the links to them. You can generate the complete navigation—from each page to any other, and to the preceding and following page. Those details are beyond the scope of this article, but Related topics has links to HTML presentations using a master document. You get the compiled version by replacing .xml with .html. Let the browser show you the tidy source of the .xml. You will be surprised by how much is generated.

Interpretation versus compilation

Interpretation means that the page is XML (file extension .xml, content type text/xml or application/xml) and the XSLT stylesheet to which a processing instruction refers is executed in the browser.

Compilation means that the browser sees HTML (file extension .html, content type text/html) that was transformed from XML in your development environment, or on the server, just before the page was requested. Xalan and Saxon are well known XSLT processors.

Interpretation is the future. All modern browsers support XSLT, and there are some advantages:
• When testing, you get the result immediately. Simply press F5 in each browser you test to reflect the changes of the source page, CSS, and XSLT stylesheets.
• The amount of information to be transferred to the client is reduced.
• The client sees a clean and tidy web page because the enrichments have not been generated yet.

But also consider the disadvantages:

• There may be old browsers that do not support XSLT. If you publish your page for a controlled environment (intranet), this won't be an issue.
• Some modern browsers forbid an XSLT stylesheet from referring to another stylesheet in another directory.
• Combining XSLT with other features, such as SVG or iframe, might lead to problems in some browsers.
• Because most browsers do not support XSLT 2.0 or the upcoming 3.0, you cannot use the new features. No XPath 2.0 if ( ) then … else and no user written XPath functions.

Whether compiled or interpreted, in both cases other transformations (CSS, JavaScript) of the page are performed after the XSLT transformation.

**Conclusion**

In this article, you learned how to use XSLT stylesheets to enrich your XHTML documents. You can use the examples in this article as a launching point to build your own XSLT stylesheets.
Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Article, and samples, as XML.</td>
<td>xsl-stylesheets.zip</td>
<td>29KB</td>
</tr>
</tbody>
</table>

Note

1. The download file includes:
   - An example of a historical presentation using XSLT stylesheets (in German, but links to translations into any other language are generated by the XSLT framework): http://zocher-regel.gmxhome.de/ArbaloSchlacht/11Drusus.xml
   - The same file, but compiled to XHTML: http://zocher-regel.gmxhome.de/ArbaloSchlacht/11Drusus.html
   - A description of the XSLT framework used for this article, using the framework it describes: http://home.arcor.de/juergen.regel/Arbalo/index.xml
   - W3C XML Path Language (XPath): http://www.w3.org/TR/xpath/
   - W3C XSL Transformations (XSLT): http://www.w3.org/TR/xslt
Related topics

- **developerWorks Web development zone**: Find articles covering various web-based solutions. See the [Web development technical library](https://ibm.com/developerWorks) for a wide range of technical articles and tips, tutorials, standards, and IBM Redbooks.
- Build your skills in XML fundamentals and advanced technologies, including XML schema, XLST, and XQuery query language in the IBM Training course Introduction to XML and Related Technologies. Available in [classroom](https://www.ibm.com/training) and [online](https://www.ibm.com/training) formats.
- **IBM product evaluation versions**: Get your hands on application development tools and middleware products.

© Copyright IBM Corporation 2012
[Trademarks](https://www.ibm.com/developerworks/ibm/trademarks/)