

Hypermedia and the Web

XSLT and XPath

XSLT – Extensible Stylesheet Language for Transformations

Compare/contrast with CSS:

CSS is used to change display characteristics of primarily HTML documents. But, CSS has limitations:

- CSS can't change the order of elements in a document
- CSS cannot perform computations (summing together multiple element values, for example)
- CSS cannot combine multiple documents

XSLT is a language for transforming XML documents into other forms.

Can transform XML into: HTML, XML, PDF, SVG, VRML, Java code, text files, etc.

Helps answer the question, how do I display XML? Convert to HTML, then view in a browser.

Related standard: XSL-FO – XML Stylesheet Language, Formatting Objects – a language for converting XML into page descriptions, such as PDF documents. Can be viewed as similar to Latex.

A few high-level observations:

XSLT stylesheets are XML documents – use XML to represent the XSLT language itself

XSLT uses rule-based pattern matching. XPath is used in the creation of the patterns. Typical form: if you see part of the document that looks like *pattern*, perform *action*.

Ways you might want to use XSLT:

- Convert XML into multiple display representations (for Web browser, handheld, cell phone, print)
- Data conversion for data interchange among different formats
- Have Web site content available in logical form, and have the actual Web site generated from the XML. More easily supports site redesign.
- Have documents in a relatively neutral format that is richer than text.

Hello World example (From XSLT, Doug Tidwell, O'Reilly, 2001)

Input XML:

```
<?xml version="1.0"?>
<greeting>
  Hello, World!
</greeting>
```

XSLT stylesheet:

```
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
  <xsl:output method="html"/>

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

  <xsl:template match="greeting">
    <html>
      <body>
        <h1>
          <xsl:value-of select="."/>
        </h1>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

Output:

```
<html>
<body>
<h1>
  Hello, World!
</h1>
</body>
</html>
```

Steps performed during the example:

- XSLT stylesheet is read and parsed
- XML source document is read and parsed into a tree structure
- XSLT processor is at the root of the source document (context set to root)
- Since there is a template that matches the document root, evaluate that template.
- The template's instruction is to evaluate any templates that apply to greeting elements
- There is one greeting element at the document root, so evaluate the second template instruction
- Elements that are not in the XSL namespace are passed through to the output (these are the HTML tags html, body, h1)
- The `xsl:value-of` puts into the output the value of the node matching the XPath expression, which in this case is the current context (i.e., the greeting node)
- Remaining HTML tags pass through to output
- No other templates match, are done.

Note:

`xsl:stylesheet` element provides `xsl` namespace declaration, and version of XSLT. XSLT processors complain if these are missing.

`xsl:output` describes which output format is being used (in this case HTML, but many other are possible – “xml” is a common one)

Built-in template rule:

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

Ensures that XML documents will be processed, even if your stylesheet only includes more specific matching rules, and omits a matching rule for the root element.

Some text below comes from XSLT, by Doug Tidwell, O'Reilly, 2001.

An XML document is a tree. XPath allows you to concisely describe sets of XML elements and attributes in this tree.

Analogy to file system paths. A file system hierarchy is a tree of directories and files – a filesystem path describes how to reach a specific file, starting either from the root, or the current working directory.

XPath exploits this intuitive notion of a path, and extends it to XML documents.

XPath expressions are evaluated against XML after it has been read by an XML processor, and hence character entity references and CDATA sections have already been parsed, and hence are unavailable to the XPath processor.

In the XML tree available for XPath expressions, there are 7 node types:

- The root node (one per document)
- Element nodes
- Attribute nodes
- Text nodes
- Comment nodes
- Processing instruction nodes
- Namespace nodes

Notes: root node contains only one element node, and possibly many PI nodes and/or comment nodes.

Element nodes correspond to element tags (but not to the enclosed content)

Attribute nodes correspond to attributes defined on elements

Text nodes are the textual content of a node, and its children

Notes: Although an element node is the parent of its attribute nodes, those attribute nodes are not children of their parent. The children of an element are the text, element, comment, and processing instruction nodes contained in the original element. If you want a document's attributes, you must ask for them specifically.

Processing Instruction Nodes:

A processing instruction node has two parts, a name (returned by the `name()` function) and a string value. The string value is everything after the name, including whitespace, but not including the `?>` that closes the processing instruction.

Namespace Nodes

Namespace nodes are almost never used in XSLT stylesheets; they exist primarily for the XSLT processor's benefit. Remember that the declaration of a namespace (such as `xmlns:auth="http://www.authors.net"`), even though it is technically an attribute in the XML source, becomes a namespace node, not an attribute node.

Notion of *context*

Context is similar to the notion of a current working directory in filesystems. Path expressions are evaluated with respect to the context.

There are two forms of path expression: abbreviated, and unabbreviated

Consider the following XML document:

```
<A>
  <B>
    <C/>
  </B>
  <B>
    <D>
      <C id="123"/>
      <C id="456"/>
    </D>
  <E/>
</B>
</A>
```

The following are abbreviated path expressions (all assume the root is the context)

/A/B/C – the C element that is a child of a B element that is a child of an A element

//C – all three C elements

//B – both B elements

//D/C – the two C elements that are a child of D

/D/C – the empty set (no D element under root)

D/E – the empty set (no matching path expressions)

A path expression is a query whose result is a set of nodes. The result set can be empty, hold one node, or hold multiple nodes.

XPath wildcards:

- * - all element nodes in the current context – only selects element nodes, not attributes, text nodes, comments, processing instructions
- @* - all attribute nodes in the current context
- node() – all nodes in the current context, regardless of type
- // - zero or more elements can occur between the slashes

Unabbreviated path expressions:

/child::A/child::B/child::C – same as /A/B/C

//child::C – same as //C

//child::D/child::C – same as //D/C

Accesses surrounding a context node are divided into axes:

Parents: “parent::” or “..” selects the parent of the context node

Attributes: attributes of the current node

“attribute::type” or “@type” are equivalent

Self: “self::” or “.”

preceding-sibling – all nodes that precede the start of the context node, and have the same parent node

following-sibling – all nodes that follow the end of the context node, and have the same parent node if any

Predicates:

Can additionally constrain the number of nodes returned.

number – select only a given element number, as in /D/C[2] – the second C that is a child of D.

attributes – select only elements with a given attribute, as in C[@id="123"] – the C element that has the id “123”.

Example from XSLT, Tidwell, with DTD omitted:

```
<?xml version="1.0"?>
<?xml-stylesheet href="sonnet.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>

<!-- Default sonnet type is Shakespearean, the other allowable -->
<!-- type is "Petrarchan." -->
<sonnet type="Shakespearean">
  <auth:author xmlns:auth="http://www.authors.com/">
    <last-name>Shakespeare</last-name>
    <first-name>William</first-name>
    <nationality>British</nationality>
    <year-of-birth>1564</year-of-birth>
    <year-of-death>1616</year-of-death>
  </auth:author>
  <!-- Is there an official title for this sonnet? They're
    sometimes named after the first line. -->
  <title>Sonnet 130</title>
  <lines>
    <line>My mistress' eyes are nothing like the sun,</line>
    <line>Coral is far more red than her lips red.</line>
    <line>If snow be white, why then her breasts are dun,</line>
    <line>If hairs be wires, black wires grow on her head.</line>
    <line>I have seen roses damasked, red and white,</line>
    <line>But no such roses see I in her cheeks.</line>
    <line>And in some perfumes is there more delight</line>
    <line>Than in the breath that from my mistress reeks.</line>
    <line>I love to hear her speak, yet well I know</line>
    <line>That music hath a far more pleasing sound.</line>
    <line>I grant I never saw a goddess go,</line>
    <line>My mistress when she walks, treads on the ground.</line>
    <line>And yet, by Heaven, I think my love as rare</line>
    <line>As any she belied with false compare.</line>
  </lines>
</sonnet>
<!-- The title of Sting's 1987 album "Nothing like the sun" is -->
<!-- from line 1 of this sonnet.
```

Class exercise:

What is selected by the following path expressions (assuming the context node is root):

`/sonnet/lines/line`

a set containing all of the lines elements

`line`

empty set – there are no child elements of root named line (the only child element of root is sonnet)

`//line`

a set containing all of the lines elements

`/sonnet//line`

a set containing all of the lines elements

`/sonnet/@type`

the type attribute (and its value) of the sonnet element

`/sonnet/auth:author/text()`

the spaces after the finishing ">" of auth:author and before the starting "<" of lastname

`/sonnet/auth:author/last-name/text()`

the text "Shakespeare"

`/processing-instruction()`

a set containing two processing instruction nodes (xml-stylesheet and cocoon-process -- <?xml version=... is not considered a PI)

`/auth:*`

empty set – no children of root are in the auth: namespace

`//auth:*`

returns only the auth:author element

`/@auth:*`

empty set – there are no attributes in the auth: namespace

`/sonnet[@type="Shakespearean"]`

the sonnet element is selected