Introduction to Extensible Stylesheet Language Transformations

Extensible Stylesheet Language Transformations (XSLT) is a language used to transform documents. The source document is often an XML file and the destination document is a web page. However, other transformations are possible. In fact, the language is a full (but awkward) programming language. It allows for a number of computations.

An XML document forms a tree. The nodes can be the root node, elements, attributes, text, comments, processing instructions (tags beginning with <?), or namespaces. A language called XPath is used to find particular nodes in the tree. In conjunction with XSLT, it can be used to include selected data in the output.

First address example

This example was used previously, but now it has a link to an XSL stylesheet. Note that the type in the link is now ="application/xml", and the reference is to an xsl document.

```
<?xml version = "1.0" ?>
<?xml-stylesheet type="application/xml" href="address.xsl"?>

<address>
    <name>Alice Lee</name>
    <email>alee@aol.com</email>
    <phone>123-45-6789</phone>
    <birthday>1983-7-15</birthday>
</address>
```

The simplest stylesheet for it has no content.

```
    <?xml version="1.0" encoding="ISO-8859-1"?>
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    </xsl:stylesheet>
```

When this empty stylesheet is applied to the xml file, the result just echoes all the data.

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

        Alice Lee
        alee@aol.com
        123-45-6789
        1983-7-15
```

If you just want one piece of data, say the name, you can use the following stylesheet:

```
    <?xml version="1.0" encoding="ISO-8859-1"?>
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <xsl:template match="address">
            <xsl:value-of select="name"/>
        </xsl:template>
    </xsl:stylesheet>
```

The result of this is just the heading followed by the name.

```
<?xml version="1.0" encoding="UTF-8"?>
Alice Lee
```

The stylesheet creates a *template* that is used to *match* data in the file. This one matches the address node and *selects* the *value-of* the name element. We can select other elements the same way.

A more useful stylesheet transforms the file into a web page. Any markup is allowed in the stylesheet, so long as it is well-formed. Since xhtml is, it can be added to the file. Here simple html tags are included in order to produce a complete html file.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <xsl:template match="address">
        <html><head><title>Address Book</title></head>
        <body>
            <xsl:value-of select="name"/>
            <br/><xsl:value-of select="email"/>
            <br/><xsl:value-of select="phone"/>
            <br/><xsl:value-of select="birthday"/>
        </body>
        </html>
    </xsl:template>
</xsl:stylesheet>
```

The result of this is the html file:

```
<html>
<head>
<META http-equiv="Content-Type" content="text/html; charset=UTF-8">
<title>Address Book</title>
</head>
<body>Alice Lee<br>alee@aol.com<br>123-45-6789<br>1983-7-15</body>
</html>
```

Note that *white space* is not preserved in the html file. (White space consists of spaces, tabs, and end of line characters.) However, when displayed in a browser, the
 tag will place each value on a separate line. The data can also be put into a table by adding table tags, a list by using <ul> and <li>, etc. Empty tags such as
 must be closed with a '/', as shown above.

Second address example

In a real address book you would have a number of names. The next file has three entries.

```
<?xml version = "1.0" ?>
<?xml-stylesheet type="application/xml" href="address.xsl"?>

<address>
    <entry>
        <name>Alice Lee</name>
        <email>alee@aol.com</email>
        <phone>123-45-6789</phone>
        <birthday>1983-07-15</birthday>
    </entry>
```

```
        <entry>
            <name>Barbara Smith</name>
            <email>bsmith@yahoo.com </email>
            <phone>234-56-7890 </phone>
            <birthday>1982-11-25</birthday>
        </entry>
        <entry>
            <name>Cathy Jones</name>
            <email>cjones@hotmail.com </email>
            <phone>345-67-8901 </phone>
            <birthday>1984-02-05</birthday>
        </entry>
    </address>
```

The stylesheet for this file has a section for the address and a section for the entries.

```
    <?xml version="1.0" encoding="ISO-8859-1"?>
    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
        <!-- Web [age section -->
        <xsl:template match="address">
            <html>
                <head><title>Address Book</title></head>
            <body>
                <table border="1">
                    <xsl:apply-templates select="entry"/>
                </table>
            </body>
            </html>
        </xsl:template>
        <!-- Data section -->
        <xsl:template match="entry">
            <tr>
                <td><xsl:value-of select="name"/></td>
                <td><xsl:value-of select="email"/></td>
                <td><xsl:value-of select="phone"/></td>
                <td><xsl:value-of select="birthday"/></td>
            </tr>
        </xsl:template>
    </xsl:stylesheet>
```

This stylesheet produces a table. The resulting web page is next.

```
    <html>
    <head>
    <META http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <title>Address Book</title>
    </head>
    <body>
    <table border="1">
    <tr>
    <td>Alice Lee</td><td>alee@aol.com</td><td>123-45-6789</td><td>1983-07-15</td>
    </tr>
```

```
<tr>
<td>Barbara Smith</td><td>bsmith@yahoo.com </td><td>234-56-7890 </td><td>1982-11-25</td>
</tr>
<tr>
<td>Cathy Jones</td><td>cjones@hotmail.com </td><td>345-67-8901 </td><td>1984-02-05</td>
</tr>
</table>
</body>
</html>
```

The resulting table is shown below. It could be made more elaborate by adding cell padding, color, etc.

Alice Lee	alee@aol.com	123-45-6789	1983-07-15
Barbara Smith	bsmith@yahoo.com	234-56-7890	1982-11-25
Cathy Jones	cjones@hotmail.com	345-67-8901	1984-02-05

Third address example

The last address example subdivides the name and birthday elements.

```
<?xml version = "1.0" ?>
<?xml-stylesheet type="application/xml" href="address.xsl"?>

<address>
    <entry>
        <name>
            <first>Alice</first>
            <last>Lee</last>
        </name>
        <email>alee@aol.com</email>
        <phone>123-45-6789</phone>
            <birthday>
            <year>1983</year>
            <month>07</month>
            <day>15</day>
        </birthday>
    </entry>
    <entry>
        <name>
            <first>Barbara</first>
            <last>Smith</last>
        </name>
        <email>bsmith@yahoo.com </email>
        <phone>234-56-7890 </phone>
        <birthday>
            <year>1982</year>
            <month>11</month>
            <day>25</day>
        </birthday>
    </entry>
</address>
```

Since name and birthday are not complex elements, they must each have separate templates.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <!-- Web page section -->
    <xsl:template match="address">

        <html>
            <head><title>Address Book</title></head>
        <body>
            <table border="1">
                <xsl:apply-templates select="entry"/>
            </table>
        </body>
        </html>
    </xsl:template>

    <!-- Entry section -->
    <xsl:template match="entry">
        <tr>
            <xsl:apply-templates select="name"/>
            <td><xsl:value-of select="email"/></td>
            <td><xsl:value-of select="phone"/></td>
            <td><xsl:apply-templates select="birthday"/></td>
        </tr>
    </xsl:template>

    <!-- Name section -->
    <xsl:template match="name">
        <td><xsl:value-of select="first"/></td>
        <td><xsl:value-of select="last"/></td>
    </xsl:template>

    <!-- Birthday section -->
    <xsl:template match="birthday">
        <xsl:value-of select="year"/>-<xsl:value-of select="month"/>-<xsl:value-of select="day"/>
    </xsl:template>
</xsl:stylesheet>
```

The name section has been divided into two separate cells in the table, while the birthday section has hyphens between the year, month and day values.

Alice	Lee	alee@aol.com	123-45-6789	1983-07-15
Barbara	Smith	bsmith@yahoo.com	234-56-7890	1982-11-25
Cathy	Jones	cjones@hotmail.com	345-67-8901	1984-02-05

In the above, *match* is an XPath command used to identify a particular element, and *value-of* gets the value of that element.

The roster example

An XSLT stylesheet can also handle attributes. Consider the following XML file.

```
<?xml version="1.0"?>
<!DOCTYPE roster SYSTEM "roster.dtd">
<?xml-stylesheet type="application/xml" href="roster.xsl"?>

<roster>
    <student>
        <name>Alice Lee</name>
        <midterm weight="40">85</midterm>
        <final weight="60">92</final>
    </student>
    <student>
        <name>Barbara Smith</name>
        <midterm weight="40">78</midterm>
        <final weight="60">84</final>
    </student>
    <student>
        <name>Cathy Jones</name>
        <midterm weight="40">82</midterm>
        <final weight="60">87</final>
    </student>
</roster>
```

Both the midterm and final tags contain attributes.

The following stylesheet can be used to put the data in a table ignoring the attributes.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <!-- Web page section -->
    <xsl:template match="roster">
        <html>
            <head><title>Class Roster</title></head>
        <body>
            <table border="1" cellpadding="5">
                <xsl:apply-templates select="student"/>
            </table>
        </body>
        </html>
    </xsl:template>

    <!-- Data section -->
    <xsl:template match="student">
        <tr>
            <td><xsl:value-of select="name"/></td>
            <td><xsl:value-of select="midterm"/></td>
            <td><xsl:value-of select="final"/></td>
        </tr>
    </xsl:template>
```

```
</xsl:stylesheet>
```

The table is shown below.

Alice Lee	85	92
Barbara Smith	78	84
Cathy Jones	82	87

XPath

XPath is the part of XSLT that is used to find places in the XML tree, and then do something with them. An XPath command lets you travel either down the tree using the forward slash, '/', or back up the tree using a double slash, '//'. For example, if the current matched node is <student>, we can go down the tree to the weight attribute with midterm/weight. (Attributes are treated as children of the node they are in.) The root node is always denoted by a single slash, '/'. So if any XPath command begins with a slash, it starts at the root node.

A stylesheet can also do arithmetic and place the results in the output. An XSLT transformer reads +, -, and * the usual way, but uses *div* for divide and *mod* for the remainder (% in Java). There are also several functions available such as round (), which rounds the value to the nearest integer.

We can add a fourth column to our table that shows the average of the midterm and final grades. The stylesheet for this follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

    <!-- Web page section -->
    <xsl:template match="roster">
        <html>
            <head><title>Class Roster</title></head>
        <body>
            <table border="1" cellspacing="10">
                <caption><b>Class Roster</b></caption>
                <tr>
                    <th>Name</th><th>Midterm</th><th>Final</th><th>Average</th>
                </tr>
                <xsl:apply-templates select="student"/>
            </table>
        </body>
        </html>
    </xsl:template>

    <!-- Student section -->
    <xsl:template match="student">
        <tr>
            <td><xsl:value-of select="name"/></td>
```

```
                <td><xsl:value-of select="midterm"/></td>
                <td><xsl:value-of select="final"/></td>
                <td><xsl:value-of select="(midterm+final)div 2"/></td>
            </tr>
        </xsl:template>
    </xsl:stylesheet>
```

In the fourth select in the student section, the stylesheet computes the average as "(midterm+final)div 2". Simple arithmetic like this can be placed anywhere in a stylesheet. The html table appears below.

Class Roster

Name	Midterm	Final	Average
Alice Lee	85	92	88.5
Barbara Smith	78	84	81
Cathy Jones	82	87	84.5

Attributes

In the example above, we ignored the attribute, weight. But it is in there for a purpose. It indicates that for the final average, the midterm should be worth 40% and the final 60%. In a stylesheet attributes are denoted by placing an 'at' sign, '@', before them. So the weight attribute is shown as @weight.

But we cannot just replace the computation, "(midterm+final)div 2", by one containing the weights. We have to use XPath to get us from the <student> node to the weight nodes. These are children of the <midterm> and <final> nodes, respectively. So the computation above changes to

```
<xsl:value-of select="(midterm/@weight*midterm+final/@weight*final)div 100"/>
```

The resulting web page table is shown below. Note that the averages are not the same as before.

Class Roster

Name	Midterm	Final	Average
Alice Lee	85	92	89.2
Barbara Smith	78	84	81.6
Cathy Jones	82	87	85

Using a Java program for transformations

The Java 1.4 release contains full support for XSLT. The files are in the javax.xml.transform package. The transformation is done in an abstract class called Transform. The most important method in the class is the transform method. It has two parameters, an input source and an output destination.

Before using the transform method, you have to get a transformer factory object and with this a transformer. This is done with the following code.

```
TransformerFactory tFactory = TransformerFactory.newInstance ();
Transformer transformer = tFactory.newTransformer (new StreamSource (sourceFile));
```

Once you have a transformer, you can use it to transform the source.

```
transformer.transform  (new StreamSource (sourceFile),
                        new StreamResult (new FileOutputStream (outputFile)));
```

These methods throw several exceptions that either must be caught or thrown again.

The following Java program is a slight modification of the SimpleTransform example that comes with Xalan, a product of the Apache open source project. Xalan is available at http://xml.apache.org/xalan-j/, which has the following description: "Xalan-Java is an XSLT processor for transforming XML documents into HTML, text, or other XML document types." It can be compiled with Java 1.4 and run with any XML file that includes a stylesheet processing instruction.

```
import javax.xml.transform.*;
import javax.xml.transform.stream.*;
import java.io.*;

/* Use the TraX interface to perform a transformation in the simplest manner possible. */
public class SimpleTransform
{
    public static void main(String[] args)
    {
        String filename = "";
        BufferedReader stdin = new BufferedReader (new InputStreamReader (System.in));
        try
        {
            System.out.print ("XML filename: ");
            filename = stdin.readLine ();
            TransformerFactory tFactory = TransformerFactory.newInstance ();
            Transformer transformer = tFactory.newTransformer
                                      (new StreamSource (filename + ".xsl"));
            transformer.transform (new StreamSource (filename + ".xml"),
                                   new StreamResult (new FileOutputStream (filename + ".html")));
            System.out.println ("The result is in " + filename + ".html ");
        } catch (IOException e) {System.out.println ("IO Error");}
          catch (TransformerException e) {e.printStackTrace(System.err);}
    } // main
} // SimpleTransform
```