Wrocław University of Science and Technology

XSLT

Authors: Damian Kościk
Krzysztof Tomków

HR EXCELLENCE IN RESEARCH

INTRODUCTION

- XSL (eXtensible Stylesheet Language)- styling language for XML,

XSL more than a style sheet language

- XSLT – transforming XML
- XPath – navigating in XML
- XSL-FO – formatting XML
- XQuery – querying XML

How does it work ?

16. November 1999.

W3C
World Wide Web Consortium

XSLT elements

Templates declaration

```
<xsl:template>
<xsl:template name="example"> </xsl:...>
<xsl: template match="BB" mode="version1">
```

Applying templates

```
<xsl:apply-templates match="/AA/BB">
  <xsl:sort order = "descending" />
</xsl:apply-templates>
```

XSLT elements

<xsl:value-of select="XPath expression"/>

```
<xsl:value-of select="/BB/CC/@id"/>
```

Why?

Used to extract the value of an XML element and add it to the output stream

XSLT elements

```
<xsl:for-each select="...">
```

text to insert and rules to apply

```
</xsl:for-each>
```

Legal **filter** operators are:

- = (equal)
- != (not equal)
- < less than
- > greater than

```
<xsl:for-each select="catalog/cd[artist='Bob Dyl']">
```

XSLT elements

- <xsl:sort select="artist"/>

- <xsl:if test="condition">
- '<', '>', ELSE are not allowed in expressions

XSLT elements

```
<xsl:choose>
  <xsl:when test="condition1">...</xsl:when>
  <xsl:when test="condition2">...</xsl:when>
   ...
  <xsl:otherwise>...</xsl:otherwise>
</xsl:choose>
```

XPath

- XPath is a major element in the XSLT standard.
- Used in XSLT for matching templates
- Similar to a directory structure

XPath

nodename	• Selects all child nodes of the named node
/	• Selects from the root node
//	• Selects nodes from the current node that match the selection no matter where they are
.	• Selects the current node
..	• Selects the parent of the current node
@	• Selects attributes

XPath

/bookstore/book[1]	• Selects the first book element that is the child of the bookstore element
/bookstore/book[last()]	• Selects the last book element that is the child of the bookstore element
/bookstore/book[last()-1]	• Selects the last but one book element that is the child of the bookstore element
/bookstore/book[position()<3]	• Selects the first two book elements that are children of the bookstore element

XPath

*	• Matches any element node
@*	• Matches any attribute node
node()	• Matches any node of any kind
//book/title | //book/price	• Selects all the title AND price elements of all book elements
//title | //price	• Selects all the title AND price elements in the document
/bookstore/book/title | //price	• Selects all the title elements of the book element of the bookstore element AND all the price elements in the document

Example

- XML:

```
<AAA>
    <BBB id="1"/>
    <CCC/>
    <BBB id="2"/>
    <BBB name="bb"/>
    <DDD>
        <BBB/>
    </DDD>
    <CCC/>
</AAA>
```

//BBB[@id='2']

//AAA/BBB[@id='2']

- XML:

```
<AAA>
    <BBB id="1/>
    <BBB id="2"/>
    <BBB id="3"/>
    <BBB/>
    <BBB/>
    <BBB id="6"/>
    <BBB name="bb"/>
```

//BBB[position() mod 2 =0]

Time for practice!

Practice - run the first sample

1. Create .xml file
2. Create .xls file
3. Paste sample code, ex. from the tutorial on w3schools.com
4. Open .xml file in an Internet browser like Chrome, Opera, etc

Practice - run the first sample

Practice – run the first sample

Error message:

Unsafe attempt to load URL
file:///C:/Users/Krzysztof/Desktop/XSLT/example1.xsl
from frame with URL
file:///C:/Users/Krzysztof/Desktop/XSLT/example1.xml.
'file:' URLs are treated as unique security origins.

Wht is it so?

Because of security reasons Internet browsers block XML files from accessing local XSLT files

Issue solving

1. Open browser from terminal:

Open your browser from the terminal with a flag *--allow-file-access-from-files*

1. I opened PowerShell
2. Changed directory to the folder with chrome .exe file
3. Typed: .*chrome.exe --allow-file-access-from-files*

Issue solving

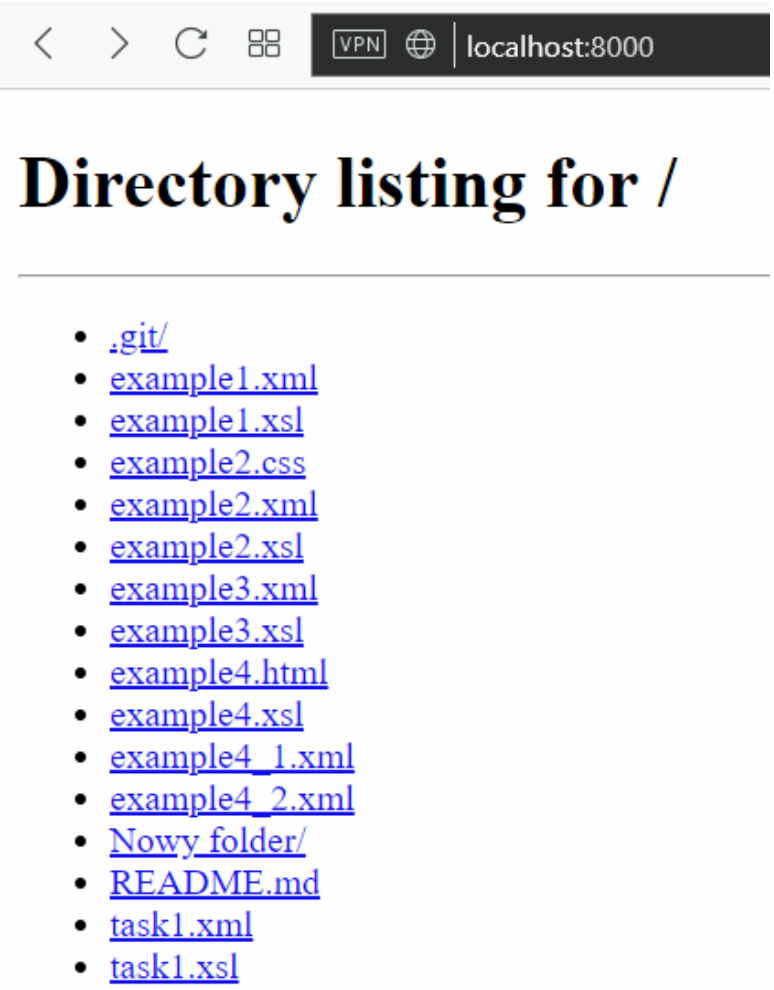
2. Use Python:

Python 2.x: *python -m SimpleHTTPServer*

Python 3.x: *python3 -m http.server* or *python -m http.server*

1. I opened PowerShell
2. Changed directory to the folder with all of my .xml and .xsl files
3. Typed: *python3 -m http.server*
4. Got: *Serving HTTP on 0.0.0.0 port 8000 (http://0.0.0.0:8000/)*

Issue solving – I recommend python



Demonstration files

All the demonstration files are created by the authors of this presentation.

Demonstration files and a manual how to run them are available at GitHub:

https://github.com/Sanoy2/XSLT

You will probably see this link again at the end of the presentation.Feel free to use it.

Source of information - .xml file

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="example1.xsl"?>
<cars>
    <car>
        <manufacturer>…</manufacturer>
        <model>…</model>
        <productionYear>…</productionYear>
        <colour>…</colour>
        <engine fuelType="…">
            <capacity>…</capacity>
            <power>…</power>
        </engine>
    </car>
    …
</cars>
```

.xsl file template

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0„
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html"/>
    <xsl:template match="/">
        <html>
            <body>

                    ...

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

Example 1 - .xsl - content of the body

```
<table>
    <tr>
        <th>Manufacturer</th>
        <th>Model</th>
        <th>Production Year</th>
        <th>Color</th>
        <th>Engine</th>
    </tr>
    <xsl:for-each select="cars/car">
    <xsl:sort select="manufacturer" order="ascending"/> <!-- ascending|descending -->
        <tr>
            <td><xsl:value-of select="manufacturer"/></td>
            <td><xsl:value-of select="model"/></td>
            <td><xsl:value-of select="productionYear"/></td>
            <td><xsl:value-of select="colour"/></td>
            <td>
                <ul>
                    <li><xsl:value-of select="engine/@fuelType"/></li>
                    <li><xsl:value-of select="engine/capacity"/></li>
                    <li><xsl:value-of select="engine/power"/></li>
                </ul>
            </td>
        </tr>
    </xsl:for-each>
</table>
```

Example 1 - result

localhost:8000/example1.xml

Manufacturer	Model	Production Year	Color	Engine
Citroen	C4 Grand Picasso	2007	Silver	• diesel • 1998 • 140
Ford	Mustang	1995	Red	• gasoline • 5496 • 240
Renault	Megane	2010	Black	• diesel • 1498 • 115
Toyota	Avensis	2015	White	• gasoline • 1599 • 110

Example 2 - .xml file

The same data

The only difference inside .xml file:

…

```
<?xml-stylesheet type="text/xsl"
href="example2.xsl"?>
```

…

Example 2 - .xsl file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
    <title>Cars 2example</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link rel="stylesheet" href="example2.css" />
</head>
<body>
```

Example 2 - .css file

```
th {
    background-color: #666699;
    color: #003300;
}
th, td {
    padding: 15px;
    text-align: left;
}
table {
    border-collapse: collapse;
}
table, th, td {
    border: 0.1em solid black;
    font: 15px arial, sans-serif;
}
tr:nth-child(even) {background-color: #cccccc;}
tr:nth-child(odd) {background-color: #808080;}
```

Example 2 - result

localhost:8000/example2.xml

Manufacturer	Model	Production Year	Color	Engine
Citroen	C4 Grand Picasso	2007	Silver	• diesel • 1998 • 140
Ford	Mustang	1995	Red	• gasoline • 5496 • 240
Renault	Megane	2010	Black	• diesel • 1498 • 115
Toyota	Avensis	2015	White	• gasoline • 1599 • 110

Example 3 - .xls file

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
    <title>Cars 3example</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"/>
</head>
<body>
    <div class="container">
        <table class="table table-striped">
            <tr>
```

Example 3 - result

localhost:8000/example3.xml

Manufacturer	Model	Production Year	Color	Engine
Citroen	C4 Grand Picasso	2007	Silver	• diesel • 1998 • 140
Ford	Mustang	1995	Red	• gasoline • 5496 • 240
Renault	Megane	2010	Black	• diesel • 1498 • 115
Toyota	Avensis	2015	White	• gasoline • 1599 • 110

Example 4 - .xml file - no link to .xsl!

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="example1.xsl"?>
<cars>
    <car>
        <manufacturer>…</manufacturer>
        <model>…</model>
        <productionYear>…</productionYear>
        <colour>…</colour>
        <engine fuelType="…">
            <capacity>…</capacity>
            <power>…</power>
        </engine>
    </car>
    …
</cars>
```

Example 4 - .xsl file

Exactly the same content as the previous one (it can be even literally the same file)

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="/">
<html>
<head>
    <title>Cars 3example</title>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/4.1.3/css/bootstrap.min.css"/>
</head>
<body>
    <div class="container">
        <table class="table table-striped">
            <tr>
```

Example 4 - .html file

```
<!DOCTYPE html>
<html>
    <head>
        <script>
         <!-- Some javascript magic -->
        </script>
            </head>
            <body>
                <div>
                    <input type="file" name="sourceFile"></input>
                    <button onclick="displayResult()">Run</button>
                </div>
            <div id="myBody" />
        </body>
</html>
```

Example 4 - javascript magic

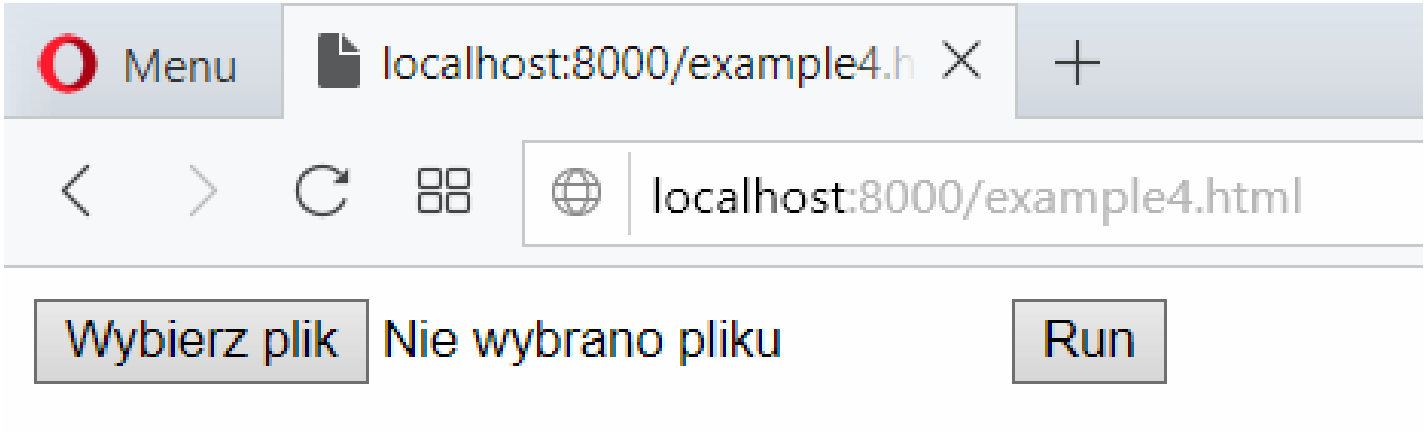
```
<script>
    function loadXMLDoc(filename) {
        if (window.ActiveXObject) {
            xhttp = new ActiveXObject("Msxml2.XMLHTTP");
        } else {
            xhttp = new XMLHttpRequest();
        }
        xhttp.open("GET", filename, false);
        try {
            xhttp.responseType = "msxml-document"
        } catch (err) {} // Helping IE11
        xhttp.send("");
        return xhttp.responseXML;
    }

    function displayResult() {
        var file = document.getElementsByName("sourceFile")[0].value;
        var filename = file.replace(/^.*[\\\/]/, '');
        xml = loadXMLDoc(filename);
        xsl = loadXMLDoc("example4.xsl");

        xsltProcessor = new XSLTProcessor();
        xsltProcessor.importStylesheet(xsl);
        resultDocument = xsltProcessor.transformToFragment(xml, document);
        document.getElementById("myBody").innerHTML = "";
        document.getElementById("myBody").appendChild(resultDocument);
    }
</script>
```

Example 4.1 - result

Run by opening .html file



Choose the file and type *Run*

Example 4.1 - result - example4_1.xml

Menu | Cars 4example

localhost:8000/example4.html

Wybierz plik example4_1.xml Run

Manufacturer	Model	Production Year	Color	Engine
Citroen	C4 Grand Picasso	2007	Silver	• diesel • 1998 • 140
Ford	Mustang	1995	Red	• gasoline • 5496 • 240
Renault	Megane	2010	Black	• diesel • 1498 • 115
Toyota	Avensis	2015	White	• gasoline • 1599 • 110

Example 4.2 - add if *to* .xsl

```
<xsl:for-each select="cars/car">
<xsl:sort select="manufacturer" order="ascending"/>
<xsl:if test="engine/@fuelType='gasoline'">

</xsl:if>
</xsl:for-each>
```

Example 4.2 - result - example4_1.xml

Manufacturer	Model	Production Year	Color	Engine
Ford	Mustang	1995	Red	• gasoline • 5496 • 240
Toyota	Avensis	2015	White	• gasoline • 1599 • 110

Example 4.3 - result - example4_2.xml

Manufacturer	Model	Production Year	Color	Engine
Audi	A6	2018	Grey	• gasoline • 1999 • 190
Fiat	Tipo	2017	Black	• gasoline • 1598 • 110
Fiat	Panda	2014	White	• gasoline • 1398 • 76
Fiat	Doblo	2015	Black	• gasoline • 1598 • 110

Example 4.4 - second sorting in .xsl

```
<xsl:for-each select="cars/car">
<xsl:sort select="manufacturer" order="ascending"/>
<xsl:sort select="model" order="ascending"/>
<xsl:if test="engine/@fuelType='gasoline'">
```

Example 4.4 - result - example4_2.xml

Manufacturer	Model	Production Year	Color	Engine
Audi	A6	2018	Grey	• gasoline • 1999 • 190
Fiat	Doblo	2015	Black	• gasoline • 1598 • 110
Fiat	Panda	2014	White	• gasoline • 1398 • 76
Fiat	Scudo	2010	Blue	• gasoline • 1998 • 140

One more issue

Wait, what?!

When you open content of the .xml file through .html file, like on the examples 4.x, there might be a situation that you change something in .xsl file but there is no difference in result after refreshing the page.

Then try to restart python server or clean the history of your browser. It usually helps. For mte additional free line between 2 sorting tags helped

Questions

Demonstration files and a manual how to run them are available at GitHub:

https://github.com/Sanoy2/XSLT

References

- http://www.zsk.ict.pwr.wroc.pl/zsk/repository/didactics/java_xml/inea101.pdf
- https://www.w3schools.com/xml/xsl_intro.asp
- https://stackoverflow.com/questions/3828898/can-chrome-be-made-to-perform-an-xsl-transform-on-a-local-fil
- **https://github.com/Sanoy2/XSLT**