ip: Call JavaScript from an XSLT style sheet

**Add functionality to your style sheets**

[Nicholas Chase](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#author1) ([nicholas@nicholaschase.com](mailto:nicholas@nicholaschase.com?subject=Call%20JavaScript%20from%20an%20XSLT%20style%20sheet&cc=dwxed@us.ibm.com)), President, Chase and Chase, Inc.

**Summary:**  XSLT style sheets allow you a great deal of freedom in manipulating the data in your original XML document. There are times, however, when you really want to do actual programming, and the XSLT recommendation is designed to allow that through the use of extensions. These extensions take the form of functions and elements and can be written in any language the processor supports. One of your options is to embed JavaScript within the style sheet, either directly or as an external file.

[View more content in this series](http://www.ibm.com/developerworks/views/xml/libraryview.jsp?search_by=tip:)

[Tag this!](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/)

[Update My dW interests](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/) ([Log in](https://www.ibm.com/developerworks/dwwi/DWAuthRouter?m=loginpage&d=http%3A%2F%2Fwww.ibm.com%2Fdeveloperworks%2Fxml%2Flibrary%2Fx-tipxsltjs%2F) | [What's this?](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#overlay))[Skip to help for Update My dW interests](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#dwmyinterestaddhelp)

**Date:**  01 Apr 2002   
**Level:** Introductory   
**Also available in:**  [Japanese](http://www.ibm.com/developerworks/jp/xml/library/x-tipxsltjs/)   
  
**Activity:**  39356 views   
**Comments:**   2 ([View](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#icomments) | [Add comment](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/) - Sign in)

Average rating 4 stars based on 29 votes Average rating (29 votes)  
[Rate this article](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#iratings)

This tip uses the Apache Project's Xalan Java 2 transformation engine and its implementation (see [Resources](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#resources)). The overall concepts are the same for any implementation, but the XSLT recommendation doesn't mandate any particular implementation method. In addition to Xalan, you will need the js.jar file (see [Resources](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#resources)), which contains an implementation of JavaScript, on your CLASSPATH, as well as bsf.jar, which is part of the Xalan distribution.

**The source document**

The example style sheet documents the entries in a guessing game, where players make three guesses of 1 to 100. The style sheet takes those three guesses and compares them to random numbers. The sample document contains two sets of guesses:

**The sample document**

|  |
| --- |
| <?xml version="1.0"?>  <entries gameID="DWO">  <entry>  <player>John</player>  <guess>3</guess>  <guess>9</guess>  <guess>222</guess>  </entry>  <entry>  <player>Mary</player>  <guess>88</guess>  <guess>76</guess>  <guess>5</guess>  </entry>  </entries> |

[**Back to top**](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#ibm-pcon)

**Create the component**

The first step in using an extension element or function is to define the code to be executed. This involves defining a new namespace and a container for the code:

**The basic style sheet**

|  |
| --- |
| <?xml version="1.0"?>  <xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"  **xmlns:lxslt="http://xml.apache.org/xslt"**  **xmlns:result="http://www.example.com/results"**  **extension-element-prefixes="result"**  version="1.0">  **<lxslt:component prefix="result" elements="rules" functions="getResult">**  **<lxslt:script lang="javascript">**  function getResult (thisGuess) {  var thisResult = parseInt(Math.random()\*100);  if (thisResult == parseInt(thisGuess)) {  return "Correct!";  } else {  return "Wrong! The actual answer was "+thisResult+  ", not "+thisGuess+".";  }  }  **</lxslt:script>**  **</lxslt:component>**  <xsl:template match="/">  <xsl:apply-templates/>  </xsl:template>  </xsl:stylesheet> |

On the surface, this is a typical style sheet, with the addition of two new namespaces. The first, with the prefix lxslt, tells the processor which elements define the new functionality. The second, result, indicates a call to the new functionality. Finally, theextension-element-prefixes attribute lets the processor know which elements should not be transformed as part of the normal flow. (They can still return a value to be output, as we'll see.)

The component itself specifies that all the code within it will be called from the result namespace prefix. It also lets the processor know which functions will be called from extension elements, and which from extension functions. The script element describes the functions themselves.

In this case, we're starting with a function that takes an argument and compares it to a random number from 1 to 100, returning a string that signifies the result.

[**Back to top**](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#ibm-pcon)

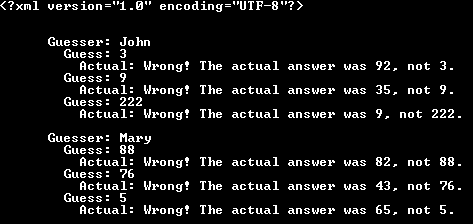
**Extension functions**

In an XSLT style sheet, extension functions actually extend XPath so you can use them just as you would a built-in function such as translate() or round().

**Calling a function**

|  |
| --- |
| ...  <xsl:template match="/">  <xsl:apply-templates/>  </xsl:template>  <xsl:template match="entry">  Guesser: <xsl:value-of select="player"/>  <xsl:apply-templates select="guess"/>  </xsl:template>  <xsl:template match="guess">  Guess: <xsl:value-of select="."/>  Actual: <xsl:value-of select="**result:getResult(string(.))**"/>  </xsl:template>  </xsl:stylesheet> |

This example passes the string value of the current node (guess) to the getResult() function. The namespace lets the processor know to trigger the function in the result component.

**Figure 1. Preliminary results**  


[**Back to top**](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#ibm-pcon)

**Using an element**

Extension elements are a bit more complex than functions. Rather than simply returning a value (although they can), they are intended to execute a particular action at a particular "time" within the processing of the style sheet. Also, rather than taking an arbitrary list of arguments, as an extension function can, the code behind an extension element has two well-defined arguments.

The rules element triggers the processing of the rules() function. This function takes as one of its arguments the ruleselement itself (elem), allowing you to retrieve the value of any custom attribute it carries.

[**Back to top**](http://www.ibm.com/developerworks/xml/library/x-tipxsltjs/#ibm-pcon)

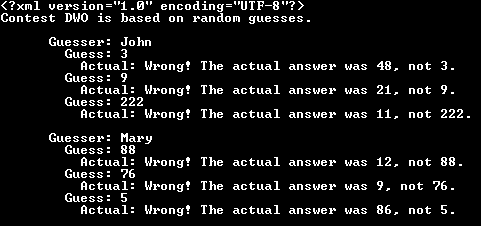
**Using the processor context**

Perhaps the most powerful aspect of an extension element is its ability to access the source document itself through the XSL processor context argument.

**The processor context**

|  |
| --- |
| ...  <lxslt:component prefix="result" elements="rules" functions="getResult">  <lxslt:script lang="javascript">  ...  function rules(ctx, elem) {  **ctxNode = ctx.getContextNode();**  **gameID = ctxNode.getFirstChild().getAttribute("gameID");**  return "Contest "+**gameID**+" is based on "+  elem.getAttribute("guessType")+" guesses.";  }  </lxslt:script>  </lxslt:component>  ... |

The first argument of the rules function is the processor context, in the form of theorg.apache.xalan.extensions.XSLProcessorContext object. This allows you to retrieve objects that represent the context node, the overall source tree, the style sheet, and the transformer currently performing the transformation. Accessing the context node is most common. Once returned by the getContextNode() method, this is a typical XML node, with typical DOM operations available.

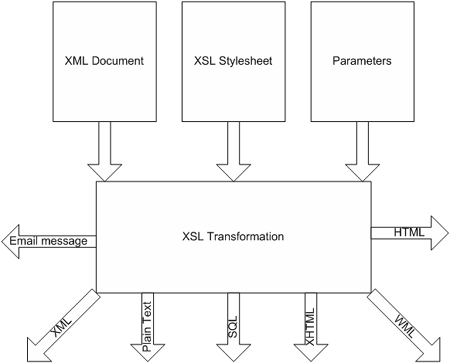
**Figure 2. The final output**  


**Resources**

* Check out the [XSLT recommendation from the W3C](http://www.w3.org/TR/xslt).
* Download Apache's [Xalan-Java 2](http://xml.apache.org/xalan-j/index.html).
* Download the [js.jar](http://www.mozilla.org/rhino/download.html) file.
* Find more XML resources on the [developerWorks XML zone](http://www.ibm.com/developerworks/xml/).
* [IBM trial software](http://www.ibm.com/developerworks/downloads/?S_TACT=105AGX06&S_CMP=art): Build your next development project with trial software available for download directly from developerWorks.

# XSLT

XSLT documents are valid XML documents that describe how another XML document should be transformed. For XSLT to work, it needs an XML document to transform and an engine to make the transformation take place. In addition, parameters can be passed in to XSLTs providing further instructions on how to do the transformation.

The diagram below shows how this all works.

## *Basic XSLT*

This lesson does not get into the details of the XSLT language. However, to get the idea of how it works, we'll take a look at a simple XML document and an XSLT stylesheet, which is used to transform the XML to HTML. Then we will look at how the browsers handle XSLT transformations.

## *Code Sample: XSLT/Demos/Paul.xml*

<?xml version="1.0"?>

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

<person>

<name>

<firstname>Paul</firstname>

<lastname>McCartney</lastname>

</name>

<job>Singer</job>

<gender>Male</gender>

</person>

**Code Explanation**

This is a straightforward XML document. The processing instruction at the top indicates that the XML document should be transformed using Beatle.xsl (shown below).

## *Code Sample: XSLT/Demos/Beatle.xsl*

<?xml version="1.0"?>

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

<xsl:output method="html"/>

<xsl:template match="child::person">

<html>

<head>

<title>

<xsl:value-of select="descendant::firstname" />

<xsl:text> </xsl:text>

<xsl:value-of select="descendant::lastname" />

</title>

</head>

<body>

<xsl:value-of select="descendant::firstname" />

<xsl:text> </xsl:text>

<xsl:value-of select="descendant::lastname" />

</body>

</html>

</xsl:template>

</xsl:stylesheet>

**Code Explanation**

Note that the document begins with an XML declaration. This is because XSLTs are XML documents themselves.

The second line is the document element of the XSLT. It states that this document is a version 1.0 XSLT document.

<xsl:stylesheet version="1.0"

xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

The third line (<xsl:output method="html"/>) indicates that the resulting output will be HTML.

The fourth line is an open <xsl:template> element. The match attribute of this tag takes an XPath, which indicates that this template applies to the person node of the XML document. Because person is the document element, this template will only run once.

There are then a few lines of HTML followed by two <xsl:value-of /> elements separated by one<xsl:text> element. The <xsl:value-of /> tag has a select attribute, which takes an XPath pointing to a specific element or group of elements within the XML document. In this case, the two <xsl:value-of />tags point to firstname and lastname elements, indicating that they should be output in the title of the HTML page. The <xsl:text> element is used to create a space between the first name and the last name elements.

<xsl:value-of select="descendant::firstname" />

<xsl:text> </xsl:text>

<xsl:value-of select="descendant::lastname" />

There are then some more HTML tags followed by the same XSLT tags, re-outputting the first and last name of the Beatle in the body of the HTML page. The output looks like this:

<html>

<head>

<meta http-equiv="Content-Type" content="text/html; charset=UTF-8">

<title>Paul McCartney</title>

</head>

<body>Paul McCartney</body>

</html>

# XSLT in the Browser

Unfortunately, XSLT transformations are handled differently by different browsers. We will see how they are handled by Internet Explorer and Mozilla and then will look at a library that makes cross-browser XSLT simple. First, a look at the documents:

### The XML

## *Code Sample: XSLT/Demos/Beatles.xml*

<?xml version="1.0"?>

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

<beatles>

<beatle link="http://www.paulmccartney.com">

<name>

<firstname>Paul</firstname>

<lastname>McCartney</lastname>

</name>

</beatle>

<beatle link="http://www.johnlennon.com">

<name>

<firstname>John</firstname>

<lastname>Lennon</lastname>

</name>

</beatle>

<beatle link="http://www.georgeharrison.com">

<name>

<firstname>George</firstname>

<lastname>Harrison</lastname>

</name>

</beatle>

<beatle link="http://www.ringostarr.com">

<name>

<firstname>Ringo</firstname>

<lastname>Starr</lastname>

</name>

</beatle>

</beatles>

### The XSLT

## *Code Sample: XSLT/Demos/Beatles.xsl*

<?xml version="1.0"?>

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

<xsl:template match="/">

<html>

<body>

<table border="1">

<xsl:for-each select="beatles/beatle">

<xsl:sort select="name/lastname"/>

<tr>

<td><a href="{@link}"><xsl:value-of select="name/lastname"/></a></td>

<td><a href="{@link}"><xsl:value-of select="name/firstname"/></a></td>

</tr>

</xsl:for-each>

</table>

</body>

</html>

</xsl:template>

</xsl:stylesheet>

### The Output

## *Code Sample: XSLT/Demos/Beatles.html*

<table border="1">

<tr>

<td>

<a href="http://www.georgeharrison.com">Harrison</a>

</td>

<td>

<a href="http://www.georgeharrison.com">George</a>

</td>

</tr>

<tr>

<td>

<a href="http://www.johnlennon.com">Lennon</a>

</td>

<td>

<a href="http://www.johnlennon.com">John</a>

</td>

</tr>

<tr>

<td>

<a href="http://www.paulmccartney.com">McCartney</a>

</td>

<td>

<a href="http://www.paulmccartney.com">Paul</a>

</td>

</tr>

<tr>

<td>

<a href="http://www.ringostarr.com">Starr</a>

</td>

<td>

<a href="http://www.ringostarr.com">Ringo</a>

</td>

</tr>

</table>

## *XSLT in Internet Explorer*

The steps for doing an XSLT transformation in Internet Explorer are as follows:

1. Load the XML document and XSLT document into XML DOM objects using ActiveX.
2. var xml = new ActiveXObject("Microsoft.XMLDOM");

var xslt = new ActiveXObject("Microsoft.XMLDOM");

1. Set the async property on both documents to false so that they both completely load before any further processing is attempted.
2. xml.async = false;

xslt.async = false;

1. Load XML and XSLT documents into the XML DOM objects.
2. xml.load("Beatles.xml");

xslt.load("Beatles.xsl");

1. Transform the XML to a string with the transformNode() method.

var output = xml.transformNode(xslt);

1. The result can then be output to the innerHTML property of any element on the page.
2. var outputDiv = document.getElementById("Output");

outputDiv.innerHTML = output;

This code shown above is in XSLT/Demos/Xslt-IE.html.

## *XSLT in Mozilla*

XSLT transformations in Mozilla are a bit more involved:

1. Load the XML document and XSLT document into XML DOM objects usingdocument.implementation.createDocument().
2. var xml = document.implementation.createDocument("", "", null);

var xslt = document.implementation.createDocument("", "", null);

1. Set the async property on both documents to false so that they both completely load before any further processing is attempted.
2. xml.async = false;

xslt.async = false;

1. Load XML and XSLT documents into the XML DOM objects.
2. xml.load("Beatles.xml");

xslt.load("Beatles.xsl");

1. Create a new XSLTProcessor object and use its importStylesheet() method to import the XSLT DOM object.
2. var processor = new XSLTProcessor();

processor.importStylesheet(xslt);

1. Transform the XML to a new XML DOM object with the transformToDocument() method of theXSLTProcessor.

var XmlDom = processor.transformToDocument(xml)

1. Create a new XMLSerializer and use it to serialize the new XML DOM object to a string.
2. var serializer = new XMLSerializer();

var output = serializer.serializeToString(XmlDom.documentElement);

1. The result can then be output to the innerHTML property of any element on the page.
2. var outputDiv = document.getElementById("Output");

outputDiv.innerHTML = output;

This code shown above is in XSLT/Demos/Xslt-FF.html.

# Sarissa

Sarissa is a JavaScript library (available at [http://sarissa.sourceforge.net](http://sarissa.sourceforge.net/)) that provides a cross-browser wrapper for XML APIs. In addition to XSLT transformations, the Sarissa library provides cross-browser methods for instantiating Document objects, loading XML from files and XML strings, and handling XPath queries.

The great thing about using Sarissa for XSLT transformations is that you don't have to learn yet a new API. Sarissa emulates Mozilla's API for Internet Explorer so that we can use to Mozilla method described above for cross-browser development. All we need to do is include the sarissa.js library and then use Sarissa's getDomDocument() method to create the XML and XSLT DOM objects. The file is shown below.

## *Code Sample: XSLT/Demos/Xslt-sarissa.html*

<html>

<head>

**<script type="text/javascript" src="../../sarissa.js"></script>**

<script type="text/javascript">

function XsltTransform()

{

**var xml = Sarissa.getDomDocument();**

**var xslt = Sarissa.getDomDocument();**

**xml.async = false;**

**xslt.async = false;**

**xml.load("Beatles.xml");**

**xslt.load("Beatles.xsl");**

**var processor = new XSLTProcessor();**

**processor.importStylesheet(xslt);**

**var XmlDom = processor.transformToDocument(xml)**

**var serializer = new XMLSerializer();**

**var output = serializer.serializeToString(XmlDom.documentElement);**

**var outputDiv = document.getElementById("Output");**

**outputDiv.innerHTML = output;**

}

</script>

<title>XSLT with Sarissa</title>

</head>

<body onload="XsltTransform();">

<div id="Output"></div>

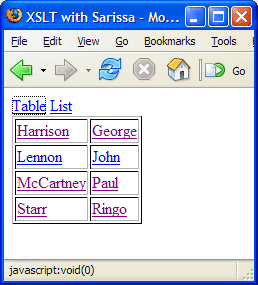
</body>

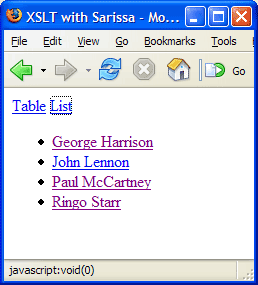
</html>

## *Exercise: XSLT Transformations with Sarissa*

*Duration: 15 to 25 minutes.*

In this exercise, you will use the Sarissa library to create a page that gives the user two views of the same data. When the page loads, it should display two links, one that reads "Table" and another that reads "List".

When the "Table" link is clicked, the Beatles are displayed in a table.

When the "List" link is clicked, the Beatles are displayed as a list.

The XML and XSLT documents are created already (Beatles.xml, Table.xsl, and List.xsl). Your job is to finishBeatles.html.

1. Open XSLT/Exercises/Beatles.html in your editor.
2. Write the JavaScript code to create the transformations.
3. Test your solution in a browser.

## *Code Sample: XSLT/Exercises/Beatles.html*

<html>

<head>

<script type="text/javascript" src="../../sarissa.js"></script>

<script type="text/javascript">

**//Write your JavaScript code here.**

</script>

<title>XSLT with Sarissa</title>

</head>

<body onload="XsltTransform('Table.xsl');">

<div>

<a href="javascript:void(0)" onclick="XsltTransform('Table.xsl');">Table</a>

<a href="javascript:void(0)" onclick="XsltTransform('List.xsl');">List</a>

</div>

<div id="Output"></div>

</body>

</html>

In sarissa, the load() method of DomDocument is deprecated. The documentation recommends using XMLHttpRequest instead. Using prototype, fix your solution to load the XML and XSLT documents using XMLHttpRequest. Note that you will not be able to do the transformation until both documents are completely loaded.

[Where is the solution?](javascript:alert('Solutions%20are%20not%20shown%20in%20the%20online%20version%20of%20this%20manual.%20%20For%20information%20on%20licensing%20our%20courseware%20or%20getting%20training,%20see%20the%20top%20of%20this%20page.');)

## *Advantages and Disadvantages of XSLT in Ajax Applications*

XSLT provides a very good standard method for transforming XML documents. Many developers find it easier to write XSLTs than to write DOM-manipulation JavaScript code.

The only real downside of XSLT is that it is a pretty complicated language. If you don't know it already, you may find that it's not easy to pick up without some serious studying.

# XSLT Transformations with JavaScript Conclusion

In this lesson of the Ajax tutorial, you have learned how to transform XML documents with XSLT in the browser and seen how these transformations can be used inside of Ajax applications.

To continue to learn Ajax go to the [top of this page](http://www.learn-ajax-tutorial.com/Xslt.cfm" \l "menuHeading) and click on the next lesson in this Ajax Tutorial's Table of Contents.

<http://www.learn-ajax-tutorial.com/Xslt.cfm>