

Accelerate innovation in the world of distributed computing

# Shedding a Little Light on XML

Is it possible to manage an XML addiction and maintain a normal life at the same time? Well, I'm trying my best. You see, I've spent the last few years really enjoying working in this technology and balancing it against a passion for table tennis, traveling, and somehow squeaking out a personal life.

This is the life of a developer.... Although I haven't been to my first XML anon meeting yet, I'm close. Picture this:

- **Me:** Hello, my name is David and I'm an XML addict.
- **Them:** Hi Dave (a group of pocket protector-wearing folks say in unison).

By day, I work as chief XML evangelist for Infoteria. By night, you can find me answering questions and adding demos to my Web site, [www.XML-Pitstop.com](http://www.XML-Pitstop.com). This monthly column is an opportunity for me to try to answer those questions that tug at our brain cells and beg to be answered. Yes, we all have them. So take this opportunity to send them in my direction at [dsilverlight@infoteria.com](mailto:dsilverlight@infoteria.com). I'll be answering questions in this column, choosing those that fall into one or more of the categories below:

- **FAQ:** What you always wanted to know
- **Black Belt:** The toughest of the tough
- **Customizing XSLT:** My personal favorite and by far the most popular
- **Going Wireless:** Bring the world of wireless to your eyes

But wait, there's more....

Another regular section of this column will be "Lessons in Style" dedicated to style, style, style. It'll cover a topic related to XSLT that's relevant, cool, and usually something that has caught my eye. Basically, a topic worth sharing with all of you.

As it'll probably show in my columns, I'm a bit partial to XSLT technology so I've decided to start off by answering a few stylesheet questions whose answers have some subtle tricks behind them, and a somewhat "controversial" XML question.

So, before we go on, let me encourage you to send me those XML questions. Remember, as disturbing as it is, this is my idea of a good time. I can't guarantee that I can get to all of them, but I'll give it my best. Hopefully, I can shed a little light on the subject for you.

## Customizing XSL

**Q:** Is it possible to pass a parameter to a stylesheet so I can have a conditional select?

**A:** This question is best answered in two steps. First step, create a stylesheet that defines and uses a parameter. The second step involves using the IXSLTemplate object, found in MSXML 2.6 and on, to define the parameter and pass it to the stylesheet.

**Note:** Passing binding values to top-level parameters is supported by many command-line invoked processors, but an XSLT processor is not obliged to provide a mechanism for binding invocation parameters to top-level parameters. The key is to be aware of the environment in which your stylesheets will be processed.

For the demos in this first column I'll be using a common programmer pas-

time – drinking – as the source for our XML data. In this first question we'll be sending a string as a parameter so we can select drinks of a certain category.

### Step 1: Create a parameterized stylesheet

At the top of your stylesheet you'll need to define the parameter we'll be passing in. In this example our parameter is named "drinktype" so the parameter definition will appear as follows:

```
<xsl:param name="drinktype"/>
```

**Note:** Since we're not specifying the value explicitly in our declaration, the default binding value is the empty string.

Now that we've defined a parameter, we'll modify our root rule (see Listing 1) to utilize a predicate that respects the parameter that will be passed in. As you'll see in the code segment below, the root rule uses the predicate of ["type=\$drinktype"] to return only the child nodes that have a type-element child whose node value is equal to the drinktype parameter.

```
<xsl:apply-templates
select="drinks/drink[type=$drink-
type]" />
```

### Step 2: Passing the parameter to the DOM

The method we're using to pass the drinktype parameter to the DOM involves the IXSLTemplate object, new to MSXML 2.6 and on, which allows us to work with compiled, cacheable XSLT stylesheets.

The data is provided in Listing 2 and the full source code is shown in Listing 3, but I'll point out the "important stuff."

First off, we're using the FreeThreaded DOM documents rather than the plain

## AUTHOR BIO

David Silverlight is the chief XML evangelist for Infoteria. He has been working in the trenches for a number of years as a software architect and consultant, specializing in database-driven Web applications. He also maintains [www.xmlpitstop.com](http://www.xmlpitstop.com), a resource for XML examples, resources, and everything else XML.

vanilla DOMDocument objects, since it's a requirement of the Template object.

'Since we are using the Template object, we will be using FreeThreaded DOM documents.

```

Set objXML =
Server.CreateObject("Msxml2.FreeThrea
dedDOMDocument.3.0")
Set objXSL =
Server.CreateObject("Msxml2.FreeThrea
dedDOMDocument.3.0")

'Load our XML and XSL
objects (FreeThreaded, that is)
objXML.async = False
objXML.Load
Server.MapPath("drinks.xml")

objXSL.async = False
objXSL.Load
Server.MapPath("drinkparams.xml")

'Set the value that we are
passing to the parameter
strDrinkType =
Server.Request("DrinkType")
    
```

We're creating an instance of our stylesheet object as a Template object. From the Template object we also get the Processor object. Using the Template object we set the Stylesheet object to our recently loaded XSL document. Next, we create a Processor object and

set the input source with the recently loaded XML document.

```

'Create an instance of our XSL
Template object
Set objTemplate =
Server.CreateObject("MSXML2.XSLTempla
te.3.0")

'Create an instance of our
Stylesheet object using our recently
loaded XSLT document
Set objTemplate.stylesheet =
objXSL

'Create an instance of our
Processor object
Set objProcessor =
objTemplate.createProcessor

'Define the input object for
our object equal to our recently
loaded XML document
objProcessor.input = objXML
    
```

Now we're ready to do what we came here for – adding the parameter. Whew! Using the AddParameter method of the Processor object, we add the parameter with the value that we wish to pass to it. If desired, we could have passed a node-set and/or other parameters.

```
objProcessor.AddParameter
"drinktype", strDrinkType
```

At this point, there's not much more to do than to transform our objects,

which now have the parameter applied to them. The output for the transformation will now be written out to the client. All done.

```
objProcessor.Transform
strHTML = objProcessor.output
Response.write strHTML
```

**Q:** How do I dynamically change both the sort field and sort order on a stylesheet? I tried to use a match pattern of "select=\$variable" and it never returns any data. Why?

**A:** If the select attribute doesn't express a relative expression to the node being sorted, the evaluation of each node will return the identical value. The end result is that the node list won't change its order from document order, and thus appears unsorted.

To demonstrate this we'll use two variables, \$sortorder and \$sortfield. In the code snippets below we sort our Drinks.xml document (see Listing 2) in ascending order by name and each criteria will be stored in their respective variables.

**1. Setting the sorting and selecting variable values**

```

<xsl:variable name="sortorder"
select="'ascending'"/>
<xsl:variable name="sortfield"
select="'name'"/>
    
```

If we wish to replace the variables with parameters, this is easily accomplished using the technique discussed in the prior question. Using the combina-

AREA	DOM	SAX
<b>Processing Mechanics</b>	DOM parses the entire document into a tree structure that's exposed as a collection of objects. The objects are easy to manipulate and give you a very convenient way to access specific objects in this structure. If you're comfortable working in an object-oriented environment, DOM will make life simple for you.	SAX is an API-based approach that essentially reads your document from beginning to end and raises events for the constructs as it processes them. It raises an event for you to process for each construct it recognizes, then moves on. It can stop when it reaches a condition or continue to EOF.
<b>Overhead</b>	DOM loads the entire document into memory, and requires a memory approximately three times the size of the file and a larger corresponding amount of memory. For larger files, the data will be spooled to a disk, which will have a slight negative effect on performance, but possibly an immense negative affect on capacities.	SAX will read only a segment of the file that it's processing into memory, thereby keeping memory requirements consistent, regardless of the size SAX processing overhead is constant, requiring a fixed amount of memory to process the file, no matter how large the file is. SAX can offer a significant reduction in memory requirements for very large files.
<b>Ideal Data Conditions</b>	-Data small to medium in size -Data with complex relationships -Data in which the complete structure must be loaded into memory -Data that might require random access -Documents that will require updating	-Very large data files -Files in which you might need to extract small pieces of data from a huge data source -Requirement to process large documents at high speeds -Documents that don't require random access -Documents in which the programmer doesn't need knowledge of the entire structure, but rather only the section of the document that's being processed at the current position.

TABLE 1 Dom versus SAX

tion, in fact, offers you an elegant way of creating parameter-based stylesheets with full sorting and filtering capabilities.

## 2. Applying the sorting/filtering variables

A common first attempt at dynamic sorting is to use the choose construct to apply the criteria based on its value, as shown below.

```
<xsl:choose>
  <xsl:when $sortorder = "ascending">
<xsl:sort select="*[name()=
$sortorder]" order="ascending"/>
  </xsl:when>
  <xsl:when $sortorder = "descending">
<xsl:sort select="*[name()=
$sortorder]" order="descending"/>
  </xsl:when>
  <xsl:otherwise>
<xsl:sort select="*[name()=
$sortorder]" order="ascending"/>
  </xsl:otherwise>
</xsl:choose>
```

However, the following is a more elegant approach (see Listing 4 for a full code example):

```
<xsl:sort select="*[name()=$sort-
field]" order="{ $sortorder}"/>
```

Keep in mind that the above “elegant” approach assumes that the node we’re basing the sort on is an element child of the nodes being sorted.

### Trick Question: Why will option “C” fail?

A. This will work: `<xsl:sort select="*[-name()=$sortfield]" order="ascending"/>`

B. This will work: `<xsl:sort select="name" order="ascending"/>`

C. This won't work: `<xsl:sort select="-$sortfield" order="ascending"/>`

On the surface it appears you should be able to simply pass the variable as your match pattern in the select clause. The reason this won't work is that the select clause is a node-set expression as opposed to the static string of “drinktype.” Using “select=-\$sortorder”, every node evaluates to the identical value so nodes remain in document order.

It's all how you handle the data

**Q:** I've heard in general that SAX should be used to process large XML files, whereas DOM can be used to process small files, say 1MB or smaller. How are the two processors different and when should they be used?

**A:** This is a simple question that has raised a lot of controversy. As you may be aware, there's a DOM versus SAX war out there. At risk of mortally offending those of you who are on one side or other, I'll try to list out some important side-by-side comparisons. It's not as simple as using SAX for large files and DOM for small files, though this is an important criteria when considering the burden on capacities.

Simply enough, the right parser depends on the type of data you're processing and the type of results you're looking for. As Table 1 indicates, a number of factors are involved in your decision of whether to use SAX or DOM. Investing a small amount of time to analyze your environment is likely to prevent the potential requirement for a code rewrite. ☹

DSILVERLIGHT@INFOTERIA.COM

### LISTING 1 DrinkParams.xsl

```
<?xml version='1.0' encoding='utf-8' ?>
<xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver-
sion="1.0">
<xsl:output method="html"/>

<!-- Display of drink listing -->
<xsl:param name="drinktype" select="'house'"/>

<xsl:template match="/">
  <html>
    <head><title></title></head>
    <body bgcolor="beige">
      <xsl:apply-templates
select="drinks/drink[drinktype=$drinktype]" />
    </body>
  </html>
</xsl:template>

<xsl:template match="drink">
  <xsl:apply-templates select="name" />
  <xsl:apply-templates select="directions" />
  <xsl:apply-templates select="ingredients" />
</xsl:template>

<xsl:template match="name">
  <font color="darkred" face="arial" size="5">
  <xsl:value-of select="." /><BR/><BR/>
  </font>
</xsl:template>
```

```
<xsl:template match="directions">
  <font face="arial black" size="3" >Directions: </font>
  <xsl:value-of select="." />

  <BR/><BR/>
</xsl:template>

<xsl:template match="ingredients">
  <font face="arial black" size="3" >Ingredients:</font>
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="ingredient">
  <ul><xsl:value-of select="." /></ul>
</xsl:template>

</xsl:stylesheet>
```

### LISTING 2 Drinks.xml

```
<?xml version="1.0"?>
<drinks>
  <drink>
    <name>AMARETTO ALEXANDER</name>
    <ingredients>
      <ingredient>1 shot Amaretto</ingredient>
      <ingredient>1 shot Creme de Cacao</ingredient>
      <ingredient>fill Cream</ingredient>
    </ingredients>
    <directions>Combine liquers in shaker along with cream
and ice. Shake and strain into a
brandy snifter. Garnish with a sprinkle of nut-
```

```

meg.</directions>
  <glasstype>brandy snifter</glasstype>
  <drinktype>house</drinktype>
</drink>

  |
  |
  .
  .
  .
  |
  |
<drink>
  <name>CHOCOLATE MARTINI</name>
  <ingredients>
    <ingredient>1 shot white creme de cacao</ingredient>
    <ingredient>1 shot Vodka</ingredient>
  </ingredients>
  <directions>Shake ingredients with ice, then strain into
a shot glass.</directions>
  <glasstype>shot glass</glasstype>
  <drinktype>shot</drinktype>
</drink>
</drinks>

```

### LISTING 3 PassingParams.asp

```

<%@ Language=VBScript %>
<%

    Dim objTemplate, objProcessor, strDrinkType,objXML,
objXSL, strHTML

    'Set the value that we are passing to the parameter
strDrinkType = "shot"

    'Since we are using the Template object, we will be
using FreeThreaded DOM documents.
    Set objXML =
Server.CreateObject("Msxml2.FreeThreadedDOMDocument.3.0")
    Set objXSL =
Server.CreateObject("Msxml2.FreeThreadedDOMDocument.3.0")

    'Load the XML document
objXML.async = False
objXML.Load Server.MapPath("drinks.xml")

    'Load the XSL document
objXSL.async = False
objXSL.Load Server.MapPath("DrinkParams.xsl")

    'Create an instance of our XSL Template object
Set objTemplate =
Server.CreateObject("MSXML2.XSLTemplate.3.0")

    'Create an instance of our stylesheet object using
our recently loaded XSLT document
Set objTemplate.stylesheet = objXSL

    'Create an instance of our Processor object

```

```

Set objProcessor = objTemplate.createProcessor

    'Define the input object for our object equal to our
recently loaded XML document
objProcessor.input = objXML

    'Now, finally we can add any parameters that we
require to our Template processor
    'In this example, we are setting the drinktype =
"shot"
objProcessor.AddParameter "drinktype", strDrinkType

    'Last but not least we do our transformation and
return the output to the client
objProcessor.Transform

    strHTML = objProcessor.output

    Response.write strHTML

%>

```

### LISTING 4 DrinksSorted.xsl

```

<?xml version='1.0' encoding='utf-8' ?><xsl:stylesheet
xmlns:xsl="http://www.w3.org/1999/XSL/Transform" ver-
sion="1.0"><xsl:output method="html"/>

<!-- Display of drink listing -->
<xsl:variable name="sortorder" select="'ascending'"/>
<xsl:variable name="sortfield" select="'name'"/>
<xsl:template match="/"> <html>
  <head><title></title></head>
  <body bgcolor="ivory" text="#000000" link="#990033"
vlink="#663300"><FONT color="LightBlue" face="Verdana">
  <h1>Summary listing of bar drinks</h1>
  <Table border="1" cellPadding="2" cellSpacing="1"
valign="TOP">
  <tr bgcolor="lightblue">
  <th>Drink Name</th>
  <th>Directions</th>
  </tr>
  <xsl:for-each select="drinks/drink" >
  <xsl:sort select="*[name()=$sortfield]" order="{sorsortorder}"/>

  <tr>
  <td bgcolor="lightblue"><xsl:value-of
select="name"/></td>
  <td><xsl:value-of select="directions"/></td>
  </tr>
  </xsl:for-each>
  </Table>
  </FONT>
  </body>
  </html>
</xsl:template>

</xsl:stylesheet>

```