



*When your own brain just isn't enough*

# 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 month, we'll focus on an important aspect of problem solving: finding the correct resources. The answers in this month's Q&A will demonstrate techniques that have had a large influence from a variety of invaluable resources, a few of which I'll mention here. In future columns, others will come into play.

I'm a strong believer in networking among experts in the field. Oddly enough, most of my networking is done with people I've never met face to face. These days, it just isn't necessary. In fact, a face-to-face meeting with a fellow computer geek at 3 a.m. isn't something I would desire, either.

There are a variety of folks I've had the opportunity and privilege to learn from, most of whom I've interacted with only through e-mail and/or messenger. Regardless of whether or not I'd recognize them as they walked down the street, I've benefited from their expertise on many an occasion. In this field, it isn't always a matter of being able to solve a problem. Sometimes it's just knowing where to look when you need help breaking through a tough section of code.

## 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.

**Q:** *How do I return the top highest x elements grouped from an unsorted nodeset?*  
**A:** Simple? On the surface, perhaps. The implementation of this solution involves a binary search, recursion, and a modular implementation of templates. (The last point isn't absolutely necessary, but should become common practice to allow code reuse.)

This question generated a pretty long thread in the XSLTalk (an advanced XSLT/XPath programming group at [http://](http://groups.yahoo.com)

[groups.yahoo.com](http://groups.yahoo.com)). Dimitre Novatchev, from VBXML.com, came up with an innovative solution. Due to its clever implementation, we will refer to it as the "Novatchev Method." The algorithm is described below. Be sure to download the source code; it's well worth checking out.

## Why Is This Difficult?

The reason this is tricky is that, although it's easy to return the top x elements alone and it's easy to group elements together, it is not easy to do the combination without looping through all of the elements of each group. If the nodeset was very large, it would be a lot of unnecessary work.

## Breakdown – 50,000 foot level

- Create a delimited string from the nodeset from which you wish to extract the top x elements.
- Parse through the dates using a single pass, finding the correct order for each date as you go. The key to performing this step efficiently is to use a binary search to quickly locate the sorted position in the string.
- The binary search is implemented as a template, which can be included into any XSLT application that would require this functionality.
- The end result is a sorted, delimited string, which is fed into a template to display as output.

## PseudoCode Level: INCLUDING TEMPLATES

The first section of code is the include section. Here we are incorporating templates that will be used into our solution. This is a technique that you should adopt if you have not already done so. It allows

you to use functionality like searching and displaying (in this example) into this stylesheet and into any other stylesheet that fits the same requirement.

```
<xsl:include
  href="BinarySearch.xsl"/>
<xsl:include
  href="DisplayStringAsTable.xsl"/>
```

## RECURSIVE INLINE SORTING

The bulk of our work is performed using a template that will recursively make a single pass through our delimited date string and that will feed each date into our binary search template (see Listing 1).

As each date position is located using our binary search, it's merged into its proper location based on the Merge-Position and the length of the sort element (see Listing 2).

## DISPLAY OUR SORTED OUTPUT

Upon completion of our sorting, we send the delimited string to a template used for display. This template, which can be used to display the column of data, will take our delimited string and create a basic HTML table from it (see Listing 3).

Although you seen some segments of the source code in this answer, the full source code to this solution can be downloaded with the article.

**Q:** *Combining stylesheets: Should I use <xsl:import> or <xsl:include>?*

**A:** As was touched upon in Question 1, you can modularize your templates and reuse them across stylesheets. Taking this approach will aide you in code reuse and help to create a more generic, "black-box" view of the way you work with XSLT templates. That being said, the two mechanisms defined in the XSLT 1.0

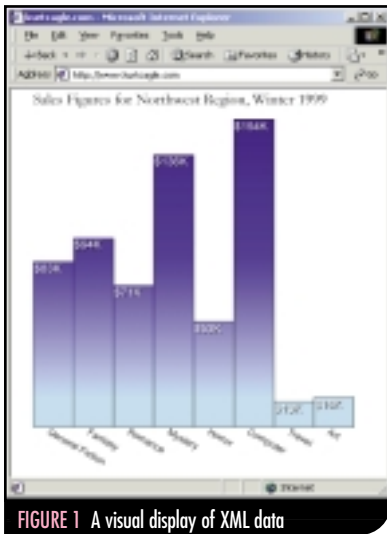


FIGURE 1 A visual display of XML data

spec for combining stylesheets are the top-level `<xsl:import>` and `<xsl:include>` elements. The differences between the behavior of these elements deal mainly with import precedence. Understanding them will help you decide which element to use in your stylesheet scenario.

The primary difference between the two is that when you use `<xsl:import>`, the import precedence of the stylesheet that's doing the importing has a higher precedence than the stylesheet that's being imported, whereas when you use the `<xsl:include>` the precedence is the same.

An error condition will be generated if you use `<xsl:include>` to incorporate a stylesheet that already exists in the importing stylesheet. The way the error is handled depends on the XSLT processor. Most processors will quietly handle the error by using the last template they encounter. Essentially, the last one found wins. In fact, the XSLT 1.0 spec states that "An XSLT processor may signal the error; if it does not signal the error, it must recover by choosing, from among the matching template rules that are left, the one that occurs last in the stylesheet."

In a nutshell, you should use `<xsl:import>` when your intention is to control precedence over templates and variables. You should use `<xsl:include>` in cases where precedence is not an issue.

**Q:** *How do I combine multiple XML documents?*

**A:** This question is one of the most common that come my way. Since many applications require data from more than one source, there is often a need to combine XML documents into a single data source before transformation. The more I play around, the more ways I find to accomplish this. I've decided to answer this by displaying a number of ways I've found to combine XML, along with the contributor who originally inspired the example. The case scenario for this answer is a set of XML documents that contain product data from different regions (NW, SW, etc.)

- **Jeni Tennison Version A:** Creating a reference to a parsed Entity in your XML documents.  
<http://sources.redhat.com/ml/xsl-list/2000-06/msg01290.html>

```
JeniCombineA.xml
<!DOCTYPE allproducts [
  <!ENTITY resourceA SYSTEM
    "ProductSalesSW.xml">
  <!ENTITY resourceB SYSTEM
    "ProductSalesNW.xml">
]>

<allproducts>
  &resourceA;
  &resourceB;
</allproducts>
```

- **Jeni Tennison Version B:** In this example, the name of the XML document is stored as an attribute value. The `document()` function is used in the stylesheet to create an instance of each XML document (see Listing 4).  
<http://sources.redhat.com/ml/xsl-list/2000-06/msg01290.html>

- **G. Ken Holman:** Creating a reference to an Unparsed Entity in your XML documents (see Listing 5).  
Source: "Practical Transformation Using XSLT and XPath ISBN 1-894049-06-3," [www.Cranesoftwrights.com](http://www.Cranesoftwrights.com)

- **Trace Wilson:** This code snippet uses a combination of the XML DOM and asp to merge the documents together (see Listing 6).  
<http://groups.yahoo.com/group/XSLTalk/message/343>

- **Kurt Cagle:** This example shows how we can use the document function with multiple nodes. Note how we can apply an XPath query to our variable, which holds the XML document names. (see Listing 7)  
<http://groups.yahoo.com/group/XSLTalk/message/1257>

- **Chris Lovett:** A glimpse into the future. XInclude is not a recommendation yet, although it is supported by Cocoon. Cool things to come....(see Listing 8).  
<http://msdn.microsoft.com/xml/articles/xml05292000.asp>

## Elements of Design – Displaying XML Data Graphically

We'll kick off the first issue of "Elements of Design" with a question: What's the most innovative thing you can do with XML? Each month we'll focus on a different XML implementation that shows off some of the novel applications of XML.

This month we'll be highlighting a method of visually displaying our XML data. In Figure 1, we show a graphic illustration of XML data containing sales numbers for various regions of a company. How is this done? Well, SVG is the underlying technology that's used to render this image. SVG, like many XML-based technologies, is still seeing only the beginnings of what it is capable of.

The full source code to the XML and XSL documents used for this demonstration and other SVG implementations can be found at Kurt Cagle's site at [www.KurtCagle.com](http://www.KurtCagle.com). Additional links to tons of SVG resources can be found at the SVG section of my own site, [www.xmlpitstop.com](http://www.xmlpitstop.com).

You will be able to find answers to questions such as:

- What is SVG?
- How can I display SVG in my browser?
- What's the current status of the W3C SVG spec?
- Others that are too numerous to answer in this column...

If you have an innovative, clever or just downright cool example of any XML-centric code that you'd like to share with the rest of the world, e-mail me. ✉

DSILVERLIGHT @ INFOTERIA.COM

### LISTING 1

```
<xsl:variable name="SortedElements">
  <xsl:call-template name="mergeSingleNode">
    <xsl:with-param name="theSiblings"
      select="/dates/datesUnSorted/date"/>
    <xsl:with-param name="siblPosition" select="2"/>
    <xsl:with-param name="sortPad" select="$theBuffer2"/>
  </xsl:call-template>
</xsl:variable>
```

### LISTING 2

```
<!-- If the merge position is greater than 1 then return
the left half of the result -->
<xsl:if test="$mergePosition > 1">
  <xsl:value-of select="substring($sortPad, $sortLen + 1,
($mergePosition - 1) * $sortLen)"/>
</xsl:if>

<!-- Now, place insert the currently evaluated date -->
```

```

<xsl:value-of select="concat('|', $argNumber)"/>

<!-- Now, concatenate the right half of the dates -->
<xsl:if test="$mergePosition != $Last">
  <xsl:value-of select="substring($sortPad, $mergePosition
    * $sortLen + 1,
    $sortLen * ($Last - $mergePosition)) "/>
</xsl:if>

```

### LISTING 3

```

<xsl:call-template name="DisplayStringAsTable">
  <xsl:with-param name="strInput" select="$SortedElements"/>
  <xsl:with-param name="strColHeader" select="'Date'"/>
  <xsl:with-param name="strPageHeader" select=
    "'The top 5 dates'"/>
</xsl:call-template>

```

### LISTING 4

```

JeniDataB.xml
<?xml version="1.0"?>
<ProductFiles>
  <file href="ProductSalesNW.xml" />
  <file href="ProductSalesSW.xml" />
</ProductFiles>

```

```

JeniCombineB.xml
<xsl:template match="ProductFiles">
<allproducts>
  <xsl:for-each select="document(file/@href)">
    <xsl:copy-of select="*" />
  </xsl:for-each>
</allproducts>
</xsl:template>

```

### LISTING 5

```

AllProducts.xml
<?xml version="1.0"?>
<!DOCTYPE allproducts [
  <!NOTATION xml SYSTEM "">
  <!ENTITY resourceA SYSTEM "ProductSalesNW.xml" NDATA xml>
  <!ENTITY resourceB SYSTEM "ProductSalesSW.xml" NDATA xml>
]>
<allproducts>
  <get-data ref="resourceA"/>
  <get-data ref="resourceB"/>
</allproducts>

```

```

Template from AllProducts.xml
<xsl:template match="get-data">
  <xsl:apply-templates select="document
    ( unparsed-entity-uri( @ref ) )"/>
</xsl:template>

```

### LISTING 6

```

CombiningXML.asp
<%@LANGUAGE=VBSCRIPT%>
<%

Dim objXML, objXSL, objRoot, objNewXML, aXMLFiles, nCount

set objXML = Server.CreateObject("Microsoft.XMLDOM")
set objXSL = Server.CreateObject("Microsoft.XMLDOM")
set objNewXML = Server.CreateObject("Microsoft.XMLDOM")

objXML.load(Server.MapPath("AllProducts.xml"))
set objRoot = objXML.documentElement

aXMLFiles = split("ProductSalesSW.xml,
  ProductSalesNW.xml", ",")
For nCount = LBound(aXMLFiles) to UBound(aXMLFiles)
  objNewXML.load(Server.MapPath( aXMLFiles(nCount)))
  objRoot.appendChild objNewXML.documentElement
next

objXSL.load(Server.MapPath("AllProducts.xsl"))
objXML.transformNodeToObject objXSL, Response

%>

```

### LISTING 7

```

KurtData.xml
<?xml version="1.0"?>

```

```

<ProductFiles>
  <file href="ProductSalesNW.xml" />
  <file href="ProductSalesSW.xml" />
</ProductFiles>

```

```

KurtCombine.xml
<xsl:template match="/">

```

```

  <!--Create a variable which holds the names of the xml
    documents we are combining -->
  <xsl:variable name="doc.refs.1">
    <doc>ProductSalesSW.xml</doc>
    <doc>ProductSalesNW.xml</doc>
  </xsl:variable>

```

```

  <!-- Iterate through the variable and access the
    xml document using the document() object.
  Note: We are using the MSXML implementation of the
  node-set function to access the document as a nodeset. -->
  <xsl:for-each select="msxsl:node-set($doc.refs.1)//doc" >
    <!-- Display product information using our
      DisplayProducts.xml template -->
    <xsl:call-template name="DisplayProds" >
      <xsl:with-param name="ProductNodes"

```

```

select="document(.)"/>
    </xsl:call-template>
  </xsl:for-each>
</xsl:template>

```

```

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

```

```

  <xsl:template name="DisplayProds">
    <xsl:param name="ProductNodes" select="." />
    <h1>Product Listing for
    <xsl:value-of select="$ProductNodes//@region" /> region
  </h1>

```

```

  <!--Display the headings for each element as a table
    header -->
  <table border="1">
    <xsl:for-each select="$ProductNodes//product[1]/*" >
      <th><xsl:value-of select="name(.)"/></th>
    </xsl:for-each>

```

```

  <!--Display the data for each element as a table row -->
  <xsl:for-each select="$ProductNodes//product">
    <xsl:sort select="prodid" order="ascending" />
    <tr>
      <xsl:for-each select="*" >
        <td><xsl:value-of select="text()" /></td>
      </xsl:for-each>
    </tr>
  </xsl:for-each>
</table>

```

```

</xsl:template>

</xsl:stylesheet>

```

### LISTING 8

```

<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="AllProducts.xsl"?>
<!DOCTYPE website SYSTEM "products.dtd">
<allproducts xmlns:x="http://www.w3.org/1999/XML/xinclud">
<h1>Product listing for all regions</h1>

<h2>Southwest Region</h2>
  <x:include href="data/ProductSalesSW.xml#xpointer
    (/products/product)"/>

<h2>Northwest Region</h2>
  <x:include href="data/ProductSalesNW.xml#xpointer
    (/products/product)"/>
</allproducts >

```