Tip: How to combine documents with XSLT

Exploring XPath functions

Benoit Marchal

May 29, 2003

This tip explains how to write XSLT style sheets that process several documents. This is useful, among other things, for combining book chapters, merging a letter template and a list of addresses, creating tables of content that span several files, or -- following the tip example -- reusing photo descriptions in different galleries.

XSLT offers an attractive mix of flexibility and power. XSLT is useful not only for publishing Web sites, but also for transforming or manipulating XML documents. Now that an XSLT processor ships with the Java platform (through the javax.xml.transform package), you can't ignore it.

A recurring question from XSLT programmers is how should you deal with multiple documents? The Java API expects only two parameters: the source (input XML document) and the result (where to save the output). While this API is suitable for many applications, in some cases you need to combine several sources. Some examples include:

- A mail merge, such as a direct mail marketing campaign, where the stylesheet merges names and addresses from a customer file with a letter template.
- Converting code lists, such as product references. Country codes often require matching a catalogue file against a list of codes stored in a separate file.
- Combining individual documents for publication. For instance, when publishing a book you might want to merge the chapter files or -- as you'll see in the following section -- photos files.

An example

The following example illustrates how you might use XML and XSLT to merge multiple documents for publication. Every photo has two files: the photo itself (in JPEG format) and an XML description with the title, date, and location. The description files might look like Listing 1.
Tip: How to combine documents with XSLT

Listing 1. geneva.xml -- an XML description of a photo

```xml
<?xml version="1.0"?>
  <ph:title>The Jet d'Eau fountain</ph:title>
  <ph:location>Geneva</ph:location>
  <ph:date>April 2003</ph:date>
  <ph:description>
    The Jet d'Eau fountain is the most recognizable symbol of Geneva. The fountain reaches 140 meters (460 feet) high, roughly the same height as the Embassy Suites hotel in Times Square.
  </ph:description>
</ph:photo>
```

The markup in Listing 1 is simple. It would not take long to write a stylesheet that publishes it in HTML. A more interesting question is how do you create a gallery that combines Listing 1 with other photo descriptions such as those in Listing 2? (You will find even more descriptions in the downloadable code, which you'll find in Related topics.)

Listing 2. london.xml -- another XML description of a photo

```xml
<?xml version="1.0"?>
  <ph:title>Double-decker bus</ph:title>
  <ph:location>London</ph:location>
  <ph:date>October 2002</ph:date>
  <ph:description>
    An inescapable symbol of London, the double-decker bus is much taller than typical buses to carry many passengers through the city's overcrowded streets.
  </ph:description>
</ph:photo>
```

Before going any further, I want to show you a few definitions. The XML document you pass to the stylesheet through the Java API is called the main source. Other documents that the stylesheet loads are called secondary sources.

The document() function

Since the Java API accepts only one source, you have to load secondary sources from the stylesheet itself. This is done through the document() function, which takes a URI as a parameter. The function parses the document and returns a node set.

For completeness, document() also accepts a node-set as a parameter, in which case the function assumes each node is a URI. Finally, you can pass a combination of a string and a node-set as parameters. The function combines the two to decide which documents to load. Still, in most cases it is simpler to pass a single URI.

For example, the following XPath loads the geneva.xml document shown in Listing 1:

```
document('geneva.xml')
```

Because document() returns a node-set, you can insert it almost anywhere an XPath is valid. More specifically, you can add selectors to the path. For example, the following XPath loads the geneva.xml document and returns its title:
XPaths with the `document()` function can appear anywhere an XPath is legal, such as the `select` attribute for the `xsl:value-of` and `xsl:apply-templates` instructions. Therefore, to display the document title, you could write the following instruction:

```xml
```

Be careful when writing XPaths for secondary sources. It's easy to forget the root, but if you do, the XPath returns an empty node-set. For example, if I forget `ph:photo` (the root of the secondary document) in the XPath, I end up with the following XPath, which returns the empty node-set:

```xml
document('geneva.xml')/ph:title
```

## Combining documents

To summarize, when combining documents with XSLT, you want to start from the main source and load the secondary sources. One of the easiest solutions is to list the secondary sources in the main one. The main source looks like Listing 3 -- it simply lists the documents to combine (as well as providing a title for the collection).

### Listing 3. index.xml lists the secondary sources

```xml
<?xml version="1.0"?>
    <ph:title>City sights</ph:title>
    <ph:entry>geneva</ph:entry>
    <ph:entry>london</ph:entry>
    <ph:entry>paris</ph:entry>
    <ph:entry>roma</ph:entry>
</ph:index>
```

### Writing the main source

You can generate Listing 3 automatically (a small Java component goes through a directory and automatically generates it). XM, the publishing framework that I developed for my *Working XML* column (also on *developerWorks*) includes such a component.

Listing 4 is the stylesheet itself. It defines templates for the elements in the main source and the secondary sources. The link between the two is in the `ph:index` template, which parses the secondary sources. The `xsl:apply-templates` instruction means that the processor walks through the secondary source and applies templates as it goes along. In practice, it behaves as if the secondary sources were included in the main one.

### Listing 4. merge.xsl -- the stylesheet processes the documents listed in the main source

```xml
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:ph="http://ananas.org/2003/tips/photo">
```
The result of applying Listing 4 to Listing 3 is an HTML document with all the photos. One of the benefits of this approach is that you can add or remove photos to the gallery by editing the index.xml document. Furthermore, it's easy to share photos across different galleries. Say you want a second gallery with more photos of Geneva. You only need to create another list, similar to Listing 3, that points to these photos. You can include the description from Listing 1 in both galleries, but you don't have to duplicate it.

**Other applications**

The technique described in this tip has numerous applications. You can use the document() function to create indices or tables of content that span multiple files. I have also used it to retrieve link titles: Just follow the URI with the document() function and recover the document title. I have also used it to go through daily logs and generate a monthly report.
## Downloadable resources

<table>
<thead>
<tr>
<th>Description</th>
<th>Name</th>
<th>Size</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>x-tipcombxslt/x-combxmlcode.zip</td>
<td></td>
</tr>
</tbody>
</table>

Tip: How to combine documents with XSLT
Related topics

- Download the source code used in this article.
- Get an introduction to a component for creating main sources from directories automatically in "Wrapping up XM version 1" (developerWorks, October 2001).
- Find more XML resources on the developerWorks XML zone. For a complete list of XML tips to date, check out the tips summary page.
- Find out how you can become an IBM Certified Developer.

© Copyright IBM Corporation 2003
Trademark
(www.ibm.com/developerworks/ibm/trademarks/)