

# HACKING THE STYLER XSL-FO STYLESHEET

*PTC/USER WORLD EVENT 2008*

James Sulak  
Jones McClure Publishing  
1113 Vine St. Suite 240  
Houston, TX 77002  
713-335-8264  
jsulak@jonesmcclure.com

## INTRODUCTION

When you publish the same content on multiple platforms, ensuring that that content is presented consistently is important. That doesn't mean it's easy. Different platforms often use different, incompatible languages and file formats. Any method that allows you to write logic once and reuse it everywhere saves you time and effort, and more importantly, reduces the possibility for error.

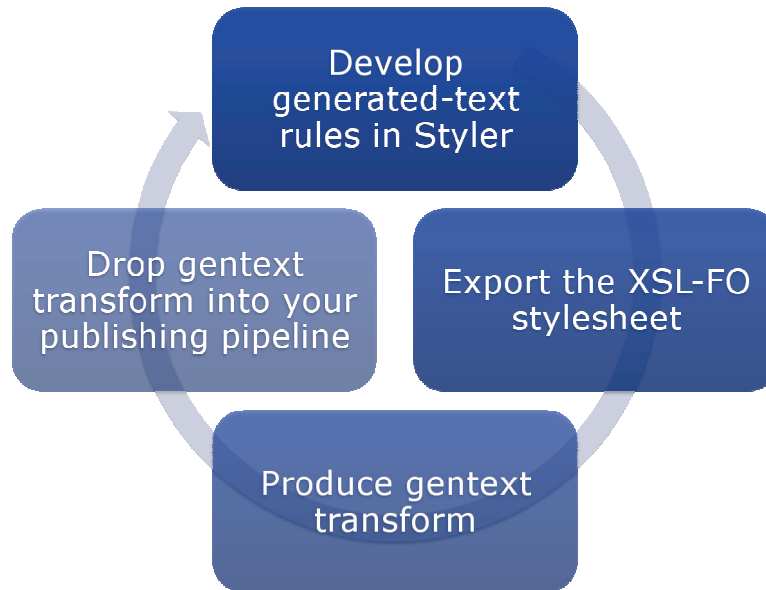
Arbortext Styler is an attempt to implement "develop once, use anywhere," and it's very good at it. But there are platforms that it simply doesn't support – such as Microsoft Word or PDFs through XML Professional Publisher.

But because Styler stylesheets are stored as XML and produce XML, the information in them can be manipulated with all the powerful XML tools you use with your own content. This presentation demonstrates how Jones McClure Publishing uses Styler to maintain its complicated generated-text rules for all its publishing platforms, including those beyond what Styler was designed for.

Specifically, it demonstrates how to write an intermediary transform that takes a Styler-exported XSL-FO stylesheet as input and outputs a generated-text transform. Every time you modify your generated-text rules in Styler, you can use the intermediary transform you wrote – without modification – to produce a new generated-text transform. The process is:

1. Develop generated-text rules in Styler, reviewing changes in editor view.
2. Export the stylesheet as an XSL-FO stylesheet.
3. Transform the XSL-FO stylesheet using an intermediary XSLT transform, producing a generated-text transform.
4. Run XML documents through the resulting generated-text transform as the first step in multiple publishing pipelines.
5. If necessary, modify generated-text rules in Styler, and repeat.

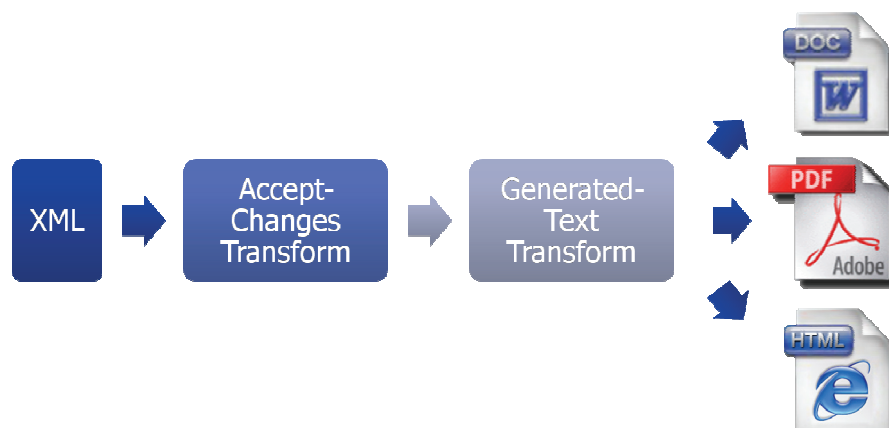
It can be thought of as a cycle:



This approach has several advantages. Developers can use the Styler stylesheets they already have instead of redeveloping the same generated-text logic for every publishing platform they use. And every time the Styler stylesheets are updated, the changes can be easily propagated to all output pipelines.

In other words, Styler can serve as a single point for all generated-text development, which is more efficient not only because it avoids redundancy, but because developers can take advantage of Styler's interface, which is often easier and more effective than other options, such as pure XSLT. Better, round-tripping is easy, because the generated text is inserted inside namespaced elements, which are trivial to remove from the result document using XSLT.

Once you create this generated-text transform, you can use it as a step in transforming your XML content into publishable file formats. For example, Jones McClure uses Ant builds with this basic form:



## OBJECTIVES

You will leave this session with a better understanding of XSLT and how to get started developing your own re-uses of Styler. Specifically, you will learn:

- The logic of the Styler-generated XSL-FO transform, and how multiple-mode and multiple-pass stylesheets work using EXSLT.
- XSLT techniques for transforming a transform.
- How to write an intermediary transform to change the Styler XSL-FO stylesheet into a generated-text transform.
- How to integrate the generated-text transform into your publishing pipelines.

## ABOUT THE SPEAKER

James Sulak is an Electronic Publishing Developer at Jones McClure Publishing, the largest independent law book publisher in Texas. He converts legacy content to XML; writes and maintains Arbortext customizations; creates new XML-based editorial processes; and develops stylesheets for multiple publishing platforms. To do all this, James uses XSLT 1.0 and 2.0, Ant, ACL, and Javascript. He can be contacted at [jsulak@jonesmcclure.com](mailto:jsulak@jonesmcclure.com).

## PRESENTATION

### DEVELOPING STYLE RULES

Most of your generated-text development should be done inside of the “Base (All Uses)” context. That ensures that what appears in Editor is the same as what appears in your other outputs. However, there are often times when you want to display something differently in Editor than you do in your published documents – for example, we place generated text around metadata to help editors, but we do not want that text output for our readers. In those cases, you can define rules in the “Print / PDF” context to override the gentext in the “Base” context. This works because when you export an XSL-FO stylesheet, the export process first tries to use the rules in “Print / PDF” context for each element, but if that is not available, it uses the “Base” context.

### THE STYLER-EXPORTED XSL-FO STYLESHEET

Although I use the FO stylesheet exclusively in this demonstration, the following techniques should also work with other exported stylesheets, such as the HTML stylesheet.

#### *SETTING UP YOUR STYLER STYLESHEET:*

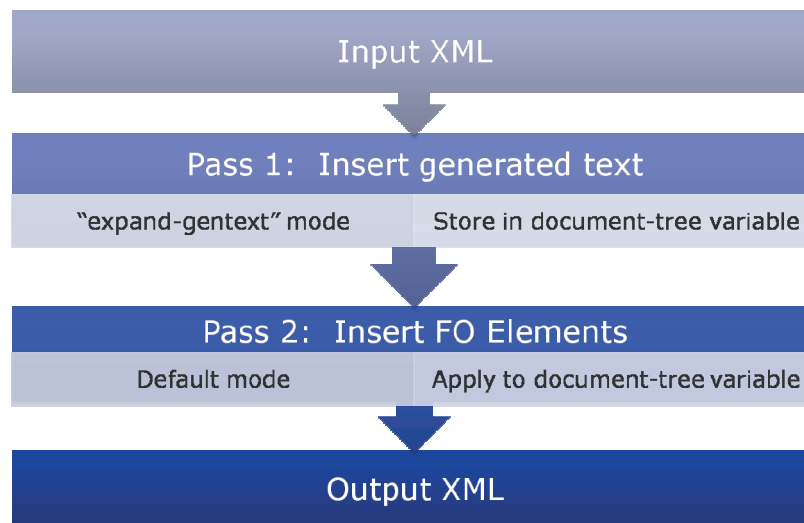
To export an XSL-FO stylesheet from Styler:

1. Make sure that your PDF output engine is set to XSL-FO (go to File > Properties, and select the XSL-FO radio button).
2. Go to File > Export > XSL-FO.

3. Choose a file location and click “Save.”

### *MULTIPLE-PASS STYLESHEETS*

The Styler-exported XSL-FO stylesheet is a multiple-pass transform. While most XSL transforms apply a set of templates to the context document and immediately output the result, a multiple-pass transform applies templates in one mode, saves the result in a variable, and then applies a second mode of templates to that variable and outputs the result. You can think of it as two transforms in one. This is how the Styler-exported XSL-FO stylesheet is structured:



In pure XSLT 1.0, it is not possible to apply templates to a set of nodes stored in a variable. For that, you need an extension to XSLT called EXSLT. This is the basic syntax (adopted and simplified from the Styler XSL-FO transform):

```
<xsl:template match="/" name="Expand-Gentext">
  <xsl:param name="document-tree">
    <xsl:apply-templates select="/" mode="expand-gentext"/>
  </xsl:param>
  <xsl:apply-templates select="exslt:node-set($document-tree)" />
</xsl:template>
```

Templates are first applied to the context document in the “expand-gentext” mode. The output of those templates, instead of going into an output document, is stored in the “document-tree” parameter. Then, the root template, using the `exslt:node-set()` function, applies the default-mode templates to this new “document” stored in that parameter. The output from those templates is sent to the final output document.

Note: In XSLT 2.0, this behavior works out of the box, with no need for the `exslt:node-set()` function.

### *TEMPLATES*

All element contexts defined in your Styler stylesheet translate into a set of templates in the exported XSL-FO transform. This includes all user formatting elements and Styler formatting elements.

The basic types of templates are:

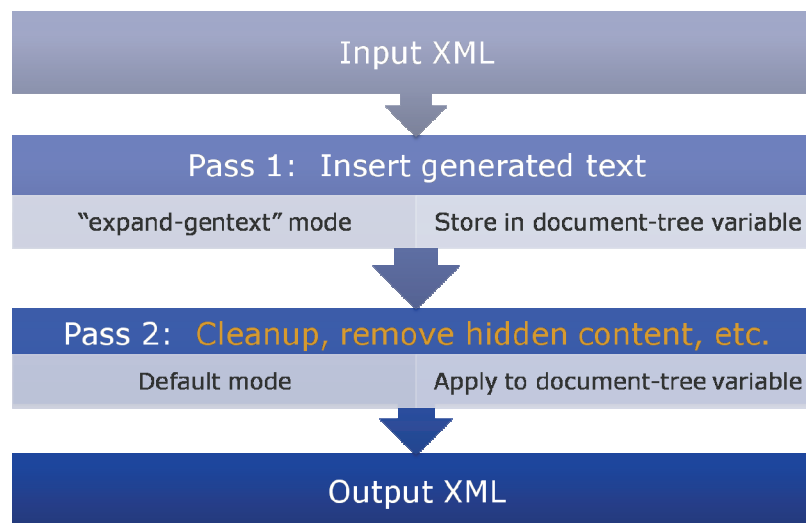
- **Templates for Context.** Default mode templates. These change document markup into FO markup, and are executed during the second pass. One of these templates is always present for a given element context.
- **Gentext Templates for Context.** “Expand-gentext” mode templates. These return your original XML along with any generated text enclosed in namespaced elements, and are executed during the first pass. One of these templates is always present for a given element context.
- **Numbering Templates for Context.** “Styler\_numbering” mode templates. These are usually called from “expand-gentext” templates, and return any numbering defined by Styler’s “Numbers and Bullets” section.

### NAMESPACE PREFIXES

- **\_sfe.** All generated text is enclosed in a `<_sfe:BeforeOrAfterText />` element.
- **\_ufe.** All user formatting elements use this namespace prefix.
- **\_gte.** This namespace prefix is used internally by the XSL-FO stylesheet to keep track of the status of generated text.
- **fo.** This namespace prefix is used for all FO elements.

### TRANSFORMING THE XSL-FO STYLESHEET

Since XSLT itself is XML, you can transform the XSL-FO stylesheet into a generated-text-only transform using XSLT. While it is possible to simply “turn off” the second (FO) pass setting the `skip-post-gentext-pass` parameter to “yes,” the better solution is to hijack that pass to do some additional processing, such as deleting hidden element contexts, resulting in a generated-text transform with this structure:



The transform `Arbortext_xsl_to_gentext_xsl.xslt` is attached to this document and included in this presentation's materials on the conference CD. Although this transform will work as-is on most document types, it is meant primarily as a demonstration. You will modify it before putting it to real use, at the very least changing the output document type.

The most important concepts you need to know to create this intermediary transform are (1) using `<xsl:namespace-alias />`, (2) removing templates that do not produce generated text, (3) removing all `fo:*` elements, and (4) inserting templates that delete "hidden" elements.

### *USING A NAMESPACE ALIAS*

The biggest obstacle to transforming XSLT is that the output namespace prefix is the same as that of the actual XSL instructions in your transform. If you use "xsl:" both in your XSL instructions and your output, your XSLT engine will not know the difference between the XSL instructions it should execute and those it should output, so your XSLT will not parse. That is, unless you use a namespace alias:

```
<xsl:namespace-alias stylesheet-prefix="xslAlt" result-prefix="xsl"/>
```

This instruction, which is placed inside of `<xsl:stylesheet />`, allows you to write your result markup in your transform using a substitute namespace prefix. Later, when the output document is created, the prefix that you actually want will be inserted in the alias's place. For example, this template matches the `<xsl:output />` element and modifies it to use a Jones McClure document type:

```
<xsl:template match="xsl:output">
  <xslAlt:output method="xml" doctype-public="-//JMP//DTD CodeBook
    v0.1//EN" doctype-system="codebook.dtd"/>
</xsl:template>
```

Note that you still use the original namespace prefix in your match expression.

### *REMOVE TEMPLATES THAT DO NOT PRODUCE GENERATED TEXT*

The most important change that the `arbortext_xsl_to_gentext_xsl.xslt` transform must make to the XSL-FO stylesheet is to remove all of the default-mode templates that turn your XML markup into FO layout markup. The best way to accomplish this is to create a template to match all `<xsl:template />` elements in the source document, and then create other templates that specifically match the elements you want to keep. For example:

```
<xsl:template match="xsl:template"/>

<xsl:template match="xsl:template[@mode or @name]">
  <xsl:copy>
    <xsl:apply-templates select="@*|node()" />
  </xsl:copy>
</xsl:template>
```

See lines 100-118 in `arbortext_xsl_to_gentext_xsl.xslt` for more details. This works because the second template, because of its predicate, implicitly has a higher priority than the first. Taking the place of all these templates will be an identity template that we insert into the output gentext transform (see lines 246-250).

### *REMOVE ALL FO ELEMENTS*

While removing the default-mode templates removes most of the FO elements, they also hide out in generated text and numbering elements. Removing them from the result transform is as easy as inserting a new template into the result transform (see lines 255-258).

This also points out another way to add content to the result document with the “xsl” namespace prefix. By using `<xsl:text disable-output-escaping="yes" />`, and using a CDATA (or unparsed character data) block, you can insert text directly into the result document without that text being parsed.

### *INSERT TEMPLATES TO REMOVE “HIDDEN” ELEMENTS*

In Styler’s “Font” tab, you can set the “Hidden” property to “yes” to suppress the output of an element’s content. That information is available in the XSL-FO stylesheet, but is hidden behind conditionals within templates. If an element is only hidden under certain conditions, our template must create a new template with a match condition derived from those conditions. See lines 58-97 to see how these templates work.

It is easy to modify your `arbortext_xsl_to_gentext_xsl.xslt` stylesheet to only delete certain hidden elements by modifying the match patterns in these templates – for example, if you want to preserve hidden metadata. It is also easy (depending on your final publishing platform) to modify those templates so that they only suppress the display of these elements and do not delete them – for example, surrounding them with `<?xpp px?>` and `<?xpp pa?>` processing instructions for XPP output.

## MORE INFORMATION

### **XML.com: Namespaces and XSLT Stylesheets**

<http://www.xml.com/pub/a/2001/04/04/trxml/>

### **Tip: Multi-pass XSLT**

<http://www.ibm.com/developerworks/xml/library/x-tipxslttmp.html>

**XSLT and XPath on the Edge**, by Jeni Tennison