MetaStylesheets

Introduction

Meta stylesheets are stylesheets that generate other stylesheets. This technique is usually applied when you have an XML language whose behaviour can be implemented using an XSLT. A stylesheet can then transform this language into it's XSLT implementation.

Essentially it's a code generation mechanism: you have a *little language* that declaratively describes what you want to happen, a code generator in XSLT (the *meta-stylesheet*) which transforms this language into the *implementation stylesheet*.

There are several examples of this:

- For details on creating XSLT stylesheets that create other stylesheets, see XML Reflection
- Schematron – an XML validation language implemented as XSLT, see Schematron - Validating XML using XSLT for a tutorial
- Web template languages – see the XML.com article Template Languages in XSLT for some relevant background

Overview

There are therefore two steps in using a meta-stylesheet:

1. Process your declarative processing description with the meta-stylesheet, generating the implementation stylesheet

2. Process your data with the implementation stylesheet, generating your output.

Here's how to do it in Cocoon. It's very easy!

Note: **It is recommended to use Xalan as opposed to XSLTC for the transformation.** Xalan is the default for Cocoon 2.0.4. For 2.1 it must be specified by using `type="xalan"` in the `<map:transform>` node.

How It's Done

```
<map:pipeline>
<map:match pattern="my.html">
  <map:generate src="my.xml"/>
  <map:transform type="xalan" src="cocoon:/implementation-stylesheet.xsl"/>
  <map:serialize/>
</map:match>
</map:pipeline>

<map:pipeline>
<map:match pattern="implementation-stylesheet.xsl">
  <map:generate src="process-description.xml"/>
  <map:transform src="meta-stylesheet.xsl"/>
  <map:serialize type="xml"/>
</map:match>
</map:pipeline>
```

Here we have two pipelines, each of which perform one steps outline above. The first pipeline does the actual work: it transforms our XML into HTML results using the implementation stylesheet.

The implementation stylesheet is automatically generated by the second pipeline, which reads a processing description, transforms it with the meta-stylesheet generating the desired implementation.

Because the implementation stylesheet is generated automatically the meta-stylesheet can take into account additional context, e.g. the request parameters, when generating the results. This makes for very flexible processing.

– LeighDodds

Issues

I *have* been able to get this working using the cocoon:/ protocol (using 2.0.3), but I found that the dynamic stylesheets didn't appear to be cached, which was a big performance hit. StephenNg

I also *have* been able to get this working using the cocoon:/ protocol (using 2.1.1), but the 'transformed' xsl is not able to find any 'included' stylesheets. (seems logical, but it is annoying. Is there a solution?)

Note

Always be aware of the fact that such a pipeline is not cached. E.g., if you have a time-consuming generator (like HTML which uses JTidy), split the pipeline into two separate ones. Instead of

```
<map:match pattern="*.html">
  <map:generate src="{1}.html"/>
  <map:transform src="cocoon:/{1}.xsl"/>
  <map:serialize/>
</map:match>
```

use

```
<map:match pattern="*.html">
  <map:generate src="cocoon:/{1}.xml"/>
  <map:transform src="cocoon:/{1}.xsl"/>
  <map:serialize/>
</map:match>

<map:match pattern="*.xml">
  <map:generate src="{1}.html"/>
  <map:serialize type="xml"/>
</map:match>
```

In the latter case, the "*.xml" pipeline is cached and the overall performance is much better.

– AndreasHartmann