StreamGenerator

Example

This example extracts the form control named `text` from the POSTed form data, and parses it as XML.

The POSTed data must come from a form (that is, it must have MIME content type of `application/x-www-form-urlencoded`).

```
<map:generate type="stream">
  <map:parameter name="form-name" value="text"/>
</map:generate>
```

The form-name parameter refers to the name of a HTTP-POST parameter, which can be a form control. For example, the following HTML form has such a form control:

```
<form action="URI-for-pipeline-with-streamgenerator" method="POST">
  <input name="text"/>
  ...
  <input type="submit"/>
</form>
```

If you try this and fill in some value in the input field, you will most likely get a `Content is not allowed in prolog` error. This is because, as stated above, the content of the form-field is parsed as XML. If you would type valid XML into the input field, the example would work. This limits the useability of the StreamGenerator for processing forms, but you should really use the RequestGenerator for that!

Example with CInclude

The following example shows how you can use the StreamGenerator with the eXist] transformer to make a 'remote transformer'. This could be used to let two Cocoon servers talk to each other, where one is an application server and the other is a database server using Cocoon (I have done this with [http://exist-db.org/).

On the 'server' side, you set up a pipeline with a StreamGenerator, like the one shown earlier, some fancy transformations, and the XML serializer at the end. On the 'client' side, you have a pipeline that generates a document that you would like to be processed by the server:

```
<map:generate src="test.xml"/>
<map:transform src="wrap_in_cinclude.xsl"/>
<map:transform type="cinclude"/> <!-- Process by server. -->
<map:serialize type="xml"/>
```

Before the document is sent to the server, it is wrapped in a cinclude element by the `wrap_in_cinclude.xsl` stylesheet:

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:cinclude="http://apache.org/cocoon/include/1.0"
>

<xsl:template match="/">
  <cinclude:includexml>
    <cinclude:src>uri-of-server-pipeline</cinclude:src>
    <cinclude:configuration>
      <cinclude:parameter>
        <cinclude:name>method</cinclude:name>
        <cinclude:value>POST</cinclude:value>
      </cinclude:parameter>
    </cinclude:configuration>
    <cinclude:parameters>
      <cinclude:parameter>
        <cinclude:name>text</cinclude:name>
        <cinclude:value>
          <xsl:copy-of select="."/> <!-- Copy the whole document. -->
        </cinclude:value>
      </cinclude:parameter>
    </cinclude:parameters>
  </cinclude:includexml>
</xsl:template>

</xsl:stylesheet>
```

The effect is that the document `test.xml` is processed by the server, and sent back to the client.

By aggregating several pipelines that start with a StreamGenerator, you can process multiple documents in one go. You will have to wrap them in `<cinclude:parameter>` elements with different {{<cinclude:name>}}s.

Note: In Cocoon 2.1.6 there is a bug in org.apache.cocoon.xml.XMLUtils which prevents cinclude from using the POST method. There is a fix, which was applied in 2.1.8.

Note: The cocoon: protocol cannot be used in `<cinclude:src>` , because `org.apache.cocoon.components.source.SourceUtil` limits is to using GET, and the StreamGenerator only accepts POST.