How to use the Paginator Transformer (2.1 legacy document)

Table of contents

Table of contents

Warning:
This document was copied as is from the Cocoon 2.1 documentation, but has not yet been fully reviewed or moved to its new home.

1. Overview

This How-To describes the how to use Cocoon's Paginator Transformer component. You can consider it a 'FilterTransformer' on pagination steroids. The Paginator Transformer filters specific data and counts pages as it transforms SAX events. It implements pagination rules based on easy-to-configure pagesheet documents.

2. Purpose

XSLT-based approaches to pagination are problematic. First of all, it's somewhat complex to define the necessary declarative logic in XSLT. Additionally, an XSLT solution is rarely reusable across different pagination use cases. These problems spurred the creation of the Paginator Transformer. You can quickly add pagination capabilities to your webapp once you have configured a simple few rules within a single configuration file, the pagesheet.

The Paginator Transformer works quite nicely for use cases involving a few tens of pages and, of course, for static generation of any number of pages. However, the Transformer must process an entire file before it can extract even a single page. Therefore, you are **strongly** advised against using it for books or other large documents on dynamic sites. Nevertheless, its output is cacheable. Thus, if the same page is requested, then the document will be reprocessed by the Transformer only when it has changed.

3. Intended Audience

Cocoon users who need pagination capabilities for their web documents. This includes frustrated users who are tired of implementing complex, XSLT-based approaches to pagination.

4. Prerequisites

Make sure you have the version 2.1 or greater of Cocoon. The PaginatorTransformer component source is located in the core area.

During the build process, the necessary configuration details for the PaginatorTransformer component are automatically copied to cocoon.xconf of cocoon.war. This means that you don't need to manually configure cocoon.xconf. However, if you are adding the paginator samples to Cocoon webapp that was **not** generated by the above build command, add the following snippet to your cocoon.xconf file, located in the WEB-INF directory of your deployed webapp.

Version 2.1:

```
<paginator class="org.apache.cocoon.transformation.pagination.Paginator"
role="org.apache.cocoon.transformation.pagination.Paginator" logger="core.paginator"/> />
```

Sample files, not directly related to this How-To, are also copied during the build process to Cocoon webapp at webapp/samples/paginator. You can access them in your web browser using the following URI:

http://localhost:8888/samples/paginator/

5. Steps

Let's start with a simple example.

5.1. Simple Example

Suppose you have an XML file, document.xml, as follows.

<?xml version="1.0"?> <images> <image /> <image /> <image /> <image /> <image /> <image /> <image /> </images>

First, you need to write a **pagesheet.** Just as a stylesheet contains instructions for an xslt processor, a pagesheet contains instructions for the paginator filter. Here is the pagesheet dtd.

<!ELEMENT pagesheet (items?, rules)*> <!ATTLIST pagesheet xmlns CDATA #IMPLIED> <!ELEMENT items (group)> <!ELEMENT group EMPTY > <!ATTLIST group name CDATA #IMPLIED element CDATA #IMPLIED > <!ELEMENT rules (link?, count?)*> <!ELEMENT link EMPTY > <!ATTLIST link type (unit | range) #REQUIRED num CDATA #REQUIRED > <!ELEMENT count EMPTY > <!ATTLIST count type (element | char) #REQUIRED num CDATA #REQUIRED name CDATA #IMPLIED namespace CDATA #IMPLIED >

Let's say you want to paginate document.xml content based on a rule of three <image> elements per page. Here's a sample pagesheet, images.xml, which does just that.

<?xml version="1.0"?> <pagesheet xmlns="http://apache.org/cocoon/paginate/1.0"> <rules> <count type="element" name="image" num="3" /> </rules> </pagesheet>

You process a source file through a pagesheet filter in a sitemap snippet like this:

<map:match pattern="page(*)"> <map:generate src="document.xml"/> <map:transform src="pagesheets/images.xml" type="paginator"> <map:parameter name="page" value="{1}"/> </map:transform> <map:serialize type="xml"/> </map:match>

Accessing the URI for page one, page(1) yields:

<?xml version="1.0" encoding="UTF-8" ?> <images xmlns:page="http://apache.org/cocoon/paginate/1.0"> <image /> <image /> <image /> <page:page current="1" total="3" current-uri="/cocoon/samples/paginator/page(1)" clean-uri="/cocoon/samples/paginator/page" /> </images>

Clearly the above XML could have been transformed into something more meaningful. Note that the transformer must process all pages to obtain the value of total. Currently, there is no way to avoid this.

5.2. Adding Navigation

Given the Paginator's a full-blown pagesheet language, there's even more we can accomplish, most importantly, navigation.

As an example, consider the following pagesheet, images2.xml.

<?xml version="1.0"?> <pagesheet xmlns="http://apache.org/cocoon/paginate/1.0"> <items> <group name="item" element="images" /> </items> <rules> <rules> <count type="element" name="image" num="3"/> <link type="unit" num="1"/> </rules> </rules> </pagesheet>

The pagesheet rules demonstrate that the transformer understands how the page was encoded in the given URI request, i.e., that parentheses surround the value of page. They also reveal that the transformer can provide navigation links to available pages, in this case, plus or minus one position.

DS: In the above paragraph, you say the transformer understand how the page was encoded. How? I don't see the evidence until the snippet produced below.

In your sitemap.xmap file, if you change the pagesheet source to images2.xml as follows:

<map:match pattern="page(*)"> <map:generate src="document.xml" /> <map:transform src="pagesheets/images2.xml" type="paginator"> <map:parameter name="page" value="{1}" /> </map:transform> <map:serialize type="xml" /> </map:match>

processing the same page(1) request yields the following (pretty-printed for this document):

<?xml version="1.0"?> <images xmlns:page="http://apache.org/cocoon/paginate/1.0"> <image /> <image /> <image /> <page:page current="1" total="3" current-uri="/cocoon/samples/paginator/page(1)" clean-uri="/cocoon/samples/paginator/page"> <page:link type="next" uri="/cocoon/samples/paginator/page(2)" page="2" /> </page:page> </images>

This result demonstrates:

- Page 0 does not exist, so no <page:link> is created for a previous page.
- Page 2 exists, so <page:link> is created, along with
  - a value of "next" for its type attribute (useful for visualization), and
  - a value of page(2) for its URI attribute (useful for linking without XSLT-specific logic)

Note that the URI is re-encoded using the same parentheses pattern, page(2).

Now, without changing anything, requesting page(2) yields the following (pretty-printed for this document):

<?xml version="1.0"?> <images xmlns:page="http://apache.org/cocoon/paginate/1.0"> <image /> <image /> <image /> <page:page current="2" total="3" current-uri="/cocoon/samples/paginator/page(2)" clean-uri="/cocoon/samples/paginator/page"> <page:link type="prev" uri="/cocoon/samples/paginator/page(1)" page="1" /> <page:link type="next" uri="/cocoon/samples/paginator/page(3)" page="3" /> </page:page> </images>

And requesting page(3) yields the following.

<?xml version="1.0"?> <images xmlns:page="http://apache.org/cocoon/paginate/1.0"> <image /> <page:page current="3" total="3" current-uri="/cocoon/samples/paginator/page(3)" clean-uri="/cocoon/samples/paginator/page"> <page:link type="prev" uri="/cocoon/samples/paginator/page(2)" page="2" /> </page:page> </images>

Note only one <image>. The original document, images.xml, only contained seven <image> elements: three for page one, three for page two, but only one for page three. Thus, the result here is the modulo (or remainder) of the division.

5.3. Real-Life Examples

Here are a few pagesheets examples which are a bit more complex.

5.3.1. DirectoryGenerator Pagination

Here's an example of paginating the contents of a directory using the DirectoryGenerator.

<?xml version="1.0"?> <pagesheet xmlns="http://apache.org/cocoon/paginate/1.0"> <rules> <count type="element" name="file" namespace="http://apache.org/cocoon/directory/2.0" num="16" /> <link type="unit" num="2" /> <link type="range" value="5" /> </rules> </pagesheet>

The rules state:

1. paginate 16 files per page

2. provide links to +/- 1 and +/- 2 pages (when available)
3. provide links to +/- 5 (when available)

So, suppose we have a directory with 300 files. If we request page 10, the generated page will be:

<?xml version="1.0"?> <dir:directory> <dir:file ... /> [other 15 dir:file] <page:page xmlns:page="http://apache.org/cocoon/paginate/1.0" current="10" total="19" current-uri="dir(10)" clean-uri="dir" > <page:range-link page="5" type="prev" uri="page(5)" /> <page:link page="8" type="prev" uri="page(8)" /> <page:link page="9" type="prev" uri="page(9)" /> <page:link page="11" type="next" uri="page(11)" /> <page:link page="12" type="next" uri="page(12)" /> <page:range-link page="15" type="next" uri="page(15)" /> </page:page> </dir:directory>

5.3.2. Asymmetric pagination

We also have the ability to indicate different rules for each page, for example:

<?xml version="1.0"?> <pagesheet xmlns="http://apache.org/cocoon/paginate/1.0"> <rules page="1"> <count type="element" name="b" num="5" /> <link type="unit" num="1" /> </rules> <rules> <count type="element" name="b" num="10" /> <link type="unit" num="2" /> </rules> </pagesheet>

5.3.3. Count types

The Paginator Transformer was designed to count. However, it's up to you to define what needs to be counted, either XML elements or characters (not yet implemented). By supplying values to the attributes of <count> in the pagesheet, you can specify exactly what needs to be counted.

The <count> element has two required and two optional attributes. The required attributes are:

- **type** the method of counting the paginator should perform, either elements or characters. When element is specified, the transformer counts startElement() SAX events. When chars is specified (currently not implemented), the transformer counts the primitive data type char.
- **num** a number which how many times counted item (element or chars) must be present within the transformed page.

Optional attributes (when type="element" is specified) are:

- **name** the name of the element, without any namespace prefix
- **namespace** the URI of the namespace. If not specified, the default namespace is used.

6. Improving the Paginator Transformer

The PaginatorTransformer was developed, initially, to paginate a directory listing. It works great when it paginates by counting elements, particularly elements which contain similar amounts of content to be displayed on pages. With documents, for example, it could paginate by counting sections or subsections. However, bear in mind that this approach does not always guarantee visually-balanced web pages.

6.1. Nested Pagination

Furthermore, simply counting elements is not always simple. Consider the following:

<?xml version="1.0"?> <a> <a> <a>

Let's say you want to paginate using one per page. What should the transformed pages look like? Here's a few possible outcomes. Which one is the best?

6.1.1. Page 1

<?xml version="1.0"?> <a> <a> <a/>

6.1.2. Page 2

<?xml version="1.0"?> <a> <a> <a/>

6.1.3. Page 3

<?xml version="1.0"?> <a> <a> <a>

It appears the current code is buggy somewhere. With deep nesting as in this example, some SAX events are lost. This creates a non-well-formed SAX stream which chokes subsequent transformers, such as XSLT, which may be sensitive to well-formedness.

Does the above might look like a mental exercise to you? Perhaps, but consider the structure of Cocoon Project's Document DTD 1.1. which includes nested <section> elements. Similar problems will emerge when paginating these documents based on this dtd. It's isn't clear whether the solution adopted above is meaningful or not for a real-world pagination. Suggestions on this are welcome.

6.2. Character-based Pagination

Given the need to visually balance pages, a counting method for characters was added, even though it isn't implemented yet. Counting by characters is especially difficult when you think about the algorithms that perform chunking.

Assume you have a document like this:

this is some text that happens to be chuncked ^

Suppose that paginating by counting the chars results in a chunking point indicated by the caret above (between the letters u and n). Ending a page at that position results in XML that is not well-formed as well as truncated words. Even if you find a way to provide well-formed XML, you still must deal with word-break issues. Therefore, we need a way to produce well-formed XML by continuing until the first 'block-level' element is encountered, for example, in this case. However, this means that the pagesheet must contain a list of such 'block-delimiting' elements. Currently, the Pagesheet parser and object model does **not** support this notion.

Conclusion? Pagination at the char level is not trivial and will require a little bit of additional work on the transformer.

6.3. Other Improvements

One possible way to improve the concept is to count by XPath results. For example, you may want to count <section> elements included in other <section> elements. Another way to improve the design is to allow booleans to be used within counting rules. For example, you could count <session> AND <chapter> elements. Most likely, XPath will help here as well.

7. Comments

Got an idea how to improve the Paginator Transformer? Post it to the cocoon-dev mailing list. Care to comment on this How-To? Help keep this document relevant by passing along any constructive feedback to the cocoon-docs mailing list.

8. Revisions

06-06-02: Content originally posted to cocoon-dev by Stefano Mazzocchi.

06-26-02: Edited and structured by Diana Shannon. Scratchpad samples also added.

06-27-02: Scratchpad samples revised by Stefano Mazzocchi.

1. Comments

add your comments