

WAPChannel Tutorial

Bernhard Huber, berni_huber@a1.net

1 WAPChannel Tutorial

This document describes site reconstruction need for WAP enabling this site. By starting this site I intended to publish the content via WAP; but some problems always pop up late.

2 Publishing WML

The first step was creating a new stylesheet `site2wml.xsl` transforming the xml documents from `document.xml` to `wml`. It works analogous to `site2html.xsl`.

Next you need to enhance the `sitemap.xmap` handling `wml` pages. Creating the site page by aggregation, and transforming the site page to `wml` using the create `site2wml.xsl`.

The next step was testing and fixing WML coding errors in the `site2wml.xsl`. Finally I uploaded the `wml` pages to a server, created a Link <http://members.a1.net/csscolumnsite/index.wml> on my WAP handset.

3 Sitemap WML Changes

The sitemap changes for serving *WML* files in several areas

- Adding a *WML* serializer.
- Adding a pipeline handling URIs having *WML* extension. including the Paginator transformer.

Sitemap snippet declaring the *WML* serializer:

```
<map:serializers...
  <map:serializer name="wml"
    src="org.apache.cocoon.serialization.XMLSerializer"
    mime-type="text/vnd.wap.wml"
    logger="sitemap.serializer.wml">
    <doctype-public>-//WAPFORUM//DTD WML 1.1//EN</doctype-public>
  <doctype-system>http://www.wapforum.org/DTD/wml_1.1.xml</doctype-system>
  <encoding>iso8859-1</encoding>
  <omit-xml-declaration>no</omit-xml-declaration>
</map:serializer>
```

...

Sitemap snippet defining the *WML* pipeline. It include already the later described Paginator transformer.

```
<map:pipelines...
  <!-- handle paginated wml pages
  -->
  <map:match pattern="*(*)\.wml">
    <map:act type="link-translator-map" src="{1}\.wml"/>

    <map:call resource="show-page">
      <map:parameter name="page" value="{1}"/>
    </map:call>

    <map:transform type="paginate"
      src="pagesheets/wmlpages.xml">
      <map:parameter name="page" value="{2}"/>
    </map:transform>

    <map:transform src="stylesheets/site2wml.xsl"/>
    <map:serialize type="wml"/>
  </map:match>

  <!-- handle non paginated wml pages
  -->
  <map:match pattern="*\\.wml">
    <map:act type="link-translator-map" src="{1}\.wml"/>

    <map:call resource="show-page">
      <map:parameter name="page" value="{1}"/>
    </map:call>

    <map:transform type="paginate"
      src="pagesheets/wmlpages.xml">
      <map:parameter name="page" value="1"/>
    </map:transform>

    <map:transform src="stylesheets/site2wml.xsl"/>
    <map:serialize type="wml"/>
  </map:match>

</map:pipeline>
...
```

Setting up the sitemap and using the `site2wml.xsl` stylesheet. This site is *WML* publishing enabled.

4 Problems

I experienced following problems

- **Page is too big** The site pages are too big, and cannot be displayed on my WAP handset.

- The links of the site all goes to resources, having extension `.html`. Thus clicking a link fails, too.

The first problem is fixed by using the Paginator transformer, available in the scratch pad area of Cocoon since version 2.1.

The second problem is fixed by introducing a semantic linking schema.

5 Paginator Transformer

As the single wml files are too large, they must get split up, but I don't want to split up the xml documents, nor splitting up the html pages, too. Thus I add the Paginator transformer in the sitemap for the wml pages, only. Well, for easy debugging, and startup problems I add the Paginator transformer for the html pages, too.

The [Paginator description](#) is very helpful, and the paginator samples in the scratch pad area, finally have helped me to update the `site2wml.xsl` for displaying the links for page scrolling.

Beside enhancing the stylesheet, the sitemap needs some enhancements, too. The Paginator transformers uses a link encoding schema by appending (*N*) to the original URI; *N* stands for the page number. Thus the second page of `index.wml` becomes `index(2).wml`.

Finally for the Paginator transformer configuration the new directory `pagesheets` holds the paginator configuration files.

In short the paginator configuration specifies

- Upon which element pagination is triggered
- What links are added to the page in the `page:page` fragment added by the Paginator transformer.

The paginator links introduces some problems when I generated the static wml files. As the paginator adds the sevlet path I had to fix some bugs in the *Cocoon Ant* invocation, and enhancing the `Constants.LINK_OBJECT` link mapping.

6 Semantic linking

Before implementing the WAP channel I used links in the xml document like `<link href="index.html">`. The simple semantic linking for this site I changed the links to links like `<link href="page:index">`.

6.1 Page Schema

Introducing the *page* schema, there has to be some information about what's the physical

link, and who will do this link rewriting.

The requested URI itself holds the information I needed. Thus the extension of the requested URI determines the physical link. You remove the *page* schema, and append the extension of the requested URI, and that's the physical URI.

Cocoon has some nice - a bit hidden feature - of link rewriting. The *LinkTransformer* is automatically invoked, if an entry *Constants.LINK_OBJECT* exists in the `objectModel`. This entry may specify a simple `HashMap` having key entries for the original link, and an associated value for the rewritten link value.

Finally I ended up by writing a `LinkTranslatorMapAction`. Creating a special hash map overriding the `get()` for handling the *page* schema.

6.2 Cocoon Ant Rewriting

As noted above beside the *page* schema mapping, I add rewriting of servlet-path relative links. Links of the form `userhomes/~columnsite/faq(2).wml` were working fine in the servlet environment of Cocoon, but in the *Cocoon Ant* environment these links were resolved to `userhomes/~columnsite/userhomes/~columnsite/faq(2).wml`.

Thus I enhanced the `LinkTranslatorMapAction`. It removes the `ServletPath` prefix of any link it should lookup.

7 More WAP styling

The column-section separation was very helpful. The `mid-col-1` section becomes the first card of the WML deck. The other sections, ie `top-col-1`, `mid-col-1`, `mid-col-3`, were part of the second WML card. This way the WML user sees the "main content" immediately, links and additional information is moved to the second card and is immediately available, but does not pollute the "main content" WML screen.

Just to be sure, and to avoid the *Page too big* error, I limited the size of the `<p>`; showing only a limited prefix of the paragraph content.

8 Summary

It was quite some work enabling this site for WAP, but it helped me to understand some cocoon features

- `LinkTransformer` - for rewriting links automatically
- `PaginatorTransformer` - for splitting up big pages
- XSLT, WML brush up

WAPChannel Tutorial

The WAPChannel experience showed me that Cocoon is powerful and great tool for mutli-channel web webpublishing.