Locationmaps

Table of contents

Copyright © 2002 The Apache Software Foundation. All rights reserved.

1. About Locationmaps

A locationmap defines a mapping from requests to location strings.

It was conceived to:

- Provide a more powerful means for semantic linking.
- Enable Forrest with a standard configuration override mechanism.
- Decouple the conceptual source space used by Cocoon from the concrete source space, so that a change in the concrete sources does not impact on the sitemap.

In other words, the locationmap enables content to be retrieved from a location that is defined in a locationmap file (located at `PROJECT_HOME/src/documentation/content/locationmap.xml` file. The advantage of this is that the URL seen by the user need bear no relation to the location of the source document, thus Forrest can separate the client URL space from the source document URL space. Thus, using the locationmap it is possible to pull together documents from many different locations into a single uniform site.

In addition, since the user URL space is now not connected to the source URL space it is possible to move source documents without breaking any existing user links.

The syntax of a locationmap resembles that of the sitemap, in that it also makes use of Matchers and Selectors to traverse a tree of nodes towards a leaf. In the case of the locationmap however the leaf does not identify a pipeline, but instead identifies a location string.

Apache Forrest looks in the standard location for the source file first (by default `PROJECT_HOME/src/documentation/content/...`), if a file is found in this location then the locationmap is not consulted. However, if one is not found then the locationmap is used to resolve the source file. The locationmap is resolved via the core sitemap, this means that you can generate it dynamically if you so wish. Simply add a match that looks something like this to your projects sitemap:

```
<map:match pattern="locationmap-project.xml">
  <map:generate src="..."/>
  <map:transform src="..."/>
  <map:serialize type="xml"/>
</map:match>
```

2. Naming Convention

For those that are familiar with name resolution servers or the Handles Service, it might be easier to think of the locationmap as a name resolution module or sort of a handle resolution module that it accepts "names" or whatever you desire to call these "hints" and returns the location.

The thought is that by using hints that look a little like a file name it disguises what locationmaps are really doing for us. By using URN-style names, we are truly disassociating the name/hint from the physical location.

For example, here is a locationmap entry based purely on filename:

Copyright © 2002 The Apache Software Foundation. All rights reserved.

```
<map:transform src="{lm:html-to-document.xsl}"/>
```

and here is that same entry using a "name" style. One implies a certain physical location, whereas the one below is truly a name that needs to be resolved to a physical location.

```
<map:transform src="{lm:transform.html.document}"/>
```

Locationmaps are also used by plugins, and your project can also have its own locationmap.

Where the resource is provided by a plugin rather than Forrest itself, this is prefixed with the last part of the plugin name. For example:

```
<map:transform src="{lm:projectInfo.transform.changes.document}"/>
```

The above match means look in the projectInfo plugin for a transformer stylesheet with filename changes-to-document.xsl

The naming convention is essentially one of:

```
[PLUGIN_NAME.]resource-type(dot)from-format(dot)to-format
```

or

```
[PLUGIN_NAME.]resource-type(dot)type(dot)name
```

Examples of these:

```
transform.xthml2.html
graphic.png.project-logo
projectInfo.transform.changes.rss
```

3. Explanation of Locationmap processing

Some specific examples will explain. Please follow the relevant source files.

The document Cocoon sitemaps explained (better understand that document before trying to understand locationmaps) has two specific transformers which use locationmap references. The first one is: `<map:transform src="{lm:transform.linkmap.document}"/>`

The Locationmap component first consults the primary locationmap `$FORREST_HOME/main/webapp/locationmap.xml` file. This first mounts any project locationmap if available, then the locationmaps from plugins, then the rest of the core locationmaps in the `$FORREST_HOME/main/webapp/` directory. As with sitemaps, the first match wins. So matches in your project locationmap have precedence, then plugins, then core.

So let us get back to our specifc example, "`lm:transform.linkmap.document`". The "lm:" part means use the locationmap protocol. There is no specific match for this in your project locationmap, nor in any of the plugin locationmaps, so this falls through to the core locationmaps.

Copyright © 2002 The Apache Software Foundation. All rights reserved.

However, you will not find any specific match for this in the core locationmaps, so what is happening?

See the last match in `$FORREST_HOME/main/webapp/locationmap-transforms.xml` file. This a catchall matcher for any reference starting with "`transform.`"

```
    <match pattern="transform.*.*">
      <select>
        ...
        ...
        <location src="{forrest:forrest.stylesheets}/{1}-to-{2}.xsl"/>
      </select>
    </match>
```

As you know from your understanding of Cocoon sitemaps, the first asterisk yields "linkmap" and the second asterisk yields "document". So, ignoring the other possible locations in this group which look in the various skins stylesheet directories (see locationmap-transforms.xml source), it will finally resolve to that last possible location to look for a stylesheet called `linkmap-to-document.xsl` in the core stylesheets directory `$FORREST_HOME/main/webapp/resources/stylesheets/`

That explains the magic of the locationmap naming convention.

4. Multiple Location Selectors

You can define multiple possble locations for a file in the locationmap with the following code:

```
<match pattern="tabs.xml">
  <select type="exists">
    <location src="{properties:content.xdocs}tabs1.xml"/>
    <location src="{properties:content.xdocs}tabs2.xml"/>
  </select>
</match>
```

Each location will be tested in turn, if the file exists then it will be returned as a match, otherwise testing will continue.

5. Other Locationmap examples

5.1. Retrieving an XDoc via HTTP

Normally files are generated from `{properties:content.xdocs}`. Using the Locationmap it is possible to make these files come from elsewhere. This is useful if you want to pull files from different directory structures, or even remote repositories. For example, the following location match will match any request for a document below "remote." and will retrieve the source file from the Apache Forrest SVN repository (directly from the ASF's SVN webserver). This is an ideal way to ensure that your published docs are always up-to-date.

```
 <match pattern="project.remote.**.xml">
   <location
src="http://svn.apache.org/repos/asf/forrest/trunk/site-author/content/xdocs/{1}.xml"
/>
```


Copyright © 2002 The Apache Software Foundation. All rights reserved.

```
 </match>
```

Note that because this is a wildcard matcher you can request any page from the svn server simply by requesting `/remote.PATH/TO/FILE/FILENAME.html`. In addition, we can request any other output format available via Forrest plugins.

When including resources from remote repositories one has to be careful about things like `site` and `ext` linking. If the targets are not defined in the local `site.xml` file then these links will be broken.

Warning:

Because of the above limitation many of the links in the page generated from the above example are broken.

5.2. Retrieving HTML from a CMS

Using the locationmap you can use Forrest to retrieve data from a Content Management System (CMS), wither local or remote. As an example we will consider Apache Lenya.

Apache Lenya is a Java Open-Source Content Management System based on open standards such as XML and XSLT and the Apache Software Stack, in particular the XML publishing framework Apache Cocoon.

The following locationmap matcher will instruct Forrest to retrieve content from `http://lenya.zones.apache.org:8888/default/live/*.html?raw=true`, whenever a local URL of `lenya/**` is encountered.

```
 <match pattern="lenya/**.xml">
   <location
src="http://lenya.zones.apache.org:8888/default/live/{1}.html?raw=true" />
 </match>
```

However, since the source returned by this match is HTML and not XDoc we must convert this to our internal XDoc format before we can use it. We do this by adding the match below to our project's `sitemap.xmap` file.

```
<map:match pattern="lenya/**.xml">
  <map:generate type="html" src="{lm:{0}}" />
  <map:transform src="{forrest:forrest.stylesheets}/html-to-document.xsl" />
  <map:serialize type="xml"/>
</map:match>
```

Since this snippet uses the HTML generator you must also ensure that your sitemap has the HTML generator component defined. That is, your sitemap must also include:

```
<map:components>
  <map:generators default="file">
    <map:generator name="html"
      src="org.apache.cocoon.generation.HTMLGenerator">
      <jtidy-config>WEB-INF/jtidy.properties</jtidy-config>
    </map:generator>
  </map:generators>
</map:components>
```

Since the HTML generator uses JTidy we need to make available a JTidy configuration file. This is placed in `PROJECT_HOME/src/documentation/WEB-INF/jtidy.properties` (the location can be changed in the above sitemap snippet). A sample config file is given below:

Copyright © 2002 The Apache Software Foundation. All rights reserved.

```
indent=yes
indent-spaces=8
wrap=72
markup=no
output-xml=no
input-xml=no
show-warnings=yes
numeric-entities=yes
quote-marks=yes
quote-nbsp=yes
quote-ampersand=yes
break-before-br=yes
uppercase-tags=no
uppercase-attributes=no
char-encoding=latin1
```

Note:

This requirement to add items to your project sitemap will be removed in a future version either by Lenya outputting XDoc, or by Forrest switching to using XHTML as its internal format, or through the development of a plugin for Lenya that will include the necessary processing (whichever comes first).

Warning:

This demo is an example only, it does not fully work at this time. For example, absolute URLs in the source document need to be rewritten to ensure that they are matched by the locationmap.

5.3. Source Resolving

As well as being able to use the locationmap as a parameter in the sitemap, it is also possible to use it as a psuedo-protocol within resources processed by Forrest. For example, you can use it to import an XSL provided by one plugin within another plugin:

```
        <xsl:import src="lm://OOo.transform.write.xdoc"/>
```

5.4. Link Rewriting

The locationmap can be used to rewrite URLs when the page is generated. For example, when the locationmap has:

```
<match pattern="rewriteDemo/**">
  <location src="http://www.domain.org/{1}.xml" />
</match>
```

With the above match in the locationmap, a link which has `href="lm:rewriteDemo/index"` will be rewritten to an offsite address at `domain.org`

5.5. Debugging Locationmaps

Debugging the locationmap can be difficult because Cocoons error messages no longer provide meaningful data. We hope to improve this over time, in the meantime we recommend that you increase the log level of the locationmap logger. To do this edit the following line in $FORREST_HOME/main/webapp/WEB-INF/logkit.conf:

Copyright © 2002 The Apache Software Foundation. All rights reserved.

```
<category name="modules.mapper.lm" log-level="INFO">
```

For example, you could change the log level to "DEBUG".

Output from this logger can be found in $PROJECT_HOME/build/webapp/WEB-INF/logs/locationmap.log

You should not run production systems with this logger set higher than "INFO as each request can generate large amounts of log information.

Copyright © 2002 The Apache Software Foundation. All rights reserved.