JaxeCocoon

Jaxe and Cocoon

Introduction

This document shows you how Cocoon can start Jaxe as a Java Web Start application. In this way your users can edit XML files that are part of the Cocoon site. Think about a poor man's CMS.

Jaxe is a Java XML editor. See http://jaxe.sourceforge.net/

It does validation "on the fly", and gives hints to your user what he/she is allowed to enter according to the W3C schema describing your file. It shines when your users have to enter large chunks of text. Many free XML editors are attribute-oriented and not suited for entering large amounts of content.

I modified Jaxe so you can use it as a Java Web Start application. Jaws (Java Web Start) is a protocol that allows "thin client" apps on the desktop. It is a little bit like applets. But your browser is no longer involved. See http://java.sun.com/j2se/1.4.2/docs/guide/jws/developersguide/contents.html

The minimal modification to Jaxe is easy, you have to read/write from/to URL's instead of File's.

So the scenario is like this

1. The user clicks on the xml link he wants to edit.
2. The first time a user clicks on this link, the Jaxe editor is downloaded and cached on his pc.
3. Jaxe is started and it downloads the xml file from Cocoon.
4. When saving the file, it is uploaded and saved by Cocoon with the Stream generator and the write-source transformer

Steps

These are the steps involved. Only snippets of code are shown below. You will find the complete files in the attachment.

1. If you want to use Jaws, you have to make sure your web server understands the JNLP mime type. For Apache you have to add these line to your conf/mime.types configuration:
```
application/x-java-jnlp-file JNLP
```
2. Create a webpage with some links on it that trigger flowscript. Flowscript should find out which page the user wants to edit. In my example this is defined by the variables branch_id, model_id, and module_id. These variables are passed to the jnlp file which is generated by the jx generator. So we call the jnlp page from flowscript:
```
cocoon.sendPage("jaws/jaxe.jnlp", { "title" : "Jaxe for Cocoon", "branch": branch_id, "model": model_id,
"module": module_id} );
```
3. And the page is matched by this piece of the sitemap:
```
  <map:match pattern="jaws/*.jnlp">
    <map:generate type="jx" src="jaws/{1}.jnlp"/>
    <map:serialize type="jnlp"/>
  </map:match>
```
4. In order for 3) to work, we have to define a serializer with the correct mime type at the top of the pipeline:
```
 <map:components>
   <map:serializers default="html">
                 <map:serializer name="jnlp"
                  src="org.apache.cocoon.serialization.XMLSerializer"
                  mime-type="application/x-java-jnlp-file">
                </map:serializer>
   </map:serializers>
 </map:components>
```
5. The jnlp file is a jx template. In the snippet below you can see that Jaxe is started with two arguments: the jaxe configuration file (should be included in a resources.jar) and the file the user wants to edit (with parameters that will be substituted):
```
  <application-desc main-class="jaxe.Jaxe">
          <argument>config/Module_nl_Jaxe_cfg.xml</argument>
          <argument>http://localhost:8080/cocoon/dg/basisrie/read/main/${branch}/${model}/modules
/${module}.xml</argument>
  </application-desc>
```

6. Jaxe loads the .xml file from Cocoon. This is matched by this piece of the sitemap:

```
<map:match pattern="read/**">
  <map:generate type="jx" src="checkout/{1}"/>
  <map:transform src="stylesheets/simple.xsl"/>
  <map:serialize type="xml"/>
</map:match>
```

7. When Jaxe wants to save a file, it posts the content to the web server. Before doing that, Jaxe replaces the /read in the URL into a /write. So this is matched by another piece of the pipeline:

```
<!-- errors in the syntax are not reported -->
<map:match pattern="write/**">
 <map:generate type="stream"/>
      <map:transform src="stylesheets/doc2write.xsl">
        <map:parameter name="file" value="checkout/{1}"/>
      </map:transform>
      <map:transform type="write-source"/>
      <map:transform src="stylesheets/writeresult2simple.xsl"/>
 <map:serialize type="text"/>
</map:match>
```

That's all. Download the Zip file and have a look at the full code.

Remarks

1. At the start of this document I said that the modifications to Jaxe to make this work are easy. That was correct. It is easy to load/save from/to a URL instead of a File. But Jaxe has a lot of features that may trigger local disk access. Think about "Save as...". This may clash with your security model. You have three options.
   a. Leave it as they are. Depending on your configuration your user may or may not get an "access denied" error when clicking "Save as..". (In some cases nothing happens at all, it is silently ignored).
   b. Strip out all this functionality. Look for the word "fichier" in the sources. BTW, did I tell you that the sources of Jaxe are in the French language 🙂 ? This option has the advantage that you can also remove some jars from the download. E.g. Xalan. 1 Mb less in the initial download...
   c. Modify the sources so you can work in "hybrid" mode. So your user can both open/save from Cocoon and open/save on his local hard disk. This may appear the best solution. But is has two drawbacks: 1) You have to do some serious modifications to the Jaxe source and 2) you have to sign all your jars. When the user starts Jaxe, he will be annoyed with a question like: "Do you trust Hugo Burm". Now that's an easy question.
   The security model is explained in the Jaws docs. For local access to work, you have to uncomment some lines in the jnlp file:

   ```
   <!--
     <security>
         <all-permissions/>
     </security>
   -->
   ```

   In the zip file in the attachment, I went for option 3). So the Jaxe source is modified (I started from Jaxe 1.9.2), and all jar's are signed.
2. More on the security issue: in the example anyone can upload to this URL and replace the content. A real world application needs user management.
3. The Jaws docs show you a lot of code that is intended to find out what OS and what browser you are using, find out which version of Java you are using, find out whether Jaws is installed, an do an automatic install of Java 1.4.2 or the Jaws package if necessary. All these code did not work for me. And apart from that, I do not like the code (e.g, if it finds out that you are running Windows, it starts an Active-X control to find more details about your system.). So the simple solution is: tell your users they have to install the JRE 1.4.2 or better in order to make this work.
4. The Jaxe config dir is in the resources.jar.
5. The Jaxe sources are GPL.

Download the example: JaxeCocoon20040906.zip