BeginnerSimpleWebappOrganisation

Organising your development (cookbook)

- TARGET-AUDIENCE: ***beginner*** advanced expert
- COCOON-RELEASES: 2.0.3, 2.0.4
- DOCUMENT-STATUS: ***draft*** reviewed released

What you will get from this page

I will show you how you can setup a cleanly separated development environment with minimal modifications of the existing Cocoon infrastructure. You will end up with a hierarchy of folders all completely under your control with only one modification in the Cocoon standard distribution.

Your basic skills

- You have basic knowledge about XML
- You know how to access a command line shell to your OS

Technical prerequisites

- You need a cleanly installed version of Cocoon.

If you don't have access to Cocoon right now, you can just install it on your local computer. Please refer to BeginnerInstallation for further information.

Links to other information sources

...

The problem: Organisational complexity

Once you have installed Cocoon you will want to start doing something valuable with the system. The very first encounter with Cocoon will take place on the Cocoon sitemap, which happens to be a file right in the root directory (the $cocoon_root) of the Cocoon webapplication: `sitemap.xmap`.

As you will read elsewhere, the `sitemap.xmap` is the heart of Cocoon and, in order to get Cocoon doing something for you, you will first have to modify this file. Then you open the file and first get overwhelmed with a rather complex XML-structure! There is a clear need to keep this as untouched as possible. Not only do you want to keep the Cocoon infrastructure intact and operating, but you also have the feeling that you want to keep your work cleanly separated from the Cocoon distribution.

The solution: Separating your work by use of subsitemaps

Because Cocoon is designed for high scalability, there is a concept of "divide and conquer" just built into the sitemap concept. The concept of *submounting* is well described in the Cocoon documentation. See chapter "the sitemap"

We will use subsitemaps in order to get our work separated out into a hierarchy of subfolders. And here is how it's done:

Part I: Use the existing possibilities

I will point you to a detail right in the middle of the originally distributed `$cocoon_root/sitemap.xmap` (near line 860):
There you find following code snippet (i took off the comments here):

```
<map:pipeline>
  <map:match pattern="mount/*/**">
    <map:mount check-reload="yes" src="mount/{1}/" uri-prefix="mount/{1}"/>
  </map:match>
</map:pipeline>
```

This little piece of the sitemap allows you to separate your work cleanly from the rest of cocoon. What does it do ? Look at the `pattern="mount/*/**"` definition in the secnd line. This *match pattern* is interpreted by cocoon as follows:

a - Any URL, that points to the Cocoon webapp

b - followed by a 'mount/'

c - followed by an arbitrary foldername (the first "*" in the pattern)

d - followed by a '/' (the "/" in the pattern)

e - followed by a path of arbitrary depth ("**" in the pattern).

When the interpreter hits the wildcard character '*' or the sequence '**' in the pattern, it will automatically resolve the actual value into local *sitemap-parameters*, where the first wildcard-character is resolved into the parameter {1}, then second wildchard-character is resolved into the parameter {2}, ...

Following rules apply:

1. a single '*' is resolved into any character sequence inbetween two '/' characters.
2. a double '**' is resolved into any character sequence delimited by a '/*' on the right side or a */ at the left side.

These abstract rules can best be understood by looking at an example. In the following we apply the URL (given in the first line) to the match-pattern (in the secnd line):

```
http://localhost:8080/cocoon/ mount/ work/index.xml   <-- URL
~~~~~~~~~~a~~~~~~~~~~~~~~~~~ ~~b~~~ ~~c~d~e~~~~~~~
                              mount/   * / **         <-- pattern
                                      {1}  {2}        <-- sitemap-parameters
```

Here Cocoon associates the string "work" to the parameter **1** and the string "index.xml" to the parameter {2}. These parameters can be freely used within the pipeline snippet as you can see above. In this case we only make use of **1** within in the <map:mount> tag.

Cocoon tries to open a subsitemap within the folder indicated by the "src" attribute of the <map:mount> tag. In this example that would be `$cocoon_root/mount/{1}/` which resolves to `$cocoon_root/mount/work/` as you can conclude from the explaination above.

If you take a look at $cocoon_root you will already find the subfolder named "mount". All you have to do right now is create a new folder of the same name as **1** ("work" in the example above) right within the mount folder and start your work therein. And now we are ready for connecting your "work" folder to cocoon:

Part II: Connecting your "work" folder to Cocoon

Each time Cocoon is accessed with a URL of the form http://localhost:8080/cocoon/ mount/{yourFolder}/{path of arbitrary depth}, it will look in the $cocoon_root/mount/{yourFolder} folder for a `sitemap.xmap` file. Within `$cocoon_root/mount/{yourFolder}/sitemap.xmap` you will now define your project specific components, e.g. your pipelines. At the beginning of your development, this file may be very short.

Assume you want to serve an XML-content that is serialized into an HTML-output by use of an XSL-transformation. This, I think, is the simplest "use case" for Cocoon. I assume, the XSLT transformation is kept in the file "mytransform.xsl" and all files you want to serve have the ending ".xml". The setup is then straightforward:

- Enter your work folder
- Create a new sitemap.xmap within your "work" folder and place following content into the newly created file:

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

  <map:components>
    <map:generators   default="file"/>
    <map:transformers default="xslt"/>
    <map:readers      default="resource"/>
    <map:serializers  default="html"/>
    <map:matchers default="wildcard"/>
  </map:components>

  <map:pipelines>
    <map:pipeline>
      <!-- xml files -->
      <map:match pattern="*.html">
        <map:generate src="{1}.xml"/>
        <map:transform src="mytransform.xsl"/>
        <map:serialize type="html"/>
      </map:match>
    </map:pipeline>
  </map:pipelines>
</map:sitemap>
```

What did we achieve?

First we told Cocoon to start a xslt transformation whenever we try to retrieve an html file from the "work" folder. e.g., the following URL:

`http://localhost:8080/cocoon/mount/work/index.html`

gets Cocoon to look for a file named "index.xml" within your work folder. Then the file "index.xml" is transformed by use of the XSLT transformation stored in "mytransform.xsl". Finally the result is serialised by use of the HTML-serialiser. The final result is sent to the browser as standard html file.

Note: There is no file named `index.html` anywhere in your folder. This is only the pattern, that triggers the processing of the file index.xml and eventually sending back HTML-content to the browser! Of course you may want to name your xsl transform differently, or you may choose not to use the pattern "*.html" to trigger this pipeline. From here it's up to you, how you proceed.

Please remember only one crucial point: the match patterns are completely decoupled from the true names of files in your folder. e.g. you may choose the pattern "*.xml" instead of "*.html" and keep the rest of the pipeline as is. Then Cocoon would send back your html-result, if it encounters the URL

`http://localhost:8080/cocoon/mount/work/index.xml`

(see here: Although you ask for "index.xml", you get back a html result!)

Advanced issue: separating your work folder from Cocoon

Now you have managed to separate your work from the Cocoon distribution. But you still work right within the cocoon webapp folder. This may be not desirable for several reasons. You will find more detailed information in BeginnerAdvancedWebappOrganisation.

Appendix

An xml you can put on your work directory (index.xml):

```
<?xml version="1.0"?>
 <page>
   <title>Basic XML/XSL Transformation Example </title>
   <greeting>Hello World</greeting>
 </page>
```

An XSL stylesheet (mytransform.xsl) for above:

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

 <xsl:template match="page">
  <html>
   <head>
    <title><xsl:value-of select="title"/></title>
   </head>
   <body>
    <h1><xsl:value-of select="title"/></h1>
    <p><xsl:value-of select="greeting"/></p>
   </body>
  </html>
 </xsl:template>

</xsl:stylesheet>
```

(Note: Make sure that the first processing instruction (<?xml version="1.0"?>) in the XML document has no spaces before it, this will result in an internal server error)

---

page metadata

- AUTHOR: Dabbous
- AUTHOR-CONTACT: hussayn.dabbous@saxess.de
- REVIEWED-BY: DerekH (minor editing)
  Maskkkk (Added note at bottom)
  JamesMason (minor formatting)
- REVIEWER-CONTACT:
