Paloose Documentation

Hugh Field-Richards

11th December 2012

Abstract

This is a paper copy of the Paloose documentation that is on the Paloose Web site (www.paloose.org). It was generated automatically by a suitable XSL transformer from XML to LaTeX so the formatting is not always optimum. If there are any differences between the printed and Web versions then the Web site should be considered definitive.

Contents

1 Welcome to Paloose

Paloose is a simplified version of Cocoon using PHP. It resulted from scratching a long standing personal itch: that there are very few ISPs who will support Java/Tomcat for web sites, other than as a very expensive "professional" addition. Almost all will support PHP5 (sorry, Paloose does not use PHP4), so I decided to write my version of a simple, cut-down Cocoon in PHP5. I wanted to use XML on my personal sites but could not use Cocoon because of the expense. I have been using Paloose for some time now and have always found it a good substitute for Cocoon in all but the most complex sites. Paloose may also encourage others to start using XML and XSL without having to use extra bits such as Tomcat, Jetty or a full Cocoon installation.

Please note that the technology underlying Paloose does not make it suitable for very large sites. If you need performance (see a discussion of performance issues here) then upgrade to Cocoon — the extra expense of an updated server account will probably be unnoticeable in the overall cost of a large site anyway. However, having the ability to try out XML and XSL ideas in a PHP environment with a subset of Cocoon is very useful.

Although Paloose is free software (see Licence here) I would appreciate an EMAIL (hsfr@hsfr.org.uk) from anyone who downloads it — especially those who actual use it. Comments are welcome, good or bad.

2 Intoduction

Paloose is a PHP5 version of the Cocoon system. The Cocoon system allows separation of content and style so that designers can concentrate on each part separately, something that was always difficult to do with HTML, although that situation has improved. In general the content is stored as XML (although not exclusively) which is then transformed (via XSL) into a form that can be displayed on a client browser. By changing the transformations a variety of outputs can be supported (different browsers, RSS feeds etc.), all from the same content. The key to controlling this process is a *sitemap* which contains the instructions of how each incoming request is handled.

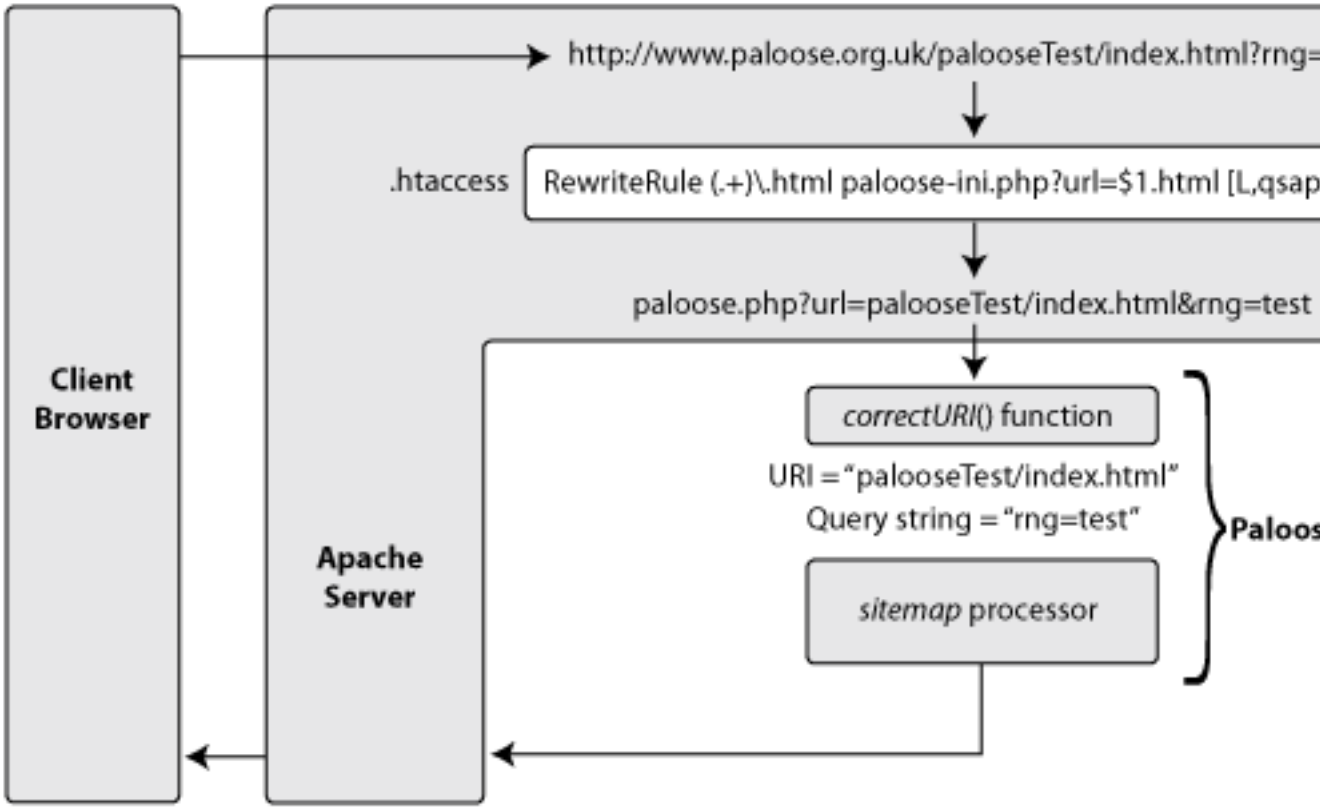
Each sitemap consists of a series of pipelines that enclose a set of components in exactly the same form as Cocoon. These components provide the basic mechanism of reading (generating), transforming and outputting (serializing) data. See the components' description for more details of sitemaps. Although it is not essential, a knowledge of Cocoon is useful.

Paloose can work in two ways: as a *server* or as a *client*.

2 Paloose Server

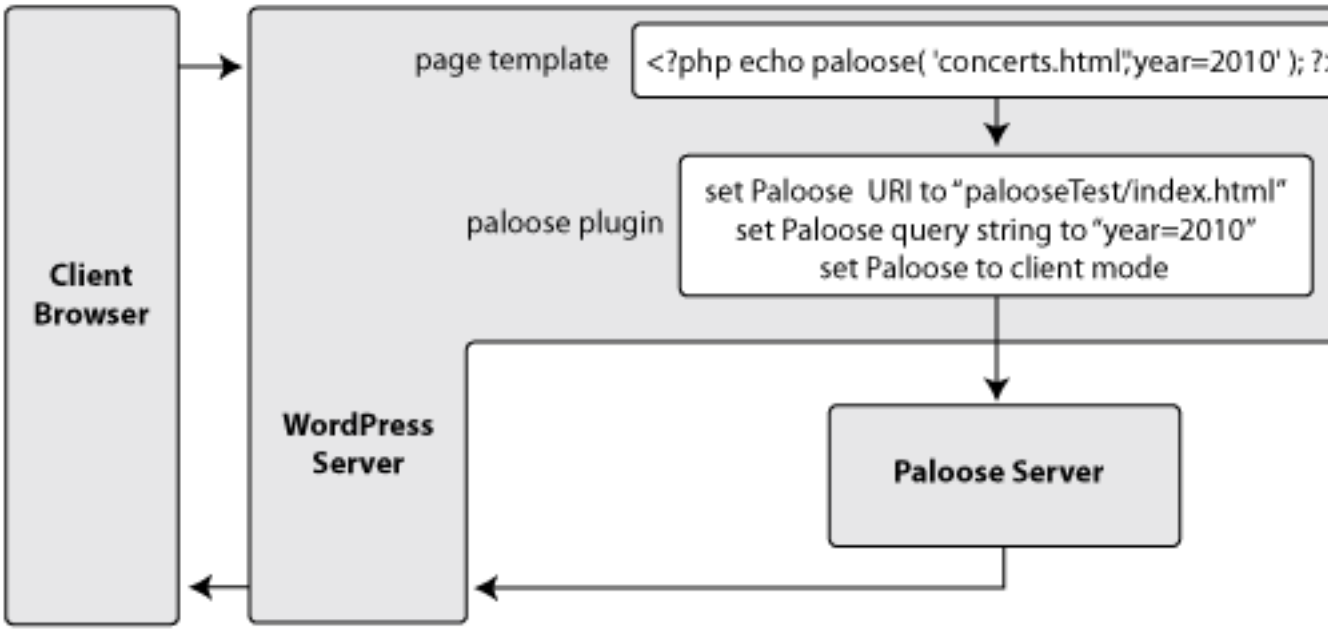
The Paloose server is designed as a "standalone" system which works with the Apache server. The term server here when applied to Paloose is a little loose since it could be said that it is acting a client to Apache; much as Cocoon is built on the Tomcat server. However it is relatively safe to consider it as a server to distinguish it from the client system.

When used as a server the requests that are received by Apache are forwarded directly to the Paloose system by Apache using the .htaccess file. The flow of control is shown here:



2 Paloose Client

It is possible to use Paloose as a client to such Web applications/servers as WordPress. The process requires making a suitable interface to the server and instructing Paloose to act as a client. For example in WordPress there is a suitable plugin that provides this interface, together with the administration page to change the Paloose environment (although this is not normally necessary). The overall flow is:



For more details on Plugins for Paloose see here

3 Installation Guide

Installation is very simple — it is just making sure that the Paloose directory is in the right place.

1. Unpack the download into the destination directory. This is currently in `/Library/WebServer/Documents` on my development box (MacPro Quad-Core Intel Xeon 2.66 GHz running Mac OS X 10.6.4). This will be *ROOT_DIR* in the examples below.

```
${ROOT_DIR}/paloose hsfr$ ls -l
total 200
-rw-r--r--   1 hsfr  wheel  101095 May 17 12:20 paloose-latest.tgz
${ROOT_DIR}/paloose hsfr$ tar zxvf paloose-latest.tgz
configs/
lib/
lib/environment/
...
${ROOT_DIR}/paloose hsfr$ rm paloose-latest.tgz
${ROOT_DIR}/paloose hsfr$ ls
configs        lib        resources
${ROOT_DIR}/paloose hsfr$
```

2. That's it.

The only thing that must be done to get Paloose to run is to make sure that your ISP is happy to run PHP5 with the correct XML parser. For example, when I built my PHP5 on my local server the configure command was run with "`--with-xsl`". The important thing is that you must not use "`--with-xslt-sablot`". You need to turn off Sablotron support and use *libxml*.

Configuring Paloose is described here and requires Paloose to be told where the various components are stored. However a standard insall required very little changes.

4 Configuring Paloose

Configuring Paloose is dependant on what system you are running. A server based system (that is how the original Paloose was configured) uses a simple calling program in your site home page and is describe here. Using Paloose as a server also will require setting a suitable *.htaccess* file. This is relatively simple and is described here.

If your are using paloose as a client (a plugin for such systems as WordPress or Joomla!) then follow these instructions.

5 First Site

Note that the Paloose system is a subset of Cocoon so almost all of the features in Paloose are directly derived from Cocoon so it is useful but not essential to know about Cocoon. In order to see how it works let us build that archetypal beginner's site: "Hello World".

1. Create a base folder to hold your site where your Apache server will see it. In this case we will create a folder '`HelloWorld`' in the root directory that we created during the install.

```
${ROOT_DIR} hsfr$ mkdir HelloWorld
${ROOT_DIR} hsfr$ cd HelloWorld
${ROOT_DIR}/HelloWorld hsfr$
```

2. Create the folders '`resources/transforms`' and '`resources/syles`'.

```
${ROOT_DIR}/HelloWorld hsfr$ mkdir -p resources/transforms
${ROOT_DIR}/HelloWorld hsfr$ mkdir -p resources/styles
${ROOT_DIR}/HelloWorld hsfr$ mkdir -p content
${ROOT_DIR}/HelloWorld hsfr$
```

3. Create a '`content/page.xml`' file which is the content of the page. Note that there is no style information here at all — we are only describing what the individual parts of the page are. The file should contain the following XML code:

```
<?xml version="1.0" encoding="UTF-8"?>
<page>
   <heading>Hello World</heading>
   <p>My first page using Paloose.</p>
</page>
```

4. Now we need a transformation to take that and turn it into something that the browser will understand. Create a file '`resources/transforms/page2html.xsl`' containing the following:

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

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

<!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

<xsl:template match="/">
   <xsl:element name="html">
      <xsl:call-template name="htmlHead"/>
      <xsl:call-template name="htmlBody"/>
```

```
      </xsl:element>
   </xsl:template>

   <xsl:template name="htmlHead">
      <xsl:element name="head">
         <xsl:element name="title">
            <xsl:value-of select="//heading"/>
         </xsl:element>
      </xsl:element>
   </xsl:template>

   <xsl:template name="htmlBody">
      <xsl:element name="body">
         <xsl:apply-templates mode="inline-text"/>
      </xsl:element>
   </xsl:template>

   <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- -->

   <xsl:template match="heading" mode="inline-text">
      <xsl:element name="div">
         <xsl:attribute name="class">heading</xsl:attribute>
         <xsl:apply-templates mode="inline-text"/>
      </xsl:element>
   </xsl:template>

   <xsl:template match="p" mode="inline-text">
      <xsl:element name="div">
         <xsl:attribute name="class">normalPara</xsl:attribute>
         <xsl:apply-templates mode="inline-text"/>
      </xsl:element>
   </xsl:template>

</xsl:stylesheet>
```

This is a fairly crude transformation and real ones will be considerably more complex that this.

5. Note that the code that this produces is based on `div`s. The final style is not put in until we add a style file. So create a file:

```
   body {
      background-color: #d2cab5;
      color: #66766d;
      margin: 20px 0px 0px 20px;
   }

   .normalPara {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      margin: 10px 0px 0px 0px;
   }

   .heading {
      font-size: 18px;
      font-family: Georgia, "Times New Roman", Times, serif;
      font-style: italic;
      color: #56554a;
   }
```

It is worth noting here that we have three separate (and hopefully orthogonal) files that make up our simple site. The content (heading and paras), the layout structure (text blocks) and the layout style (fonts, colours etc). All should be able to be manipulated with minimal interference with each other.

6. We need to tie them all together so that a request for `http://hostname/HelloWorld/page.html` will produce the page that we require. The control of this is within the *sitemap.xmap* file. It follows standard Cocoon practice with only minor differences in the component definitions.

First of all we define the components that we are going to use. Note that to those wo are more used to Cocoon the '*src*' attribute normally defines a Java package, in Paloose it is the name of a PHP file. So `paloose://lib/generation/FileGenerator`' defines a file '`${ROOT_DIR}/paloose/lib/generation/FileGenerator.php`'. This means that you can define your own components fairly easily and place them where you want (there is an example component given in the distribution).

```
<?xml version="1.0" encoding="UTF-8"?>

<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

   <map:components>

   <map:generators default="file">
      <map:generator name="file" src="resource://lib/generation/FileGenerator"/>
   </map:generators>

   <map:transformers default="xslt">
      <map:transformer name="xslt" src="resource://lib/transforming/TRAXTransformer"/>
   </map:transformers>

   <map:serializers default="html">
      <map:serializer name="html" mime-type="text/html"
         src="resource://lib/serialization/HTMLSerializer"/>
   </map:serializers>

   <map:matchers default="wildcard">
      <map:matcher name="wildcard" src="resource://lib/matching/WildcardURIMatcher"/>
   </map:matchers>

   </map:components>
```

The components that are defined above are the default ones and support almost all of what you might require. The HTML serializer needs some more work but the current version should suffice most people's needs at present. Next we use these components to build pipelines (in the same fashion as Cocoon) to handle requests.

```
   <map:pipelines>

      <map:pipeline>
         <map:match pattern="**.html">
            <map:generate src="context://content/{1}.xml"/>
            <map:transform src="context://resources/transforms/page2html.xsl"/>
            <map:serialize/>
         </map:match>
     </map:pipeline>

   </map:pipelines>

</map:sitemap>
```

7. Having done the sitemap we must now tell the Apache server what to do to use Paloose. This is done using the `.htaccess` file (note the leading period in the name). The key is to use the *Rewrite* facility of the Apache server. So place the `.htaccess` file in the home directory of your site with the following:

```
RewriteEngine On
RewriteRule (.+)\.html paloose.php?url=$1.html [L,qsappend]
```

This tells Apache that all requests for files in the HelloWorld directory (in which the `.htaccess` file sits) should obey the rule above. So a request for '`http://host-name/HelloWorld/page.html`' would be translated to '`http://host-name/HelloWorld/paloose.php?url=page.html`'.

Warning

If you see the following error (or similar)

```
"Parse error: syntax error, unexpected '=', expecting '(' in
   ....../paloose/lib/Paloose.php
   on line 88"
```

when you try and run the simple site, it is almost certain that you are running PHP4. You will need to speak to your ISP to find out how to direct all Paloose code to use PHP5 (assuming that they have this as an option).

8. The final part of the puzzle is what to do with the rewritten request above. We need a file `paloose.php` in the `HelloWorld folder`. This contains all the necessary user defined information and the link to start Paloose running. There is an example file in the Paloose distribution to help you get started in `$ {ROOT_DIR}/paloose/resources/templates/paloose.php.dist`.

   The interesting parts of the `paloose.php` file that you may want to change, are detailed below. Change the *PALOOSE_DIRECTORY* address here to indicate where you have put the main Paloose folder. It can be relative to your application site folder.

```
define( 'PALOOSE_DIRECTORY', '../paloose' );
```

   The next lines define where the main log4php logger configuration file is kept. You can use pseudo-protocols here. You also define the path to the Log4PHP distribution (`LOG4PHP_DIR`). This is a change since version 1.1.2b1. If you are using versions earlier than this then the logging library is included within Paloose and this line (`LOG4PHP_DIR`) is not necessary.

```
define( 'LOGGER_CONFIG', 'context://configs/Paloose.xml' );
define( 'LOG4PHP_DIR', '../log4php' );
define( 'TIME_ZONE', 'Europe/London' );
```

   Normally should not have to change the root sitemap if you use the standard Cocoon names. The directory is always the top level user directory.

```
define( 'ROOT_SITEMAP', 'sitemap.xmap' );
```

   The error page to give back for User errors (errors in sitemap etc). If you override these to your area make sure that you use an absolute directory path — you can use the '*context*' pseudo-protocol here.

```
define( 'USER_EXCEPTION_PAGE', 'resource://resources/errorHandling/userError.html' );
define( 'USER_EXCEPTION_TRANSFORM', 'resource://resources/transforms/errorPage2html.xsl' );
```

   Be careful of the path here — although it is relative it looks as an absolute one. It is in fact relative to the Apache root document directory. Must always have a leading path separator.

```
define( 'USER_EXCEPTION_STYLE', '/paloose/resources/styles/userError.css' );
```

   The file that contains the index for each level of the gallery. For this simple example it can be ignored. Indeed 99% of the time it could be left as this. The ImageMagick bin path is rarely needed if your server has the various binaries of the ImageMagick package on the *Path*. It is commented out in the `paloose.php.dist` file.

```
define( 'GALLERY_INDEX', 'gallery.xml' );
```

```
define( 'IMAGEMAGICK_BIN', '' );
```

The error page to return to the user for internal programming errors. These should not need to be overridden unless you want to use your own.

```
define( 'INTERNAL_EXCEPTION_PAGE', 'resource://resources/errorHandling/internalError.html' );
define( 'INTERNAL_EXCEPTION_TRANSFORM', 'resource://resources/transforms/errorPage2html.xsl' );
```

Again, be careful of the path here — it is relative to the Apache root document directory. Must always have a leading path separator.

```
define( 'INTERNAL_EXCEPTION_STYLE', '/paloose/resources/styles/internalError.css' );
```

9. Last remaining thing is to open a browser and go to the address `http://[host-name]/HelloWorld/page.html`. If all is well you should see the following

6 Sitemap Guide

The sitemap is the key part of running a site based on either Cocoon and Paloose. It defines all the components to be used and the pipelines to be run in response to a particular browser request. It is definitely worth reading the Apache documentation about the Cocoon sitemap.

Note

Remember that everything that goes through Paloose takes time. Rather than use Paloose to serve files see if Apache can do this for you. In general resources such as style sheets and images are static files and do not need transforming. This is the same situation in Cocoon: if Apache can serve it — don't process it.

6 The Structure

The Paloose site has a similar structure to the Cocoon one, just less of it.

```
<?xml version="1.0"?>
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">
   <map:components>
      <map:generators/>
      <map:transformers/>
      <map:serializers/>
      <map:selectors/>
      <map:readers/>
      <map:matchers/>
   </map:components>
   <map:views/>
   <map:resources/>
   <map:pipelines/>
</map:sitemap>
```

6 Sitemap Namespace

The Paloose namespace is identical to that used by Cocoon (`http://apache.org/cocoon/sitemap/1.0`), though there is no real reason why it should not be one specific to Paloose. This could be changed in future. If you wish to do this yourself note that the namespace is changed in the main Paloose configuration file in the user's site.

6 Components

The Paloose sitemap components are described elsewhere. However there are some common concepts that are described below.

6 Regular Expression Pattern Matching

Since version 0.7.0 it has been possible to carry out matching using regular expressions. ***Paloose regular expressions are not the same as Cocoon regular expressions.*** The PHP5 version based on Perl is used and a full description of the regexp can be found on the PHP5 manual page. The pattern variables work in similar fashion as the wildcard matcher. For example:

```
<map:match type="regexp" pattern="/.*\.(html)|(tex)|(xml)/">
      ...
</map:match>
```

would match all requests that have a file extension "html", "tex" and "xml". In the following all requests for documents between "nb1996" and "nb1999" would read the html file directly:

```
<map:match type="regexp" pattern="/nb199([6789]).html/">
   <map:read src="context://nb/199{1}/nb199{1}.html"/>
</map:match>
```

and the following would take all others in 2000 or later and process them from an XML file.

```
<map:match type="regexp" pattern="/nb20(.+).html/">
   <map:generate src="cocoon:/20{1}/nb20{1}.xml" label="xml-content"/>
   <map:transform src="context://resources/transforms/notebooks-html.xsl" label="page-transform">
      <map:parameter name="page" value="nb20{1}"/>
   </map:transform>
   <map:serialize/>
</map:match>
```

6 Variables

There are variables that can be used within the sitemap. These are stored in the appropriate module and can be accessed by using the syntax "{*module_name*:*variable_name*}". There are currently two preset modules for variables: *global*, *error* and *request-param*.

6 Global Variables

Global variables can be declared within each sitemamp and are available to each subsitemap. They are also overwritten by subsitemap declarations. They are declared within the sitemap pipelines declaration as a component configuration element, for example

```
<map:pipelines>

   <map:component-configurations>
      <global-variables>
         <composer>Bach</composer>
      </global-variables>
   </map:component-configurations>

   ...
```

which would set the global variable "*composer*" to the string "Bach". It could then be accessed like other variables within the sitemap as:

```
<map:transform type="mysql" label="sql-transform">
   <map:parameter name="show-nr-of-rows" value="true"/>
   <map:parameter name="composer" value="{global:composer}"/>
   </map:transform>
```

Note that since Version 1.3.4 the sitemap variables formed from the matcher "`{0},{1},{2},...`" have been included in the global variables. The can be accessed by using "`{global:__0},{global:__1},{global:__2},` respectively.

6 Request Parameter Variables

One important set The *RequestParameterModule* allows access to the query string variables. For example if the URI request is `index.html?locale=en` the *RequestParameterModule* gives the sitemap access to the query string. Using this is very simple

```
<map:match pattern="**.html">
   <map:generate src="context://{1}.xml"/>
   <map:transform src="context://resources/transforms/page2html.xsl">
      <map:parameter name="locale" value="{request-param:locale}"/>
   </map:transform>
   ...
</map:match>
```

This would pass the parameter *locale* into the XSLT script where it could be accessed using the following typical code:

```
<xsl:param name="locale"/>
...
<xsl:value-of select="$locale" />
```

6 Handling Pipeline Errors

When an error is detacted in a sitemap (page does not exist because a matcher has not fired, for example) a specialised error pipeline is run. More details can be found in the Handle Error documentation page.

6 Protocols

As in a Cocoon sitemap, you can use all protocols nearly everywhere in a Paloose sitemap. The following are a list of the Paloose protocols and what they mean. Note that there are subtle differences to Cocoon.

- `context://` — get a resource directory where user's site is installed
- `cocoon:/` — get a resource from the current sitemap
- `cocoon://` — get a resource using the root sitemap
- `resource://` — get a resource from directory where Paloose is installed

7 Mounting sitemaps

One of the most useful concepts of the Paloose/Cocoon sitemap is the ability to "divide and conquer". It is possible for a sitemap to invoke another "sub" sitemap and transfer control to it. There is a certain amout of inheritance of components so that they do not have to be redefined. Although this does not preclude having extra components defined in the subsitemaps. Subsitemaps allow a hierarchy or tree of sitemaps underneath the master sitemap. The advantage of this is that a site with several different areas can be developed separately using a sitemap for each area, making larger sites easier to maintain. The master sitemap controls which sitemap is used dependent on the request.

```
<map:match pattern="documentation/*.html">
   <map:mount uri-prefix="documentation" src="documentation/sitemap.xmap"/>
</map:match>
```

The attributes are similar to those used in Cocoon

- *src* — the file name os the subsitemap. If *src* ends is a path separator (for example "/") then the filename `sitemap.xmap` will be added, otherwise the defined filename will be used.
- *uri-prefix* — defines the part of the request URI that should be removed when passing the request into the subsitemap. For example using the mount element above if the requested URI was "`documentation/sitemap.html`" then "`documentation/`" would be removed from the request passed to the subsitemap (`documentation/sitemap.xmap`). Note that the trailing path separator is removed as well. ***As far as I can see Cocoon insists on this attribute being present (but can be an empty string). Paloose makes this attribute optional.***

Note

Performance note: there will be a performance penalty if subsitemaps are nested too deep.

8 Redirecting Requests

It is possible to redirect a request to Paloose to a completely different URI (on the same or different site). For example if I had split the Paloose documentation onto a completely different server then I might put

```
<map:match pattern="documentation/*.html">
   <map:redirect-to uri="http://<doumentation-host>/documentation/{1}.html"/>
</map:match>
```

The attribute is similar to that used in Cocoon:

- *uri* — the uri which must be used for the redirection.

9 Selectors

Selectors are the conditional constructions for sitemaps. The most commonly used one is the *BrowserSelector* which allows different routes in the pipeline dependant on the client's browser. Useful for differences in browser behaviour. The Paloose site makes use of this so that the text based Lynx and the iPhone browser are supported.

- BrowserSelector — selects pipeline fragment according to user's browser.
- RequestParameterSelector — selects pipeline fragment according to query string request parameter.
- RegexpSelector — selects pipeline fragment according to a regular expression in a variable.
- VariableSelector — selects a pipeline on basis of * request parameter from the query string..

9 Component Declaration

Selectors are defined in the component declaration part of the Sitemap.

```
<map:selectors default="browser">
    <map:selector name="browser" src="resource://lib/selection/BrowserSelector">
       <map:browser name="explorer" useragent="MSIE"/>
       ...
    </map:selector>
 </map:selectors>
```

The *default* attribute specifies the type of selector to use if none is specified in a pipeline.

9 BrowserSelector

The root sitemap needs to have the *BrowserSelector* declared as a component:

```
  <map:components>
       ...
       <map:selectors default="browser">
          <map:selector name="browser" src="resource://lib/selection/BrowserSelector">
             <map:browser name="explorer" useragent="MSIE"/>
             <map:browser name="pocketexplorer" useragent="MSPIE"/>
             <map:browser name="handweb" useragent="HandHTTP"/>
             <map:browser name="avantgo" useragent="AvantGo"/>
             <map:browser name="imode" useragent="DoCoMo"/>
             <map:browser name="opera" useragent="Opera"/>
             <map:browser name="lynx" useragent="Lynx"/>
             <map:browser name="java" useragent="Java"/>
             <map:browser name="wap" useragent="Nokia"/>
             <map:browser name="wap" useragent="UP"/>
             <map:browser name="wap" useragent="Wapalizer"/>
             <map:browser name="mozilla5" useragent="Mozilla/5"/>
             <map:browser name="mozilla5" useragent="Netscape6/"/>
             <map:browser name="netscape" useragent="Mozilla"/>
             <map:browser name="safari" useragent="Safari"/>
             <map:browser name="iphone" useragent="iPhone"/>
          </map:selector>
       </map:selectors>
       ...
    </map:components>
```

where

- **name** — the name of this selector (in this case *browser*).
- **src** — the location of the PHP package for this component.

Each sub-element, `<map:browser>`, defines a particular browser by its user agent type, where

- **name** — the name of the selection when used in the pipeline.
- **useragent** — the client browser defined by its agent.

We can use the selector in the pipeline in any position provided that the overall conditions for the pipeline are met: that is, a pipeline must start with a generator and end with a serializer, or have a mount or reader pipeline element. For example (from this site):

```
        <map:match pattern="**.html">
        <map:aggregate element="root" label="aggr-content">
           <map:part src="cocoon:/headings.xml" element="headings" strip-root="true"/>
           <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
           <map:part src="cocoon:/newsArticles.xml" element="news-articles" strip-root="true"/>
           <map:part src="cocoon:/{1}.xml" element="content" strip-root="true"/>
        </map:aggregate>
        <map:select type="browser">
           <map:when test="lynx">
              <map:transform src="context://resources/transforms/page2lynx.xsl" label="page-transform">
                 <map:parameter name="page" value="{1}"/>
              </map:transform>
           </map:when>
           <map:otherwise>
              <map:transform src="context://resources/transforms/page2html.xsl" label="page-transform">
                 <map:parameter name="page" value="{1}"/>
              </map:transform>
           </map:otherwise>
        </map:select>
        <map:transform src="context://resources/transforms/stripNamespaces.xsl"/>
        <map:serialize type="html" />
     </map:match>
```

9 RequestParameterSelector

The sitemap needs to have the *RequestParameterSelector* declared as a component:

```
   <map:selectors default="browser">
      ...
      <map:selector name="request-parameter" src="resource://lib/selection/RequestParameterSelector">
         <map:parameter-name>type</map:parameter-name>
      </map:selector>
   </map:selectors>
```

where

- **name** — the name of this selector (in this case *request-parameter*).
- **src** — the location of the PHP package for this component.

The `<map:paraneter-name>` tag defines a default test parameter in the query string if one is not defined in the pipeline (in this case 'type'.

We can use the selector in the pipeline in any position provided that the overall conditions for the pipeline are met: that is, a pipeline must start with a generator and end with a serializer, or have a mount, call or reader pipeline element. For example (from the Chandos Symphony Orchestra site):

```
<map:match pattern="*.html">
      <map:aggregate element="root" label="aggr-content">
         <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
         <map:part src="cocoon:/newsArticles.xml" element="newsArticles" strip-root="true"/>
         <map:part src="cocoon:/futureConcerts.xml" element="futureConcerts" strip-root="true"/>
         <map:part src="cocoon:/{1}.xml" element="content" strip-root="true"/>
      </map:aggregate>
      <map:select type="request-parameter">
```

```
        <map:parameter name="parameter-name" value="type"/>
        <map:when test="xml">
           <map:transform src="context://resources/transforms/extractXML.xsl" label="page-transform">
              <map:parameter name="page" value="{1}"/>
           </map:transform>
           <map:call resource="outputXML"/>
        </map:when>
        <map:when test="text">
           <map:transform src="context://resources/transforms/extractXML.xsl" label="page-transform">
              <map:parameter name="page" value="{1}"/>
           </map:transform>
           <map:transform src="context://resources/transforms/xml2text.xsl" label="text-transform">
              <map:parameter name="page" value="{1}"/>
           </map:transform>
           <map:call resource="outputPage"/>
        </map:when>
        <map:otherwise>
           <map:transform src="context://resources/transforms/page2html.xsl" label="page-transform">
              <map:parameter name="page" value="{1}"/>
           </map:transform>
           <map:call resource="outputPage"/>
        </map:otherwise>
     </map:select>
  </map:match>
```

9 RegexpSelector

The sitemap needs to have the *RegexpSelector* declared as a component:

```
  <map:selectors default="browser">
        <map:selector name="regexp-selector" src="resource://lib/selection/RegexpSelector"/>
  </map:selectors>
```

where

- **name** — the name of this selector (in this case *regexp-selector*).
- **src** — the location of the PHP package for this component.

We can use the selector in the pipeline in any position provided that the overall conditions for the pipeline are met: that is, a pipeline must start with a generator and end with a serializer, or have a mount, call or reader pipeline element. For example (from an experimental Paloose CMS system:

```
    <map:component-configurations>
        <map:global-variables>
           <themesDir>context://themes</themesDir>
           <defaultTheme>{ant:default-theme}</defaultTheme>
           <currentTheme>{cookies:theme}</currentTheme>
           <rootDir>context://</rootDir>
        </map:global-variables>
     </map:component-configurations>

     <map:pipeline>

        <map:match pattern="/(page).html/" type="regexp">
           <map:act type="cookies">
              <map:select type="regexp-selector">
                 <map:parameter name="test-value" value="{global:currentTheme}"/>
                 <map:when test="/.+/">
                    <map:generate src="context://themes/{global:currentTheme}/data/body.xml" label="raw-xml"/>
                 </map:when>
                 <map:otherwise>
                    <map:generate src="context://themes/astral/data/body.xml" label="raw-xml"/>
                 </map:otherwise>
              </map:select>
              ...
```

```
            <map:transform type="moduleWrite"/>
         </map:act>
      </map:match>
```

In the example the selector checks for a current theme (/.+/ is any character). If the string is empty then the otherwise clause is carried out. If there is a theme declared then the appropriate theme directory is used.

9 VariableSelector

The sitemap needs to have the *VariableSelector* declared as a component:

```
<map:selectors default="browser">
      <map:selector name="variable-selector" src="resource://lib/selection/VariableSelector"/>
</map:selectors>
```

where

- **name** — the name of this selector (in this case *variable-selector*).
- **src** — the location of the PHP package for this component.

We can use the selector in the pipeline in any position provided that the overall conditions for the pipeline are met: that is, a pipeline must start with a generator and end with a serializer, or have a mount, call or reader pipeline element. For example (from an experimental Paloose CMS system:

```
    <map:component-configurations>
        <map:global-variables>
           <themesDir>context://themes</themesDir>
           <defaultTheme>{ant:default-theme}</defaultTheme>
           <currentTheme>{cookies:theme}</currentTheme>
           <rootDir>context://</rootDir>
        </map:global-variables>
     </map:component-configurations>

     <map:pipeline>

        <map:match pattern="/(page).html/" type="regexp">
           <map:act type="cookies">
              <map:select type="variable-selector">
                 <map:parameter name="parameter-name" value="{cookies:theme}"/>
                 <map:when test="">
                    <map:generate src="context://themes/astral/data/body.xml" label="raw-xml"/>
                 </map:when>
                 <map:otherwise>
                    <map:generate src="context://themes/{cookies:theme}/data/body.xml" label="raw-xml"/>
                 </map:otherwise>
              </map:select>
              ...
              <map:transform type="moduleWrite"/>
           </map:act>
        </map:match>
```

This almost the same as the previous example.

10 Aggregation

Aggregation is one of the most useful facilities of Cocoon and Paloose. It takes several XML documents and aggregates them into a single XML document with appropriate enclosing tags for each part. Within the `<map:aggregate>` element there are a number of `<map:part>` elements that define which XML documents shold be aggregated together. A working example is given here.

The `<map:aggregate>` element has a single attribute

- *element* — the parent element within which the entire aggregated document must be placed.

Each `<map:part>` element has a number of attributes that control how each part is to be included:

- *src* — the name and location of the document to be included. Pseudo protocols can be used (although full URLs have not been implemented in Paloose yet).
- *element* — the parent element within which this part must be placed.
- *strip-root* — if this is "true" then the root element of the included document will be removed before inclusion.

As an example of aggregation consider the following extract from the Paloose site (note the use of an internal only pipeline):

```
<map:pipelines>
   <map:pipeline>

      <map:match pattern="**.html">
         <map:aggregate element="root" >
            <map:part src="cocoon:/headings.xml" element="headings" strip-root="true"/>
            <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
            <map:part src="cocoon:/newsArticles.xml" element="news-articles" strip-root="true"/>
            <map:part src="cocoon:/{1}.xml" element="content" strip-root="true"/>
         </map:aggregate>
         ...
      </map:match>
   </map:pipeline>

   <map:pipeline internal-only="true">
      <map:match pattern="menus.xml">
         <map:generate src="context://content/menus.xml"/>
         <map:serialize/>
      </map:match>

      <map:match pattern="headings.xml">
         <map:generate src="context://content/headings.xml"/>
         <map:serialize/>
      </map:match>

      <map:match pattern="newsArticles.xml">
         <map:generate src="context://content/newsArticles.xml"/>
         <map:serialize/>
      </map:match>

      <map:match pattern="**.xml">
         <map:generate src="context://content/{1}.xml"/>
         <map:serialize/>
      </map:match>
   </map:pipeline>
</map:pipelines>
```

This will produce an aggregated document:

```
<?xml version="1.0"?>
<root>
   <headings>
      ...
```

```
    </headings>
    <menus>
       ...
    </menus>
    <news-articles>
       ...
    </news-articles>
    <content>
       ...
    </content>
</root>
```

11 Views

Views are means of interrupting the normal pipeline flow under the control of the user via the quesry string. It is similar to that of Cocoon but not quite so extensive. However even in its simple form it is a useful debugging aid. Take a typical example:

```
<map:views>
   <map:view name="raw" from-label="raw-content">
      <map:transform src="context://resources/transforms/xml2xhtml.xsl"/>
      <map:serialize/>
   </map:view>
</map:views>
```

The view is accessed by the *name* attribute and the label used in the actual pipeline is defined by the *label* attribute. ote that not components can be used in view pipelines; only *call*, *transform* and *serialize*. A typical pipe that uses this could be:

```
<map:match pattern="**.html">
   <map:aggregate element="root" label="raw-content">
      <map:part src="cocoon:/headings.xml" element="headings" strip-root="true"/>
      <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
      <map:part src="cocoon:/newsArticles.xml" element="news-articles" strip-root="true"/>
      <map:part src="cocoon:/{1}.xml" element="content" strip-root="true"/>
   </map:aggregate>
   <map:transform src="context://resources/transforms/page2html.xsl">
      <map:parameter name="page" value="{1}"/>
   </map:transform>
   <map:transform src="context://resources/transforms/stripNamespaces.xsl"/>
   <map:serialize type="html"/>
</map:match>
```

If the user puts in the URL, `http://hostname/index.html?paloose-view=raw`, the view pipeline named *raw* will be run after the aggregator is run in the normal pipeline, using the label *raw-content*. In this case the transform "`xml2xhtml.xsl`" (an XML pretty-printer) will be run on the output of the aggregator.

The only pipeline elements that can be labelled for views are *aggregate*, *generate* and *transform*.

12 Generators

Generators form the start of a pipeline and there should always be one present (but see aggregate, mount and read). Currently supported generators are:

- FileGenerator — reads an XML file and inserts it into the pipeline as a DOM.
- PXTemplateGenerator — The same as *FileGenerator* except that it can process Paloose variable modules.
- DirectoryGenerator — Outputs a directory listing.
- GedComGenerator — Takes an GEDCOM file and produces an XML equivalent.

12 Component Declaration

Generators are defined in the component declaration part of the Sitemap.

```
<map:generators default="file">
   <map:generator name="file" src="resource://lib/generation/FileGenerator"/>
   <map:generator name="directory" src="resource://lib/generation/DirectoryGenerator"/>
   <map:generator name="px" src="resource://lib/generation/PXTemplateGenerator"/>
</map:generators>
```

The *default* attribute specifies the type of generator to use if none is specified in a pipeline.

12 Using generators

Generators should always be placed first in a pipeline. The only exception to this is an aggregator (which will have an implied generator as part of its use), and the mount and read components. For example (using the above component declaration) the following is a simple pipeline extract:

```
        <map:match pattern="*.xml">
           <map:generate src="context://content/{1}.xml" label="xml-content" cachable="no" type="px"/>
           <-- Some transforms -->
           <map:serialize type="xhtml"/>
        </map:match>
```

where

- *type* — the type of generator to be used (defined in the component declaration by the name attribute). If this is omitted then the default type from the component declaration is used.
- *src* — the file to be input into the pipeline.
- *cachable* — should this component use caching (yes—no). The default is “no”
- *label* — the label used by the views facility

13 FileGenerator

File generators read an XML file and present it to the pipeline for processing. They are always the first item of a pipeline (except for an aggregator that implies a generator). So a typical use of *FileGenerator* would be:

```
<map:generators default="file">
   <map:generator name="file" src="resource://lib/generation/FileGenerator"/>
   ...
</map:generators>

<map:pipelines>

   <map:pipeline>

      <map:match pattern="downloads.xml">
         <map:generate src="context://content/downloads.xml"/>
         ...
      </map:match>

   </map:pipeline>
</map:pipelines>
```

where

- *src* — the file to be input into the pipeline.

The above code injects the file `content/download.xml` in the main site directory (context) into the pipeline. See also, PXTemplateGenerator.

Note

In versions after 1.3.4 implicit entities defined by ISOlat1, ISOpub and ISOnum are translated into their numerical form. The *EntityTransformer* now only handles explictly declared entities within a `DOCTYPE` statement within the XML file.

13 Caching Support (avaliable after Version 1.3.0)

The *FileGenerator* component supports caching of the data within the pipeline. The input file is normally checked for well-formedness (no schema validation) but caching the data stores the XML after this process. As a result very little time is saved except on very large XML source files.

Warning

Note that if you have an XML file that includes other files (via xinclude), the generator caching will not detect that these other XML files have been modified. If you modify them it is important to clear out the caching to let the cache files to be rebuilt.

Enabling caching is done globally and locally. To turn on caching for all *FileGenerators* there is an attribute *cachable* that should be set to true (true/yes/1). So setting the global flag would be (default is false):

```
<map:generators default="file">
   <map:generator name="file" src="resource://lib/generation/FileGenerator" cachable="yes"/>
</map:generators>
```

This can be overriden by using the same attribute when the pipeline component is used:

```
<map:match pattern="*.xml">
   <map:generate src="context://content/{1}.xml"
         label="xml-content"
         cachable="no"/>
   <map:serialize/>
</map:match>
```

The above would turn off the caching for just this instance in the pipeline. It is also possible to change the action of the cache via the query string by using the request parameters. The following would control the generator instance by including the query string "`?cachable=1`" or "`?cachable=0`"

```
<map:generators default="file">
   <map:generator name="file" src="resource://lib/generation/FileGenerator" cachable="no">
      <map:use-request-parameters>true</map:use-request-parameters>
   </map:generator>
</map:generators>

...

<map:match pattern="index.xml">
   <map:generate src="context://content/index.xml"
         label="index-xml"
         cachable="{request-param:cachable}"/>
   <map:serialize type="xml"/>
</map:match>
```

14 DirectoryGenerator

Directory generators read a specified directory and present it to the pipeline as an XML document for processing. So a typical use of *DirectoryGenerator* would be:

```
<map:generators default="file">
   <map:generator name="directory" src="resource://lib/generation/DirectoryGenerator"/>
   ...
</map:generators>

<map:pipelines>
   <map:pipeline>
      <map:match pattern="get-dir.html">
         <map:generate type="directory" src="context://resources">
            <map:parameter name="depth" value="2"/>
            <map:parameter name="dateFormat" value="F d Y H:i:s"/>
            <map:parameter name="include" value="/.*/"/>
            <map:parameter name="exclude" value="/.+\.xmap/"/>
            <map:parameter name="reverse" value="false"/>
         </map:generate>
         <map:call resource="xml2xhtml"/>
       </map:match>
   </map:pipeline>

</map:pipelines>
```

where

- *src* — the root directory to start scanning.
- *depth* — (optional) the recurse depth for the directory scan (defaults to 1, the requested directory only).
- *dateFormat* — (optional) the format of the date string (follows the PHP format).
- *include* — (optional) a regular expression (Perl) defining what files/directories to include in the scan.
- *exclude* — (optional) a regular expression (Perl) defining what files/directories to exclude in the scan.
- *reverse* — (optional) `true`/`false`(default) to determine the direction of the sort.

The example above injects the following typical XML into the pipeline (taken from the Paloose directory):

```
<dir:directory
      name="/Library/Apache2/htdocs/pp/resources"
      lastmodified="April 02 2007 11:24:09"
      reverse="false"
      requested="true"
      size="306"
      xmlns:dir="http://apache.org/cocoon/directory/2.0">
   <dir:directory name="/Library/Apache2/htdocs/pp/resources/transforms"
      lastmodified="April 23 2007 10:21:06"
      size="1428">
      <dir:file name="xml2rss.xsl"
         lastmodified="April 02 2007 11:24:11" size="3330"/>
      <dir:file name="xml2xhtml.xsl"
         lastmodified="April 02 2007 11:24:11" size="8018"/>
      ...
      <dir:file name="admin-buildAuthenticateDOM.xsl"
         lastmodified="April 02 2007 11:24:11" size="3322"/>
      <dir:file name="admin-addUser.xsl"
         lastmodified="April 02 2007 11:24:11" size="2606"/>
   </dir:directory>
   ...
   <dir:directory name="/Library/Apache2/htdocs/pp/resources/images"
      lastmodified="April 02 2007 11:24:13" size="1258">
      <dir:file name="wcag1AA.png" lastmodified="April 02 2007 11:24:12" size="2288"/>
```

```
        <dir:file name="w3c_ab.png" lastmodified="April 02 2007 11:24:12" size="1296"/>
        <dir:file name="vcss.png" lastmodified="April 02 2007 11:24:12" size="1134"/>
        ...
        <dir:file name="RSS.gif" lastmodified="April 02 2007 11:24:13" size="451"/>
        <dir:file name=".DS_Store" lastmodified="December 03 2006 19:47:42" size="6148"/>
    </dir:directory>
</dir:directory>
```

where

- *name* — the name of the directory/file.
- *lastmodified* — when the directory/file was last modified (format according to *dateFormat*).
- *requested* — always `true`.
- *size* — size of the directory/file in bytes.
- *reverse* — true/false(default) showing the direction of the sort.

15 GedComGenerator

This generator reads a file in GEDCOM 5.5 format and injects a simplified XML version of it into the pipeline. It does not try to change it into the GEDCOM 6 XML representation. If that is required then a suitable transformer must be written (currently not supplied with Paloose). A typical use of *GedComGenerator* would be:

```
<map:generators default="file">
   <map:generator name="file" src="resource://lib/generation/GedComGenerator"/>
   ...
</map:generators>

<map:pipelines>

   <map:pipeline internal-only="true">

       <map:match pattern="ged.xml">
         <map:generate src="context://content/data/Field-Richards.ged"
               type="gedcom"
               label="xml-content">
            <map:parameter name="generateXMLFile" value="context://cache/Field-Richards.xml"/>
            <map:parameter name="generateDOM" value="yes"/>
         </map:generate>
         <map:serialize/>
      </map:match>

   </map:pipeline>

</map:pipelines>
```

Which injects the GedCom file `content/gallery.xml` in the main site directory (context) into the pipeline as a DOM and where

- *generateXMLFile* — (optional) the name of a file in which to store an XML representation of the GedCom input.
- *generateDOM* — (optional, default = "yes") whether to generate the DOM from the GedCom file.
- *useXMLFile* — (optional) the file to use if the DOM is not generated.

These parameters give a crude caching function as the generation process takes a little while. It also gives the opportunity for the translation from GecCom to XML to be done externally.

Warning

After Version 1.3.0 the above parameters will be deprecated and the generator component caching scheme used instead.

15 Simple example

The following is output from the Heredis application:

```
0 HEAD
1 SOUR HEREDIS 7 PC
2 VERS MAC X 10.0
2 CORP bsd concept
3 WEB www.heredis.com
1 DATE 14 OCT 2008
1 GEDC
2 VERS 5.5
2 FORM LINEAGE-LINKED
```

```
1 CHAR MACINTOSH
1 PLAC
2 FORM Town, Area code, County, Region, Country, Subdivision
1 SUBM @SO@
0 @SO@ SUBM
1 NAME Hugh Field-Richards
1 ADDR xxxxxxxxxxxx
2 CONT xxxxxxxxxxxx
2 CONT xxxxxxxxxxxx
1 EMAIL hsfr@hsfr.org.uk
...
```

which will produce the following XML:

```
<?xml version="1.0"?>
<gedcom xmlns:g="http://gedcom.org/dtd/gedxml55.dtd">
   <g:head>
      <g:sour data="HEREDIS 7 PC">
         <g:vers data="MAC X 10.0"/>
         <g:corp data="bsd concept ">
            <g:web data="www.heredis.com"/>
         </g:corp>
      </g:sour>
      <g:date data="10 SEP 2008"/>
      <g:gedc>
         <g:vers data="5.5"/>
         <g:form data="LINEAGE-LINKED"/>
      </g:gedc>
      <g:char data="MACINTOSH"/>
      <g:plac/>
      <g:subm ref="@SO@"/>
   </g:head>
   <g:subm id="@SO@">
      <g:name data="Hugh Field-Richards"/>
      <g:addr data="xxxxxxxxxxxx">
         <g:cont data="xxxxxxxxxxxx"> </g:cont>
         <g:cont data="xxxxxxxxxxxx"> </g:cont>
      </g:addr>
      <g:email data="hsrf@hsfr.org.uk"/>
   </g:subm>
   ...
```

16 PXTemplateGenerator

The *PXTemplateGenerator* generator reads an XML file and is identical to FileGenerator except for one crucial difference: embedded Paloose sitemap variables are expanded. These variables take the form "{*module_name*:*variable_name*}". Like any other generator they should always be the first item of a pipeline. So a typical use of *PXTemplateGenerator* would be:

```
<map:generators default="file">
   <map:generator name="px" src="resource://lib/generation/PXTemplateGenerator"/>
   ...
</map:generators>

<map:pipelines>

   <map:pipeline>

     <map:match pattern="**.xml">
         <map:generate type="px" src="context://admin/{1}.xml" label="xml-content"/>
         ...
     </map:match>

   </map:pipeline>
</map:pipelines>
```

16 Simple Example

One case where this is very useful is displaying session information, for example personalising an admin session. If the XML file contains:

```
<t:heading level="1">Welcome, {session:__username}, to the Paloose Admin Page</t:heading>
```

Assuming the session variable "*__username*" contains "*hsfr*" the following will be output into the pipeline:

```
<t:heading level="1">Welcome, hsfr, to the Paloose Admin Page</t:heading>
```

Note

In versions after 1.3.4 implicit entities defined by ISOlat1, ISOpub and ISOnum are translated into their numerical form. The *EntityTransformer* now only handles explicitly declared entities within a `DOCTYPE` statement within the XML file.

16 Caching Support (available after Version 1.3.0)

The *PXTemplateGenerator* component supports caching of the data within the pipeline. The input file is normally checked for well-formedness (no schema validation) but caching the data stores the XML after this process. As a result very little time is saved except on very large XML source files.

Warning

Note that if you have an XML file that includes other files (via xinclude), the generator caching will not detect that these other XML files have been modified. If you modify them it is important to clear out the caching to let the cache files to be rebuilt.

Enabling caching is done globally and locally. To turn on caching for all *PXTemplateGenerators* there is an attribute *cachable* that should be set to true (true/yes/1). So setting the global flag would be (default is false):

```
<map:generators default="file">
   <map:generator src="resource://lib/generation/PXTemplateGenerator"
      name="px"
      cachable="yes"/>
</map:generators>
```

This can be overriden by using the same attribute when the pipeline component is used:

```
   <map:match pattern="*.xml">
      <map:generate src="context://content/{1}.xml"
         label="xml-content"
         type="px"
         cachable="no"/>
      <map:serialize/>
   </map:match>
```

The above would turn off the caching for just this instance in the pipeline. It is also possible to change the action of the cache via the query string by using the request parameters. The following would control the generator instance by including the query string "`http://[host]/[path-file-name]?cachable=1`" or "`http://[host]/[path-file-name]?cachable=0`"

```
<map:generators default="file">
   <map:generator src="resource://lib/generation/PXTemplateGenerator"
      name="px"
      cachable="no">
      <map:use-request-parameters>true</map:use-request-parameters>
   </map:generator>
</map:generators>

...

<map:match pattern="index.xml">
   <map:generate src="context://content/index.xml"
      type="px"
      label="index-xml"
      cachable="{request-param:cachable}"/>
   <map:serialize type="xml"/>
</map:match>
```

17 Transformers

Transformers take the pipeline data as a DOM document, transform it into another DOM and output back into the pipeline. There can be as many transformer as taste and performance will allow (I think that the maximum I ever had in a Cocoon pipeline was 12, and all of them necessary — honest).

- TRAXTransformer — allows processing of the DOM according to a specified XSLT file.
- PageHitTransformer — inserts the number of page hits for the page that is being processed.
- GalleryTransformer — manages a photo gallery system.
- I18nTransformer — supports internationalisation.
- SourceWritingTransformer — allows the pipeline XML to be diverted to an external file.
- FilterTransformer — allows restricted numbers of named tags to be passed.
- SQLTransformer — allows queries to be made to a SQL database.
- XIncludeTransformer — allows external documents (or parts of documents) to be melded into the document traversing the pipeline.
- EntityTransformer — translates user and standard entities embedded within the document.
- ModuleWriteTransformer — updates the module variables from within the data stream.
- PasswordTransformer — provides a simple password encoding (based on md5).
- LogTransformer — provides a simple method of logging XML fragments from within the pipeline.

17 Component Declarations

Transformers are defined in the component declaration part of the Sitemap. For example

```
<map:transformers default="xslt">
   <map:transformer name="xslt" src="resource://lib/transforming/TRAXTransformer">
      <map:use-request-parameters>true</map:use-request-parameters>
   </map:transformer>
   <map:transformer name="pageHit" src="resource://lib/transforming/PageHitTransformer">
      <map:parameter name="file" value="context://logs/PageHit.cnt"/>
      <map:parameter name="unique" value="false"/>
      <map:parameter name="cookie-name" value="PalooseTextHit"/>
      <map:parameter name="ignore" value="127.0.0.1"/>
   </map:transformer>
   <map:transformer name="i18n" src="resource://lib/transforming/I18nTransformer">
      <map:catalogues default="index">
         <map:catalogue id="index" name="index" location="context://content/translations"/>
      </map:catalogues>
      <map:untranslated-text>untranslated text</map:untranslated-text>
   </map:transformer>
   <map:transformer name="gallery" src="resource://lib/transforming/GalleryTransformer">
      <map:parameter name="root" value="context://gallery/"/>
      <map:parameter name="image-cache" value="context://resources/images/cache/"/>
      <map:parameter name="max-thumbnail-width" value="150"/>
      <map:parameter name="max-thumbnail-height" value="150"/>
      <map:parameter name="resize" value="1"/>
      <map:parameter name="max-width" value="600"/>
      <map:parameter name="max-height" value="600"/>
   </map:transformer>
   <map:transformer name="log" src="resource://lib/transforming/LogTransformer"/>
   <map:transformer name="xinclude" src="resource://lib/transforming/XIncludeTransformer"/>
   <map:transformer name="password" src="resource://lib/transforming/PasswordTransformer"/>
   <map:transformer name="write-source" src="resource://lib/transforming/SourceWritingTransformer" />
</map:transformers>
```

The *default* attribute specifies the type of serializer to use if none is specified in a pipeline.

18 TraxTransformer

XSL Transformers are the heart of any Paloose system (and Cocoon). At minimum they are the means for turning XML content into displayable HTML. They must not be first or last in the pipeline as they take and return DOM information in the pipe. A typical use of TRAXTransformer would be

```
<map:transformers default="xslt">
   <map:transformer name="xslt" src="resource://lib/transforming/TRAXTransformer">
      <map:use-request-parameters>true</map:use-request-parameters>
   </map:transformer>

<map:pipelines>
   <map:pipeline>

      <map:match pattern="news-rss.xml">
         <map:generate src="context://content/newsArticles.xml"/>
         <transform src="context://resources/transforms/xml2rss.xsl"/>
         <map:serialize/>
      </map:match>

   </map:pipeline>
</map:pipelines>
```

which would process the file "`newsArticles.xml`" into RSS format using the XSL file "`xml2rss.xsl`.

18 Caching Support (avaliable after Version 1.3.0)

The *TraxTransformer* component supports caching of the data within the pipeline. The transformer takes several inputs that determine whether the pipeline data cache can be used: the state of the transformation file (XSL), the input DOM from the previous stage, and the parameters (have they changed).

Warning

Note that if you have an XSL file that includes other stylesheets, the transformer caching will not detect that these other file sheets have been modified. If you modify them it is important to clear out the caching to let the cache files to be rebuilt.

Enabling caching is done globally and locally. To turn on caching for all *TraxTransformers* there is an attribute *cachable* that should be set to true (true/yes/1). So setting the global flag would be (default is false):

```
<map:transformers default="xslt">
   <map:transformer src="resource://lib/transforming/TRAXTransformer"
      name="xslt"
      cachable="no"/>
   ...
</map:transformers>
```

This can be overriden by using the same attribute when the pipeline component is used:

```
<map:transform src="context://resources/transforms/page2html.xsl" cachable="no">
   <map:parameter name="page" value="{1}"/>
</map:transform>
```

The above would turn off the caching for just this instance in the pipeline. It is also possible to change the action of the cache via the query string by using the request parameters. The following would control the generator instance by including the query string "`http://[host]/[path-file-name]?cachable=1`" or "`http://[host]/[path-file-name]?cachable=0`".

```
<map:transformers default="xslt">
   <map:transformer src="resource://lib/transforming/TRAXTransformer"
         name="xslt"
         cachable="no">
      <map:use-request-parameters>true</map:use-request-parameters>
   </map:transformer>
   ...
</map:transformers>

...

<map:transform src="context://resources/transforms/page2html.xsl"
      cachable="{request-param:cachable}">
   <map:parameter name="page" value="{1}"/>
</map:transform>
```

19 XIncludeTransformer

The *XIncludeTransformer* provides a means of including other XML fragments into the XML file being processed. It follows the XInclude specification. It is possible to include XML or text in several ways:

- Include an entire XML file,
- Include part of an entire XML file, or
- Include straight text.

The *XIncludeTransformer* component is declared and used in the sitemap as:

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

   <map:components>

      <map:transformers default="xslt">
         <map:transformer name="xinclude" src="resource://lib/transforming/XIncludeTransformer"/>
         ...
      </map:transformers>

   </map:components>

   <map:pipelines>

      <map:pipeline>

         <map:match pattern="**.html">
            <map:generate src="context://content/index.xml" label="index-xml"/>
            <map:transform type="xinclude"/>
            ...
         </map:match>

      </map:pipeline>
```

19 Simple Example

Within the source XML include statements are entered simply as this folowing example shows. The example consists of a source file:

```
<?xml version="1.0" encoding="UTF-8"?>
<page:page xmlns:xi="http://www.w3.org/2001/XInclude"
   xmlns:t="http://www.hsfr.org.uk/Schema/Text"
   xmlns:page="http://www.hsfr.org.uk/Schema/Page">

   <page:meta>
      ...
   </page:meta>

   <page:content>

      <t:heading level="1">XInclude Test</t:heading>

      <xi:include href="part-1.xml">
         <xi:fallback>
            <t:p>Error.</t:p>
         </xi:fallback>
      </xi:include>

      <xi:include href="part-1.xml#element(/1/2/1)" parse="xml">
         <xi:fallback>
            <t:p>Error.</t:p>
         </xi:fallback>
      </xi:include>
```

```
      <xi:include
               href="part-2.xml#xmlns(page=http://www.hsfr.org.uk/Schema/Page)xpointer(//page:content/*)"
               parse="xml">
         <xi:fallback>
            <t:p>Error.</t:p>
         </xi:fallback>
      </xi:include>

   </page:content>

</page:page>
```

and three included files:

part-1.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<page:page  xmlns:t="http://www.hsfr.org.uk/Schema/Text" xmlns:page="http://www.hsfr.org.uk/Schema/Page">

   <page:meta>
      ...
   </page:meta>

   <page:content>
      <t:p>First paragraph</t:p>
   </page:content>
</page:page>
```

part-2.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<page:page  xmlns:t="http://www.hsfr.org.uk/Schema/Text" xmlns:page="http://www.hsfr.org.uk/Schema/Page">

   <page:meta>
      ...
   </page:meta>

   <page:content>
      <t:p>Second paragraph</t:p>
   </page:content>
</page:page>
```

part-3.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<page:page  xmlns:t="http://www.hsfr.org.uk/Schema/Text" xmlns:page="http://www.hsfr.org.uk/Schema/Page">

   <page:meta>
      ...
   </page:meta>

   <page:content>
      <t:p>Third paragraph</t:p>
   </page:content>
</page:page>
```

After the transformer has run the document will be:

```
<page:page
      xmlns:page="http://www.hsfr.org.uk/Schema/Page"
      xmlns:t="http://www.hsfr.org.uk/Schema/Text">
   <page:meta>
      ...
   </page:meta>
   <page:content>
      <t:heading level="1">XInclude Test</t:heading>
      <page:page>
         <page:meta>
            ...
         </page:meta>
         <page:content>
```

```
            <t:p>First paragraph</t:p>
         </page:content>
      </page:page>
      <page:content>
         <t:p>Second paragraph</t:p>
      </page:content>
      <t:p>Second paragraph</t:p>
   </page:content>
</page:page>
```

20 EntityTransformer

The *EntityTransformer* provides a mechanism of expanding Entities within the DOM. It is declared and used as

```
    <map:transformers default="xslt">
        <map:transformer name="xslt" src="resource://lib/transforming/TRAXTransformer">
           <map:use-request-parameters>true</map:use-request-parameters>
        </map:transformer>
        <map:transformer name="ent" src="resource://lib/transforming/EntityTransformer"/>
     </map:transformers>

     ...

     <map:pipeline>
         <map:match pattern="**.html">
           <map:generate src="context://content/{1}.xml" label="content-xml"/>
           <map:transform type="ent"/>
           <map:transform src="context://resources/transforms/page2html.xsl" label="transform-xml"/>
           <map:serialize type="xhtml"/>
        </map:match>
```

Note

In versions after 1.3.4 implicit entities defined by ISOlat1, ISOpub and ISOnum are translated into their numerical form by the appropriate generator, *FileGenerator* and *PXTemplateGenerator*. The *EntityTransformer* now only handles explictly declared entities within a `DOCTYPE` statement within the XML file.

As an example using the sitemap above consider the following XHTML input document:

test-1.xml

```
<?xml version="1.0"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd"
[
    <!ENTITY testTitle    "Entity Test 1" >
] >

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
   <head>
      <title>Entity Test 1</title>
      <style>
        body  { background-color: white; color: black; font-family: monospace; font-size: 14pt; }
         td   { text-align: center; }
      </style>
   </head>
   <body>
      <h1>&testTitle;</h1>
      <table border="1" width="200px">
         <tr><th>Name</th><th>Decimal</th><th>Name</th></tr>
         <tr><td>lt</td><td>&#60;</td><td>&lt;</td></tr>
         <tr><td>gt</td><td>&#62;</td><td>&gt;</td></tr>
         <tr><td>ndash</td><td>&#x2013;</td><td>&ndash;</td></tr>
         <tr><td>mdash</td><td>&#x2014;</td><td>&mdash;</td></tr>
         <tr><td>oslash</td><td>&#248;</td><td>&oslash;</td></tr>
         <tr><td>ccedil</td><td>&#231;</td><td>&ccedil;</td></tr>
         <tr><td>frac13</td><td>&#x2153;</td><td>&frac13;</td></tr>
         <tr><td>ltri</td><td>&#x25C3;</td><td>&ltri;</td></tr>
      </table>
   </body>
</html>
```

All the implicit entities `lt`, `gt`, `ndash`, `mdash`, `oslash`, `ccedil`, `frac13` and `ltri`, are handled by the generator pipeline component. Only the explicit entitiy, `testTitle` is handled by the *EntityTransformer*.

Running the above through a suitable serializer will display as:

Entity Test 1

Name	Decimal	Name
lt	<	<
gt	>	>
ndash	–	–
mdash	—	—
oslash	ø	ø
ccedil	ç	ç
frac13	⅓	⅓
ltri	◁	◁

21 SourceWritingTransformer

The Source Writing Transformer is very similar to the Cocoon version. It provides a means to divert XML from the pipeline into an external file. It can also add or delete fragments within the file. There are three tags that form the *SourceWritingTransformer* framework:

- *source:write*
- *source:insert*
- *source:delete*

21 Source Writing Namespace

The *SourceWritingTransformer* tags exist in their own namespace, which is the same as the Cocoon transformer: "`http://apache.org/cocoon/source/1.0`".

21 Source Writing Output

The transformer replaces the tags within the document with the results of the operation. The generalised output (identical to Cocoon) is:

```
<source:sourceResult>
   <source:action>new|overwritten|none</source:action>
   <source:behaviour>write|insert<source:behaviour>
   <source:execution>success|failure</source:execution>
   <source:serializer>xml</source:serializer>
   <source:source>Full file name of processed file</source:source>
   <source:message>a message about what happened</source:message>
</source:sourceResult>
```

21 Source Write

Source write tags allow the writing of a complete file to a folder. The overall structure is:

```
<source:write [create="true"]">
   <source:source/>
   [<source:path/>]
   <source:fragment/>
 </source:write>
```

where

- *create* attribute — defines whether to create the file first if it does not exist.
- *source:source* — is the file name of the file to be written. It can have pseudo variables such as "`context://`".
- *source:path* — is an optional XPath for defining the root structure of the file with which the fragment is placed.
- *source:fragment* — is the fragment of XML that will be written.

Note that there is no serializer attribute that is present in Cocoon. For example the following structure:

```
<source:write create="true">
   <source:source>context://test.xml</source:source>
   <source:path>/root/AAA</source:path>
   <source:fragment>
      <BBB>
```

```
         <CCC name="bob"/>
      </BBB>
   </source:fragment>
</source:write>
```

will write the following file (I have added indentation for clarity)

context://test.xml

```
<?xml version="1.0"?>
<root>
   <AAA>
      <BBB>
         <CCC name="bob"/>
      </BBB>
   </AAA>
</root>
```

Note

If you omit *source:path* it is important that the XML within *source:fragment* has only a single node, which will become the root node of the written document.

If you do not as in this example:

```
<source:write create="true">
   <source:source>context://configs/test.xml</source:source>
   <source:path/>
   <source:fragment>
      <BBB/>
      <CCC name="bob"/>
   </source:fragment>
</source:write>
```

then the following error is returned

```
<source:sourceResult>
   <source:action>new</source:action>
   <source:behaviour>write<source:behaviour>
   <source:execution>failure</source:execution>
   <source:serializer>xml</source:serializer>
   <source:source>/...../configs/test.xml</source:source>
   <source:message>Problem processing source document in write-source:
         Fragment must have one root element if no path declared</source:message>
</source:sourceResult>
```

21 Source Insert

Source write tags allow the writing of a complete file to a folder. The overall structure is:

```
<source:insert [create="true"]">
   <source:source/>
   <source:path/>
   <source:fragment/>
   [<source:replace/>]
</source:insert>
```

where

- *create* attribute — defines whether to create the file first if it does not exist.
- *source:source* — is the file name of the file to be written. It can have pseudo variables such as "`context://`".
- *source:path* — is an optional XPath for defining the root structure of the file with which the fragment is placed.
- *source:fragment* — is the fragment of XML that will be written.
- *source:replace* — an optional XPath to select the node that is to be replaced by the XML fragment.

Note

Note that there is no @serializer attribute that is present in Cocoon. Note also that the Cocoon *source:reinsert* tag is not used — I confess that I did not understand exactly what was required here and so it seemed to be safe to leave it out.

Assume that we have a file that has been created above:

context://test.xml

```
<?xml version="1.0"?>
<root>
   <AAA>
      <BBB>
         <CCC name="bob"/>
      </BBB>
   </AAA>
</root>
```

The *source:insert* comes in several flavours:

21 Case 1 (replace not specified)

```
<source:insert create="true">
      <source:source>context://configs/test.xml</source:source>
      <source:path>/root/AAA</source:path>
      <source:fragment>
         <BBB/>
            <CCC name="alice"/>
         </BBB>
      </source:fragment>
   </source:insert>
```

will append the fragment as a child of `/root/AAA`:

context://test.xml

```
<?xml version="1.0"?>
<root>
   <AAA>
      <BBB>
         <CCC name="bob"/>
      </BBB>
      <BBB>
         <CCC name="alice"/>
      </BBB>
   </AAA>
</root>
```

21 Case 2 (replace specified, node exists, overwrite true)

```
<source:insert overwrite="true">
      <source:source>context://configs/test.xml</source:source>
```

```
      <source:path>/root/AAA</source:path>
      <source:replace>BBB/CCC[ @name='alice' ]/parent::*</source:replace>
      <source:fragment>
         <BBB>
            <CCC name="carol"/>
         </BBB>
      </source:fragment>
   </source:insert>
```

will replace the second *BBB* node with the fragment:

context://test.xml

```
<?xml version="1.0"?>
<root>
   <AAA>
      <BBB>
         <CCC name="bob"/>
      </BBB>
      <BBB>
         <CCC name="carol"/>
      </BBB>
   </AAA>
</root>
```

21 Case 3 (replace specified, overwrite false)

```
<source:insert overwrite="false">
      <source:source>context://configs/test.xml</source:source>
      <source:path>/root/AAA</source:path>
      <source:replace>BBB/CCC[ @name='alice' ]/parent::*</source:replace>
      <source:fragment>
         <BBB>
            <CCC name="carol"/>
         </BBB>
      </source:fragment>
   </source:insert>
```

causes no action to be taken.

21 Case 4 (replace specified, node does not exist, overwrite true or false)

```
<source:insert>
      <source:source>context://configs/test.xml</source:source>
      <source:path>/root/AAA</source:path>
      <source:replace>BBB/CCC[ @name='oscar' ]/parent::*</source:replace>
      <source:fragment>
         <BBB>
            <CCC name="carol"/>
         </BBB>
      </source:fragment>
   </source:insert>
```

will replace the second *BBB* node with the fragment:

context://test.xml

```
<?xml version="1.0"?>
<root>
   <AAA>
      <BBB>
         <CCC name="bob"/>
      </BBB>
      <BBB>
         <CCC name="alice"/>
```

```
      </BBB>
      <BBB>
         <CCC name="carol"/>
      </BBB>
   </AAA>
</root>
```

21 Source Delete

Source delete tags delete the specified source file. The overall structure is:

```
<source:insert>
   <source:source/>
</source:insert>
```

where

- *source:source* — is the file name of the file to be deleted. It can have pseudo variables such as "`context://`".

For example:

```
<source:delete>
   <source:source>context://configs/test.xml</source:source>
</source:delete>
```

22 SQLTransformer

Warning

Note that this does not match the Cocoon method 100%. There are important differences that are discussed in below.

SQL Transformers provide an interface between Paloose and SQL-savvy database engines. They take a query and return the results of that query to the pipeline (or a suitable error message). A typical declaration of the SQLTransformer component would be:

```
<map:transformers default="xslt">
   <map:transformer name="mysql" src="resource://lib/transforming/SQLTransformer">
      <map:parameter name="type" value="mysql"/>
      <map:parameter name="host" value="localhost:3306"/>
      <map:parameter name="user" value="root"/>
      <map:parameter name="password" value="*******"/>
   </map:transformer>
   <map:transformer name="xslt" src="resource://lib/transforming/TRAXTransformer">
      <map:use-request-parameters>true</map:use-request-parameters>
   </map:transformer>
</map:transformers>
```

where

- *type* — the type of this database. In this case it is a MySQL database. (At present this is the only value and will be extended in future versions.
- *host* — the host where the database server is running.
- *user* — the database login user.
- *password* — the password associated with this user.

The pipeline would then be:

```
<map:pipeline>

   <map:match pattern="**.html">
      <map:generate src="context://content/{1}.xml" label="xml-content"/>
      <map:transform type="mysql" label="sql-transform">
         <map:parameter name="show-nr-of-rows" value="true"/>
      </map:transform>
      ...
   </map:match>
```

where

- *show-nr-of-rows* — is a flag to determine whether the number of rows found is output with the result (true or false).

22 Errors

Errors from the database engine are reported in the following typical fashion

```
<page:content xmlns:default="http://apache.org/cocoon/SQL/2.0"
         xmlns:t="http://www.hsfr.org.uk/Schema/Text">
   <t:heading level="1">SQL Transform Test</t:heading>
   <default:sql-error xmlns="http://apache.org/cocoon/SQL/2.0">
      <default:host>localhost:3306</default:host>
```

```
      <default:user>root</default:user>
      <default:password></default:password>
      <default:message>SQL query error: => query: select * fom composer </default:message>
   </default:sql-error>
</page:content>
```

22 Using the SQLTransformer

Warning

Note that this does not match the Cocoon method 100%. There are important differences that are indicated below.

The SQL Transformer provides a link between the sitemap and user's site and a database using SQL queries. For this example I am going to assume the following database on a MySQL database on a local machine being accessed by user "root". The database has the following form:

```
mysql> use music;
Database changed
mysql> describe composer;
+-----------+-------------+------+-----+---------+-------+
| Field     | Type        | Null | Key | Default | Extra |
+-----------+-------------+------+-----+---------+-------+
| name      | varchar(40) | YES  |     | NULL    |       |
| forenames | varchar(40) | YES  |     | NULL    |       |
| birth     | date        | YES  |     | NULL    |       |
| death     | date        | YES  |     | NULL    |       |
+-----------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)

mysql> select * from composer;
+------------+------------------+------------+------------+
| name       | forenames        | birth      | death      |
+------------+------------------+------------+------------+
| Mozart     | Wolfgang Amadeus | 1756-01-27 | 1791-12-05 |
| Beethoven  | Ludvig van       | 1770-12-15 | 1827-03-26 |
| Bach       | Johann Sebastian | 1685-03-21 | 1750-07-28 |
| Bach       | Johann Christian | 1735-09-05 | 1782-01-01 |
| Haydn      | Franz Joseph     | 1732-03-31 | 1809-05-31 |
| Bernstein  | Leonard          | 1918-08-25 | 1990-10-14 |
| Boccherini | Luigi            | 1743-02-19 | 1805-05-28 |
| Ravel      | Joseph Maurice   | 1875-03-07 | 1937-12-28 |
+------------+------------------+------------+------------+
8 rows in set (0.00 sec)

mysql>
```

Very simple but it will suffice.

22 Sitemap

The root sitemap needs to have the SQL Transformer declared as a component:

```
<map:components>
     <map:transformers default="xslt">
        <map:transformer name="mysql" src="resource://lib/transforming/SQLTransformer">
           <map:parameter name="type" value="mysql"/>
           <map:parameter name="host" value="localhost:3306"/>
           <map:parameter name="user" value="root"/>
           <map:parameter name="password" value="xxxxxxx"/>
```

```
        </map:transformer>

        ...
      </map:transformers>
```

where

- **type** — the type of this database. In this case it is a MySQL database. (At present this is the only value and will be extended in future versions.
- **host** — the host where the database server is running.
- **user** — the database login user.
- **password** — the password associated with this user.

The sitemap that we will use for the following examples has a pipeline:

```
<map:match pattern="**.html">
   <map:generate src="context://content/{1}.xml" label="xml-content"/>
   <map:transform type="mysql" label="sql-transform">
      <map:parameter name="show-nr-of-rows" value="true"/>
   </map:transform>
   ...
</map:match>
```

where

- **show-nr-of-rows** — is a flag to determine whether the number of rows found is output with the result (true or false).

22 A simple query

Say we wished to get a list of all composers named "Bach" and output their details. First of all we need to form a XML query in the input file. This has the general form:

```
<execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
   <query>
      <!-- The SQL query statement -->
   </query>
</execute-query>
```

All very simple. So a real example to query the composers would be (using the samme XML form as these pages):

```
<page:content>

   <t:heading level="1">SQL Transform Test</t:heading>

     <execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
      <query name="displayAllBach" database="music">
           select * from composer
           where name = "Bach"
      </query>
     </execute-query>

</page:content>
```

where the attributes of *query* are:

- **name** — the name of this query which will be used to tag the returned data,
- **database** — the name of the database being queried.

Warning

Note that the quary attributes have change after Version 1.3.5. The old *name* attribute is now the *database* attribute

The results are displayed in the form of each row that matches the criteria:

```
<page:content xmlns:default="http://apache.org/cocoon/SQL/2.0"
   <t:heading level="1">SQL Transform Test</t:heading>
   <default:row-set nrofrows="2" name="displayAllBach" xmlns="http://apache.org/cocoon/SQL/2.0">
      <default:row>
         <default:name>Bach</default:name>
         <default:forenames>Johann Sebastian</default:forenames>
         <default:birth>1685-03-21</default:birth>
         <default:death>1750-07-28</default:death>
      </default:row>
      <default:row>
         <default:name>Bach</default:name>
         <default:forenames>Johann Christian</default:forenames>
         <default:birth>1735-09-05</default:birth>
         <default:death>1782-01-01</default:death>
      </default:row>
   </default:row-set>
</page:content>
```

It is then up to the user to provide the correct XSL transform to process this information.

22 A simple query with substitution.

It is sometimes useful to have information put into the query at run time. For example a user name from login, or possibly a selection critera from the query request string in the URL. For example taking the query above say we wanted to select the composer name from the query, we would present the query as:

```
http://localhost/...?composer=Bach
```

and rewrite the XML file as

```
<page:content>

   <t:heading level="1">SQL Transform Test</t:heading>

     <execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
      <query name="displayQuery" database="music">
           select * from composer
           where name = "{request-param:composer}"
      </query>
     </execute-query>

</page:content>
```

which would give the same result as the first example. It is also possible to input information from the sitemap:

```
<map:match pattern="**.html">
   <map:generate src="context://content/{1}.xml" label="xml-content"/>
   <map:transform type="mysql" label="sql-transform">
      <map:parameter name="show-nr-of-rows" value="true"/>
```

```
        <map:parameter name="composer" value="Bach"/>
    </map:transform>
    <map:transform src="context://resources/transforms/xml2xhtml.xsl"/>
    <map:serialize type="html"/>
</map:match>
```

with a query as follows:

```
<page:content>

   <t:heading level="1">SQL Transform Test</t:heading>

     <execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
      <query name="displayQuery" database="music">
           select * from composer
           where name = "{param:composer}"
      </query>
     </execute-query>

</page:content>
```

Warning

Note that Paloose does not follow the Cocoon scheme. The latter uses an embedded tag structure. I may change this in future if it proves to be a problem.

23 FilterTransformer

Sometimes it is necessary to restrict the number of tags within a block. This is particular relevant to SQL results which are returned as a set of rows. A typical use of FilterTransformer would be

```
<map:transformers default="xslt">
    <map:transformer name="xslt" src="resource://lib/transforming/TRAXTransformer">
       <map:use-request-parameters>true</map:use-request-parameters>
    </map:transformer>
    <map:transformer name="mysql" src="resource://lib/transforming/SQLTransformer">
       <map:parameter name="type" value="mysql"/>
       <map:parameter name="host" value="localhost:3306"/>
       <map:parameter name="user" value="root"/>
    </map:transformer>
    <map:transformer name="filter" src="resource://lib/transforming/FilterTransformer"/>
 </map:transformers>

 ...

 <map:pipeline>

    <map:match pattern="**.html">
       <map:generate src="context://content/{1}.xml" label="xml-content"/>
       <map:transform type="mysql" label="sql-transform">
          <map:parameter name="show-nr-of-rows" value="true"/>
          <map:parameter name="composer" value="Bach"/>
       </map:transform>
       <map:transform type="filter">
          <map:parameter name="element-name" value="http://apache.org/cocoon/SQL/2.0:row"/>
          <map:parameter name="count" value="2"/>
          <map:parameter name="blocknr" value="3"/>
       </map:transform>
       ...
    </map:match>
```

where

- *element-name* — the name of the tag which will be restricted. It can either be a tag with or without a namespace. However the dec;aration must match what is in the document.
- *count* — the size of the blocks.
- *blocknr* — the block number that is required.

Say the following data is stored in the database:

```
+------------+------------------+------------+------------+
| name       | forenames        | birth      | death      |
+------------+------------------+------------+------------+
| Mozart     | Wolfgang Amadeus | 1756-01-27 | 1791-12-05 |
| Beethoven  | Ludvig van       | 1770-12-15 | 1827-03-26 |
| Bach       | Johann Sebastian | 1685-03-21 | 1750-07-28 |
| Bach       | Johann Christian | 1735-09-05 | 1782-01-01 |
| Haydn      | Franz Joseph     | 1732-03-31 | 1809-05-31 |
| Bernstein  | Leonard          | 1918-08-25 | 1990-10-14 |
| Boccherini | Luigi            | 1743-02-19 | 1805-05-28 |
| Ravel      | Joseph Maurice   | 1875-03-07 | 1937-12-28 |
+------------+------------------+------------+------------+
```

Then the above filter instance (*count*=2 and *blocknr*=3) would return from a query "*select * from composer*", the following XML:

```
<page:content xmlns:default="http://apache.org/cocoon/SQL/2.0"
    xmlns:t="http://www.hsfr.org.uk/Schema/Text">
    <t:heading level="1">SQL Transform Test</t:heading>
    <default:row-set nrofrows="8" name="music">
```

```
      <default:block id="1"/>
      <default:block id="2"/>
      <default:block id="3">
         <default:row>
            <default:name>Haydn</default:name>
            <default:forenames>Franz Joseph</default:forenames>
            <default:birth>1732-03-31</default:birth>
            <default:death>1809-05-31</default:death>
         </default:row>
         <default:row>
            <default:name>Bernstein</default:name>
            <default:forenames>Leonard</default:forenames>
            <default:birth>1918-08-25</default:birth>
            <default:death>1990-10-14</default:death>
         </default:row>
      </default:block>
      <default:block id="4"/>
   </default:row-set>
</page:content>
```

24 ModuleWriteTransformer

The *ModuleWriteTransformer* updates the module variables from code within the input stream. A typocal use in the XML data stream would be

```
<paloose:module name="cookies">
   <paloose:param name="theme" value="myTheme"/>
</paloose:module>
```

Note that the above code is swallowed and does not cause any XML output.

Note

This example needs expanding

25 LogTransformer

The Log Transformer allows fragments of XML to be written to an external file to aid in debugging sitemap pipelines. A typical use of *LogTransformer* would be

```
   <map:transformers default="xslt">
      <map:transformer name="log" src="resource://lib/transforming/LogTransformer"/>
      ...
   </map:transformers>

   <map:pipelines>
      <map:pipeline>

         <map:match pattern="**.html">
            ...
            <map:transform type="log">
               <map:parameter name="logfile" value="context://logs/logfile-aggr.log"/>
               <map:parameter name="append" value="yes"/>
               <map:parameter name="filter" value="//*[local-name() = 'p'][2]"/>
            </map:transform>
...
         </map:match>

      </map:pipeline>
   </map:pipelines>
```

where

- *logfile* — defines the logfile to use (must have correct permissions set).
- *append* — defines whether the log data is appended to the end of the file (yes) or a new file is created each time (no).
- *filter* — provides a means to restrict what is output to the log file using an XPath expression. For example, the example above would just output the second paragraph (p) of the XML in the pipeline at that point. Note that the expression should be "namespace-neutral", that is, only local names should be used.

The example above might give the following (from the documentation home page):

```
   ================================================================================
      <t:p>Installation is very simple &#x2014; download the latest version
         <link:link type="uri" ref="downloads/paloose-latest.tgz">here</link:link>
         and then follow <link:link type="uri" ref="documentation/install.html">these
         (brief) instructions</link:link>.</t:p>
```

26 PasswordTransformer

The Password Transformer provides a means of transforming a plain-text string into an MD5 encrypted string. A typical use of PasswordTransformer would be

```
<map:transformers default="xslt">
   <map:transformer name="password" src="resource://lib/transforming/PasswordTransformer"/>
   ...
</map:transformers>

<map:pipelines>
   <map:pipeline>

      <map:match pattern="login.xml">
         ...
         <transform type="password"/>
         ...
      </map:match>

   </map:pipeline>
</map:pipelines>
```

The transformer is based on a single *authentication* tag which has the form:

```
<authentication username="hsfr" password="mypassword" />
```

After processing with the *PasswordTransformer* the tag becomes:

```
<authentication username="hsfr" password="h7fdT71xxxxxxxxxxxxxxxGFdgfg" />
```

27 Serializers

Serializers take the pipeline data as a DOM document and output it to the client. There are four basic types at present supported within Paloose:

- HTMLSerializer — outputs simple HTML with option of a leading `DOCTYPE` declaration and character encoding information.
- XHTMLSerializer — outputs conformant XHTML with option of a leading `DOCTYPE` declaration and character encoding information.
- TextSerializer — outputs pure text derived from the content of the pipeline DOM document.
- XMLSerializer — outputs the DOM Document as an XML stream.

27 Component Declaration

Serializers are defined in the component declaration part of the Sitemap.

```
<map:serializers default="xml">
   <map:serializer name="html" src="resource://lib/serialization/HTMLSerializer">
      <doctype-public>-//W3C//DTD HTML 4.01 Transitional//EN</doctype-public>
      <doctype-system>http://www.w3.org/TR/html4/loose.dtd</doctype-system>
      <encoding>iso-8859-1</encoding>
   </map:serializer>
   <map:serializer name="xhtml" src="resource://lib/serialization/XHTMLSerializer">
      <doctype-public>-//W3C//DTD XHTML 1.0 Strict//EN</doctype-public>
      <doctype-system>http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd</doctype-system>
      <encoding>iso-8859-1</encoding>
   </map:serializer>
   <map:serializer name="text" src="resource://lib/serialization/TextSerializer"/>
   <map:serializer name="xml" src="resource://lib/serialization/XMLSerializer"/>
</map:serializers>
```

The *default* attribute specifies the type of serializer to use if none is specified in a pipeline.

28 Actions

Actions provide a mechanism to interact with the outside world and control what path is taken within the pipeline from the results of that interaction. They take runtime parameters and utilise them in performing the action. The actions that Paloose supports are:

- SendMailAction — allows the pipeline to send mail on the basis of the data passing through the pipeline.
- Authorisation Actions — allows the pipeline to restrict use until some authorisation takes place (login etc).
- Cookies Actions — allows the pipeline access to the user's cookies (used in conjunction with ModuleWrite transformer.

28 Component Declarations

Actions are defined in the component declaration part of the Sitemap. For example

```
<map:actions>
   <map:action name="sendmail" src="resource://lib/acting/SendMailAction"/>
   <map:action name="auth-protect" src="resource://lib/acting/AuthAction"/>
   <map:action name="auth-login" src="resource://lib/acting/LoginAction"/>
   <map:action name="auth-logout" src="resource://lib/acting/LogoutAction"/>
</map:actions>
```

29 Handling Errors

Paloose error handling is similar to Cocoon but differs in some key areas. Like Cocoon each pipeline may define its own error handling.

Warning

Cocoon's error handler uses a specialized pipeline having a pre-configured generator, whereas Paloose's error handler uses a completely standard pipeline.

In Cocoon you do not define a generator inside the error handler. In Paloose you do. Error handlers are also (like Cocoon) hierarchical. Each error handler is bound to a particular pipeline. If an error occurs in a pipeline the error handler for that is invoked. If there is no handler then the previous (via mount) pipeline is offered the error. If this cannot then its previous pipeline is tried, and so on. If there are no appropriate handler then the error handler of the `<map:pipelines>` element in the root sitemap is tried. If that fails then the internal Paloose error mechanism takes over — but the user has no control of this.

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

   <map:handle-errors>
      <map:generate src="context://content/error.xml"/>
      <map:transform src="context://resources/transforms/error2html.xsl"/>
      <map:serialize/>
   </map:handle-errors>
</map:pipelines>
```

29 Example

The Paloose site uses a simple single point error handler in the root sitemap:

```
<map:pipelines>

   <map:pipeline>

      ...

      <!-- All html requests go to a subsite map for content -->
      <map:match pattern="**.html">
         <map:mount src="content/sitemap.xmap"/>
      </map:match>

      ...

      <map:handle-errors>
         <map:generate type="px" src="context://content/error.xml"/>
         <map:transform src="context://resources/transforms/page2html.xsl" label="page-transform">
            <map:parameter name="page" value="error"/>
         </map:transform>
         <map:serialize type="xhtml"/>
      </map:handle-errors>

   </map:pipeline>

</map:pipelines>
```

The error page (or at least the content part) has a simple header and single paragraph. The part to notice here is the use of the embedded sitemap variable, {*error:message*}, which contains the message associated with the error.

context://content/error.xml

```
<page:content>
   <t:heading level="1">Oops! an error occurred, please select a new page ...</t:heading>
   <t:p label="normalPara">{error:message}</t:p>
   </page:content>
```

To see how this works try the following error URL.

30 SendMailAction

The *SendMailAction* allows the sitemap to send EMAIL messages on the basis of the data passing through the pipeline. The general format of the action is very similar to the Cocoon version. There are a set of parameters in both the component definition and the pipeline use that define how the *SendMailAction* works. The component is defined as follows

```
<map:actions>
   <map:action name="sendmail" src="resource://lib/acting/SendMailAction">
      <smtp-host>xx.xx.xx.xx</smtp-host>
      <smtp-user>xxxxxxxx</smtp-user>
      <smtp-password>*********</smtp-password>
   </map:action>
</map:actions>
```

where

- *smtp-host* — (optional) the IP address of the host to use which will deliver the EMAIL message.
- *smtp-user* — (optional) the user name of the mail account.
- *smtp-password* — (optional) the password of the mail account.

It is best to show an example of *SendMailAction* to see how it is used in the pipeline. For example a site might use it to send EMAILs using the following in the appropriate sitemap:

```
<map:match pattern="mail">
   <map:act type="sendmail">
      <map:parameter name="from" value="enquiries@paloose.org"/>
      <map:parameter name="to" value="{request-param:to-address}@paloose.org"/>
      <map:parameter name="subject" value="Paloose Site Mail"/>
      <map:parameter name="body" value="{request-param:body}"/>
      <map:aggregate element="root" label="aggr-content">
         <map:part src="cocoon:/headings.xml" element="headings" strip-root="true"/>
         <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
         <map:part src="cocoon:/newsArticles.xml" element="news-articles" strip-root="true"/>
         <map:part src="cocoon:/emailOk.xml" element="content" strip-root="true"/>
      </map:aggregate>
      <map:call resource="outputPage"/>
   </map:act>
   <map:aggregate element="root" label="aggr-content">
   <map:part src="cocoon:/headings.xml" element="headings" strip-root="true"/>
   <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
   <map:part src="cocoon:/newsArticles.xml" element="news-articles" strip-root="true"/>
   <map:part src="cocoon:/emailNotOk.xml" element="content" strip-root="true"/>
   </map:aggregate>
   <map:call resource="outputPage"/>
</map:match>
```

The generalised structure is

```
<map:act type="sendmail">
   <SendMailAction parameters>
   <success pipeline>
</map:act>
<pipeline run if action fails>
```

where

- *smtp-host* — (optional) the IP address of the host to use which will deliver the EMAIL message (as above).
- *smtp-user* — (optional) the user name of the mail account. (as above)
- *smtp-password* — (optional) the password of the mail account. (as above)

- *to* — (required) the destination of the message. This can be a list of comma separated email addresses.
- *from* — (required) the source of the message. This can be a list of comma separated email addresses.
- *cc* — (optional) the carbon copy destination of the message. This can be a list of comma separated email addresses.
- *bcc* — (optional) the blind carbon copy destination of the message. This can be a list of comma separated email addresses.
- *subject* — (optional) the subject text.
- *body* — (optional) the body text.

31 CookiesAction

The *CookiesAction* allows the sitemap access to the various cookie variables in the current transaction. The component is defined as follows

```
<map:actions>
   <map:action name="sendmail" src="resource://lib/acting/CookiesAction"/>
</map:actions>
```

It is best to show an example of *CookiesAction* to see how it is used in the pipeline. If there is a section of the pipeline that requires access to the cookie variables then it is enclosed within the action. For example, from an experimental Paloose CMS site:

```
<map:pipelines>

   <map:component-configurations>
      <map:global-variables>
         <themesDir>context://themes</themesDir>
         <defaultTheme>{ant:default-theme}</defaultTheme>
         <currentTheme>{cookies:theme}</currentTheme>
         <rootDir>context://</rootDir>
      </map:global-variables>
   </map:component-configurations>

   <map:pipeline>

      <map:match pattern="/(page).html/" type="regexp">
         <map:act type="cookies">
            <map:select type="regexp-selector">
               <map:parameter name="test-value" value="{global:currentTheme}"/>
               <map:when test="/.+/">
                  <map:generate src="context://themes/{global:currentTheme}/data/body.xml"
                            label="raw-xml"/>
               </map:when>
               <map:otherwise>
                  <map:generate src="context://themes/astral/data/body.xml" label="raw-xml"/>
               </map:otherwise>
            </map:select>
            <map:transform src="context://system/resources/transforms/buildBody.xsl"
                     label="buildBody-transform">
               <map:parameter name="page" value="{1}"/>
               <map:parameter name="themesDir" value="{global:themesDir}"/>
               <map:parameter name="theme" value="{global:currentTheme}"/>
               <map:parameter name="defaultTheme" value="{global:defaultTheme}"/>
               <map:parameter name="rootDir" value="{global:rootDir}"/>
               <map:parameter name="queryString" value="{global:query-string}"/>
            </map:transform>
            <map:transform src="context://system/resources/transforms/buildPanels.xsl"
                     label="buildPanels-transform"/>
            <map:transform src="context://site/resources/transforms/buildSite.xsl"
                     label="buildSite-transform"/>
            <map:transform src="context://themes/{global:currentTheme}/resources/transforms/buildTheme.xsl"
                     label="buildTheme-transform"/>
            <map:transform src="context://system/resources/transforms/normaliseProperties.xsl"
                     label="normalise-transform"/>
            <map:transform src="context://system/resources/transforms/page2xhtml.xsl"
                     label="page-transform"/>
            <map:transform type="moduleWrite"/>
            <map:serialize type="xhtml"/>
         </map:act>
      </map:match>
```

Note the moduleWrite transformer at the end of the action will updates the cookie variables, in this case the current theme. As the action is terminated (after the serialization the cookies are updated from the cookies module (which is why there is a ModuleWriteTransformer).

32 Authorisation Actions

Warning

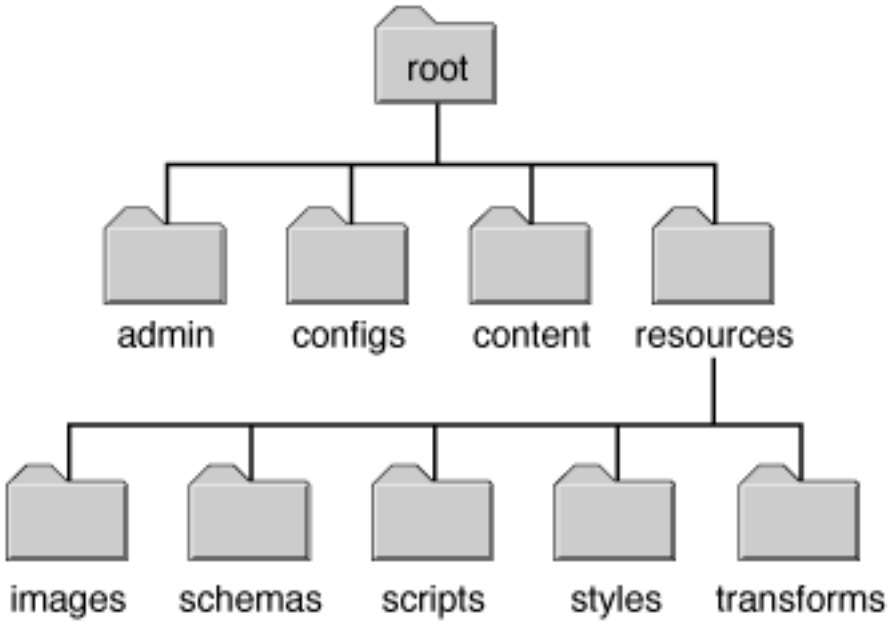
Note that the following is not the strongest method of preventing unauthorised access. The Paloose authorisation framework is only very loosely based on Cocoon so it is important to read the following carefully if you want to use this facility. Please read the Cocoon documentation on authentication as it is a useful background (it is much better than mine anyway).

The authorisation actions provide a mechanism to protect the sitemap pipeline and restrict use to only those requests that have been authorised. There are three main components, *AuthAction*, *AuthAction* and *AuthAction*. They are defined as components as follows:

```
<actions>
   <map:action name="auth-protect" src="resource://lib/acting/AuthAction"/>
   <map:action name="auth-login" src="resource://lib/acting/LoginAction"/>
   <map:action name="auth-logout" src="resource://lib/acting/LogoutAction"/>
</actions>
```

32 Authorisation Example

It is best to show an example of how to use these to explain their operation. Take a simple system of log-in to restrict certain users to administration areas. Consider a simple site with the following directory structure:



We would like to protect all the pages within the admin directory. Assuming a sub-sitemap in the admin directory with the above actions declared. The whole process falls into several stages:

- Authorisation Handler
- Protecting Individual Pages
- Authorising the User
- User Login
- User Logout

32 Authorisation Handler

The authorisation is controlled using a handler defined within the pipelines declaration. In the example we might have:

context://admin/sitemap.xmap

```
<map:pipelines>

   <map:component-configurations>
      <map:authentication-manager>
         <map:handlers>
            <map:handler name="adminHandler">
               <!-- Run this if the user needs login -->
               <map:redirect-to uri="cocoon:/login"/>
               <!-- The pipeline used to authenticate the user -->
               <map:authentication uri="cocoon:/authenticate-user.html" />
            </map:handler>
         </map:handlers>
      </map:authentication-manager>
   </map:component-configurations>
```

Like Cocoon it is possible to have several handlers to run different authorisation schemes for different documents. In the code above the handler *adminHandler* has an authentication mechanism invoked by calling (internally) the URI `cocoon:/authenticate-user.html`, which is matched to a pipeline within the current sitemap. If the user is authorised then the handler allows access to proceed. If not the use is redirected to the login process accessed using `cocoon:/login`, again within the current sitemap. ***Note that there is no application management in Paloose.***

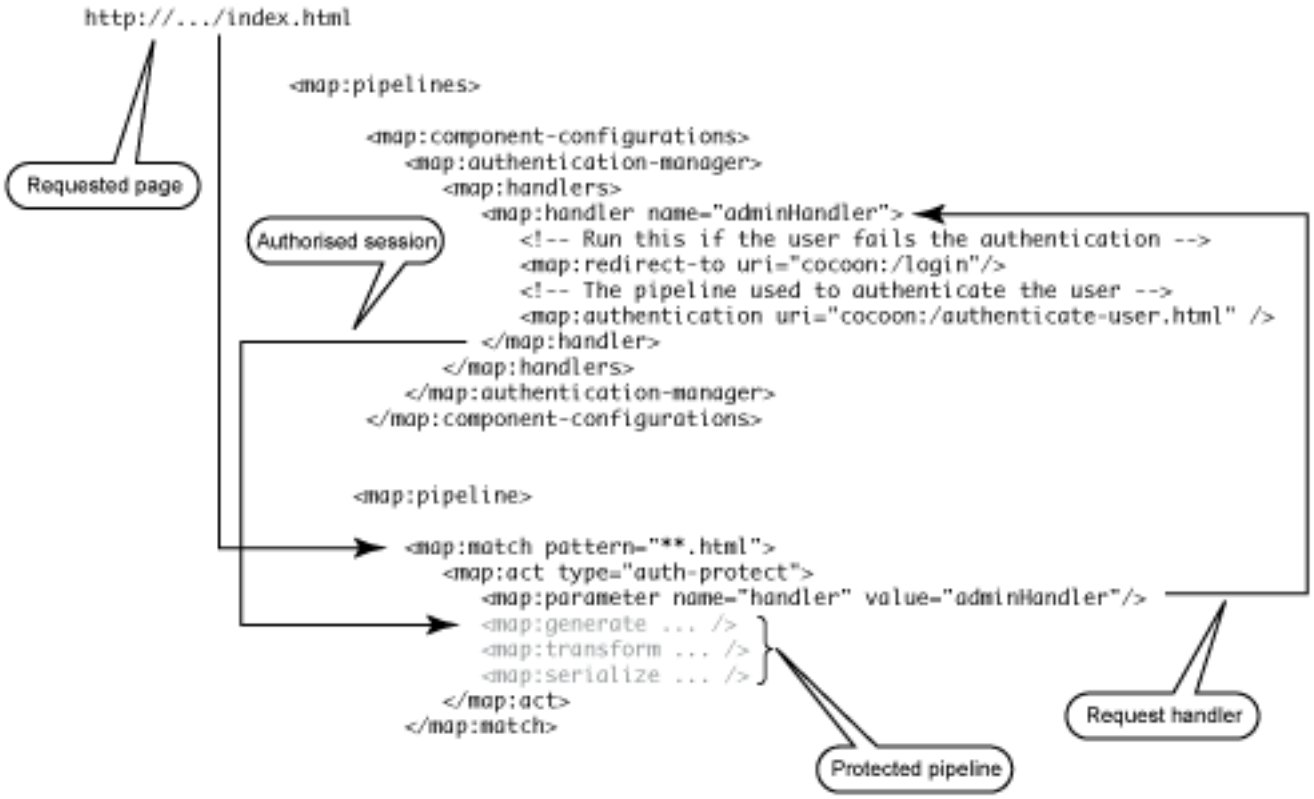
32 Protecting Individual Pages

In order to protect a request page we have to associate it with the *adminHandler* handler above. We do this by using the action *auth-protect* which was previously declared in the components section of the sitemap. The *auth-protect* action takes a single parameter defining the handler to use (*adminHandler*).

context://admin/sitemap.xmap

```
<map:match pattern="**.html">
   <map:act type="auth-protect">
      <map:parameter name="handler" value="adminHandler"/>
      <map:aggregate element="root" >
         <map:part ..../>
      </map:aggregate>
      <map:call resource="outputPage"/>
   </map:act>
</map:match>
```

In this case if the user is authorised (using the handler) to see all html pages matched in this sitemap then the pipeline will be processed as normal (aggregate, call etc). The following illustrates the relationship of the code above:



The next section deals with the actual authorising mechanism.

33 Authorising the User

First of all it is important to remember that this is only one means of doing this. The *adminHandler* defined in the component configuration section of the sitemap is:

context://admin/sitemap.xmap

```
<map:handler name="adminHandler">
   <!-- Run this if the user needs login -->
   <map:redirect-to uri="cocoon:/login"/>
   <!-- The pipeline used to authenticate the user -->
   <map:authentication uri="cocoon:/authenticate-user.html" />
</map:handler>
```

A password transformer to encrypt passwords is also declared (obviously the type of encryption is not important here and can be chosen by the user using their own transformer):

```
<map:transformers default="xslt">
   <map:transformer name="password" src="resource://lib/transforming/PasswordTransformer"/>
</map:transformers>
```

In order to authorise a user the authentication process with the URI "`cocoon:/authenticate-user.html`" is invoked. This is resolved within the current sitemap (the pseudo-protocol "`cocoon:/`") (although it does not strictly have to be there. In the example we have the following pipeline (note that it is internal only):

context://admin/sitemap.xmap

```
<map:pipeline internal-only="true">

   <map:match pattern="authenticate-user.html">
      <map:generate src="context://configs/adminUsers.xml" label="xml-content"/>
      <map:transform src="context://resources/transforms/admin-getLoginQuery.xsl">
         <map:parameter name="username" value="{request-param:username}"/>
         <map:parameter name="password" value="{request-param:password}"/>
      </map:transform>
      <!-- Encrypt the password -->
      <map:transform type="password" />
      <map:transform src="context://resources/transforms/admin-buildAuthenticateDOM.xsl" />
      <map:serialize type="xml"/>
      <!-- The output here is a standard \index{Cocoon}Cocoon authenticate structure and is returned to
         the authentication-manager -->
   </map:match>

</map:pipeline>
```

The pipeline takes the list of admin users, together with their associated data, and processes it to produce a single user after the `admin-getLoginQuery.xsl` transformer. The password entered by the user is then encrypted using the *PasswordTransformer* transformer previously declared. Finally an XML structure is built using the `admin-buildAuthenticateDOM.xsl` transformer.

The output of this is XML which is taken back by the *adminHandler* as confirmation that the use is authorised. This process is described in a little more detail below.

33 The Admin User File

The data on the users is stored in a simple XML file in the `context://configs` directory and takes the following example structure.

context://configs/adminUser.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<authentication>
   <users>
      <user>
```

```
         <username>hsfr</username>
         <password>9f7f32c605xxxxxxx225613d30b</password>
         <data>
            locally defined data about this user - address etc
         </data>
      </user>
   </users>
</authentication>
```

The file can obviously be put anywhere for stronger protection etc. The format can be changed but at the expense of rewriting all the transformers. The extra data structure holds user's details and an example of this is given on the documentation on Paloose forms. The data structure is fed into the `admin-getLoginQuery.xsl` transformer which is:

context://resources/transforms/admin-getLoginQuery.xsl

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

   <xsl:param name="password"/>
   <xsl:param name="username"/>

   <xsl:template match="//authentication">
      <xsl:element name="authentication" >
         <xsl:attribute name="username"><xsl:value-of select="$username" /></xsl:attribute>
         <xsl:attribute name="password"><xsl:value-of select="$password" /></xsl:attribute>
         <xsl:apply-templates/>
      </xsl:element>
   </xsl:template>

   <xsl:template match="@*|node()">
      <xsl:copy>
         <xsl:apply-templates select="@*|node()"/>
      </xsl:copy>
   </xsl:template>

</xsl:stylesheet>
```

This adds the user's username and password that was entered on the login form (accessed by the redirect). Thus we get:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<authentication username="hsfr" password="******">
   <users>
      <user>
         <username>hsfr</username>
         <password>9f7f32c605xxxxxxx225613d30b</password>
         <data>
            ...
         </data>
      </user>
   </users>
</authentication>
```

Next the password is encrypted using the *PasswordTransformer* transformer which outputs:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
   <authentication username="hsfr" password="9f7f32c605xxxxxxx225613d30b">
   <users>
      <user>
      <username>hsfr</username>
      <password>9f7f32c605xxxxxxx225613d30b</password>
      <data>
         ...
      </data>
      </user>
```

```
    </users>
</authentication>
```

The next stage is o make sure that the user has the correct password and build a suitable XML document to give back to the auth-action. This is done by the `admin-buildAuthenticateDOM.xsl` transformer

context://resources/transforms/admin-buildAuthenticateDOM.xsl

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

   <xsl:variable name="gPassword" select="//authentication/@password" />
   <xsl:variable name="gUsername" select="//authentication/@username" />

   <xsl:template match="authentication">
      <authentication>
         <xsl:apply-templates select="users"/>
      </authentication>
   </xsl:template>

   <xsl:template match="users">
      <xsl:apply-templates select="user"/>
   </xsl:template>

   <xsl:template match="user">
      <xsl:if test="normalize-space( username ) = $gUsername and
                    normalize-space( password ) = $gPassword">
         <ID><xsl:value-of select="username"/></ID>
         <data>
            <ID><xsl:value-of select="username"/></ID>
            <username><xsl:value-of select="username"/></username>
            <password><xsl:value-of select="password"/></password>
         </data>
      </xsl:if>
   </xsl:template>

</xsl:stylesheet>
```

This transformer outputs a valid structure for this user, which is similar to what was input (without the root attributes):

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<authentication>
   <ID>hsfr</ID>
   <data>
      <ID>hsfr</ID>
      <username>hsfr</username>
      <password>9f7f32c60........513d30b</password>
   </data>
</authentication>
```

or a blank structure:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
  <authentication/>
```

if the password does not check. This is used by the *AuthenticationManager* when performing its checks. Provided that the above structure is returned (the XML serialize stage at the end of the pipeline) you may provide any form of authorization mechanism. Anything that is between the `<data>...<data>` tags is considered user defined and is passed through transparently together with the *ID*, *username* and *password*.

The next section deals with the login mechanism.

34 User Login

The *adminHandler* redirects any failed authorisation to a suitable page, in this case the login.

context://admin/sitemap.xmap

```
<map:match pattern="login">
   <map:aggregate element="root" label="aggr-content">
      ...
      <map:part src="cocoon:/login.xml" element="content" strip-root="true"/>
   </map:aggregate>
   <map:call resource="outputPage"/>
</map:match>
```

The key piece is the login form:

context://admin/login.xml

```
<form:form xmlns:form="http://www.hsfr.org.uk/Schema/Form">
   <form:start url="checkLogin.html">Login</form:start>
   <form:field name="username" type="text">User name</form:field>
   <form:field name="password" type="password">Password</form:field>
</form:form>
```

In this example I have used the simple form that I use in my pages (using my own form namespace that I use for my personal pages — you can use your own form structure as long as it is translated to the appropriate HTML). It is translated to the following HTML:

```
<form method="post" action="checkLogin.html">
   <div class="normalPara">
      User name: <input name="username" type="text" />
      <br/>
      Password: <input name="password" type="password" />
      <br/>
   </div>
   <input type="submit" value="Login"/>
</form>
```

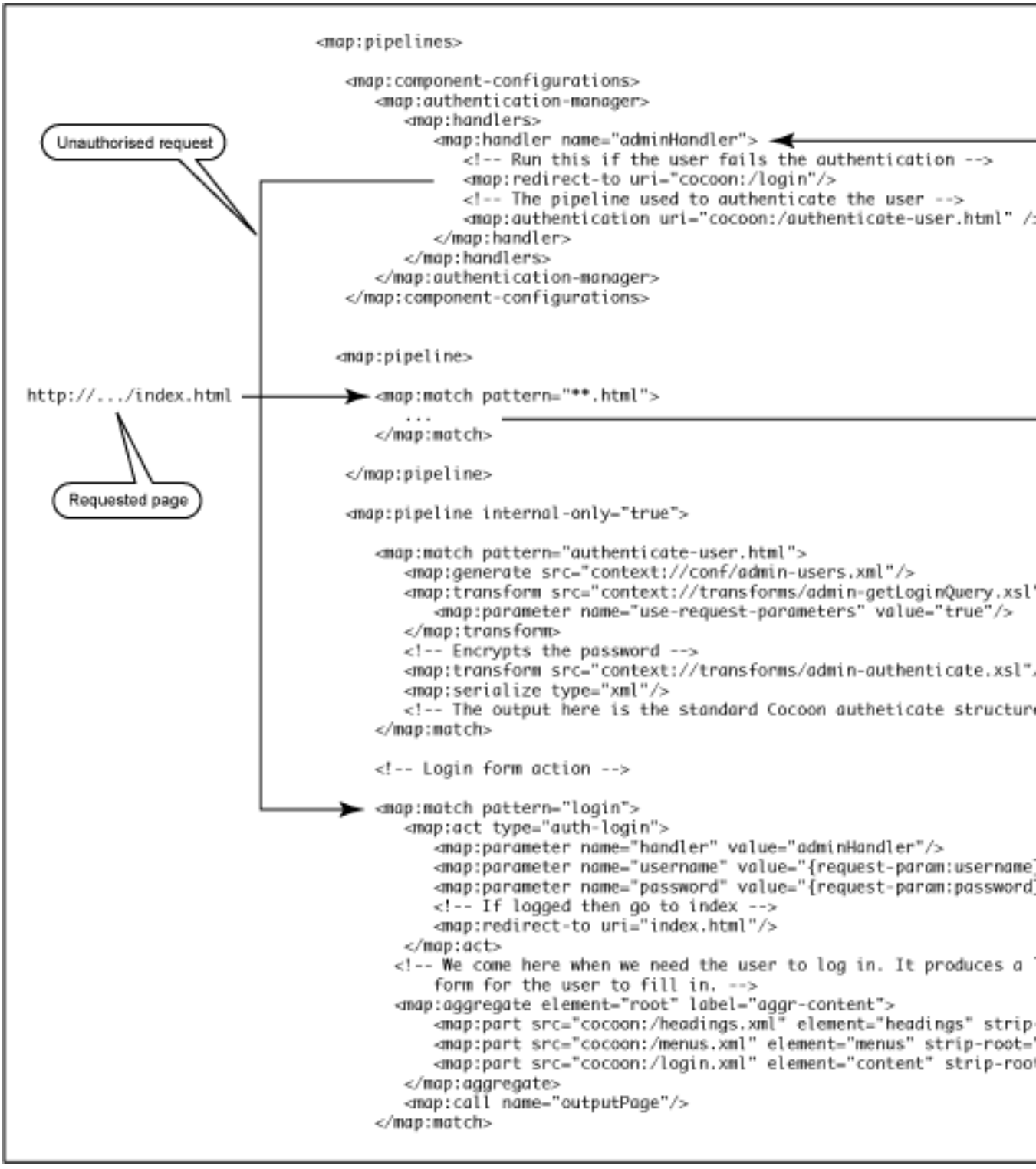
Note that this is not the same as the Paloose forms framework, although it could be used here. I have used the above for simplicity at this stage.

When the user press the "Login" button a request for the `checkLogin.html` page is made and is caught by the following matcher:

context://admin/sitemap.xmap

```
<map:match pattern="checkLogin.html">
   <map:act type="auth-login">
      <map:parameter name="handler" value="adminHandler"/>
      <map:parameter name="username" value="{request-param:username}"/>
      <map:parameter name="password" value="{request-param:password}"/>
      <map:redirect-to uri="cocoon:/adminIndex.html"/>   <!-- Run if authorisation works -->
   </map:act>
   <map:aggregate element="root" label="aggr-content">   <!-- Run if authorisation fails -->
      ...
      <map:part src="cocoon:/loginError.xml" element="content" strip-root="true"/>
   </map:aggregate>
   <map:call resource="outputPage"/>
</map:match>
```

The `auth-login` action deals with the login allowing for failed logins. In this case the latter would display the `loginError.xml`. The following shows the relationship of the various parts of the login code within the sitemap:



The next section deals with the logout mechanism.

35 User Logout

Logout is relatively simple, for example:

context://admin/sitemap.xmap

```
<map:match pattern="logout.html">
   <map:act type="auth-logout">
      <map:parameter name="handler" value="adminHandler"/>
      <map:aggregate element="root" label="aggr-content">
         <map:part src="cocoon:/headings.xml" element="headings" strip-root="true"/>
         <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
         <map:part src="cocoon:/newsArticles.xml" element="news-articles" strip-root="true"/>
         <map:part src="cocoon:/login.xml" element="content" strip-root="true"/>
      </map:aggregate>
      <map:call resource="outputPage"/>
   </map:act>
   <!-- If problem -->
   <map:redirect-to uri="logoutProblem.html"/>
</map:match>
```

A successful logout goes back to the login screen. An unsuccessful logout brings up a suitable error page.

36 Flows (and Forms)

It is useful to provide some background to some of the decisions that I made for flows and forms. This is where Paloose diverges from Cocoon most. The latter is based on Java and Javascript which was not available to me (conceptually, not practically, as Paloose is based solely on PHP5). Thus I had to base what I did on a pure PHP approach while including the concepts of Cocoon flows and forms. I made several false starts which arose from several design problems:

- How to store the information at each point of the flow?
- Which form template to use (CForms, XForms etc.)?
- How to allow forward and back (continuations) in the forms?

All of these problems seemed intractable at first with decisions made about one influencing, detrimentally, a decision made elsewhere. Mirroring the Cocoon forms template, CForms was initially desirable for commonality with Cocoon. But it soon became apparent that this was going to make the whole approach far too complicated for what I wanted. Much code was wasted in exploring this but I believe it was the right decision. In the end I based the Paloose forms (PForms) on the JXForms that older version of Cocoon used. (I had produced some sites with this in the past so I was reasonably familiar with it). PForms is not radically different from JXForms but it does not slavishly follow the latter. I believe that PForms is suitable for the restrictions I had set on Paloose and, as I have said elsewhere, if you have a site that requires all the facilities of CForms then you should probably be using Cocoon anyway.

Once I had settled on PForms there was the problem of how to implement the flow script, which in Cocoon uses Javascript. The Cocoon approach is to take a "snapshot" of the Javascript engine (a simplistic way of describing it) in order to maintain continuity between client requests. I was unable to find a sensible way of doing this with PHP5 (which does not mean to say that one does not exist). So I had to find a means of providing the user with a simple method of continuations based solely on a PHP5 approach. I believe I have achieved this while keeping to the spirit of Cocoon.

All these solutions, however successful, have made me diverge from the strict Cocoon approach so please read the documentation very carefully.

Flows and Forms are best explained with an example.

36 The Flow Script

Warning

This part of Paloose is completely different from Cocoon — so please study carefully.

The aim is to provide a means of navigating a series of linked forms to provide a coherent whole. It is done by producing a PHP script which allows this movement between the pages. It allows us to check for data and give the user an opportunity to change their mind before finally submitting the data. The overall structure of this flow class is:

context://resources/scripts/AddUserTest.php

```
<?php

require_once( PHP_DIR . "/flows/Continuations.php" );

class AddUserTest extends Continuations {

   /** Logger instance for this class */
   private $gLogger;

   private $gAddFormTestModel = array (
      "username" => "",
      "password" => "",
      "passwordCheck" => "",
      "fullname" => "",
   );

   function __construct()
   {
       $this->gLogger =& LoggerManager::getLogger( __CLASS__ );
      parent::__construct( $this->gAddFormTestModel );
   }

   ...

}
?>
```

The class is based on the Paloose class *Continuations* which provides the basis of the flow control. The data model declared as the array, *$ gAddFormTestModel* stores the data that will be collected from the form (similar to the Cocoon data model approach). The construct method sets up a logger (optional) and registers the data model with the *Continuations* class.

Warning

Do not use any id starting with "__", these are reserved for Paloose. Also do not have "." within the field names as they get changed when the form is sent via POST.

For each operation that the flow does we need to add a class which is referenced within the sitemap pipeline. In this case it is the *add* method, whose basic structure is:

context://resources/scripts/AddUserTest.php

```
public function add()
{
     global $gModules;

   $requestParameterModule = $gModules[ 'request-param' ];

   $finished = false;
   while ( !$finished ) {
```

```
        $errors = false;
         switch ( $this->gContinuation ) {

            case 0 :
               $this->sendPage(
                  "addUserTest-1.html",        // The page to send to the client
                  $this->gAddFormTestModel,    // The current data model to send
                  ++$this->gContinuation,      // The continuation link for the next stage
                  $this->gViolations );        // Array of errors (keyed by data model keys)
               $finished = true;
               break;

            case 1 :
              // Send next page
               break;

            case 2 :
              // Send next page
               break;

            case 3 :
              // Send next page
               break;

           ...
        }
     }
}
```

The method *sendPage($ inPage, &$ inFormModel, $ inContinuation, $ inViolations)* is a method in the *Continuations* class which is used to send a page to the client. The parameters are

- *$ inPage* — the page to send.
- *$ inFormModel* — the data model (passed by reference).
- *$ inContinuation* — the continuation id, used by the switch statement.
- *$ inViolations* — an array of entry errors (keyed by the data model keys (see below).

In order to use this script it must be declared in the site:

context://documentation/sitemap.xmap

```
<map:flow language="php">
   <map:script src="context://resources/scripts/AddUserTest.php"/>
</map:flow>
```

We will also need the *PXTemplateGenerator*:

context://documentation/sitemap.xmap

```
<map:components>
   <map:generators default="file">
      <map:generator name="px" src="resource://lib/generation/PXTemplateGenerator"/>
   </map:generators>
   ...
</map:components>
```

The pipeline entry consists of two parts. The first matches to the start of the whole process:

context://documentation/sitemap.xmap

```
<map:pipeline>
   <map:match pattern="addUserTest.html">
      <map:call function="AddUserTest::add"/>
   </map:match>
```

It has a single pipeline component that calls the function *AddUserTest::add*, which is the method *add* in the class *AddUserTest* which we declared above. When the match is made it starts of the flow process and returns the page `addUserTest-1.html` since the continuation id is "0".

The second pipeline entry is to match the continuations which is done by the following:

context://documentation/sitemap.xmap

```
    <map:match pattern="addUserTest.kont">
       <map:call function="AddUserTest::add"/>
    </map:match>
 </map:pipeline>
```

Sending the page `addUserTest-1.html` requires an internal pipeline to build the page:

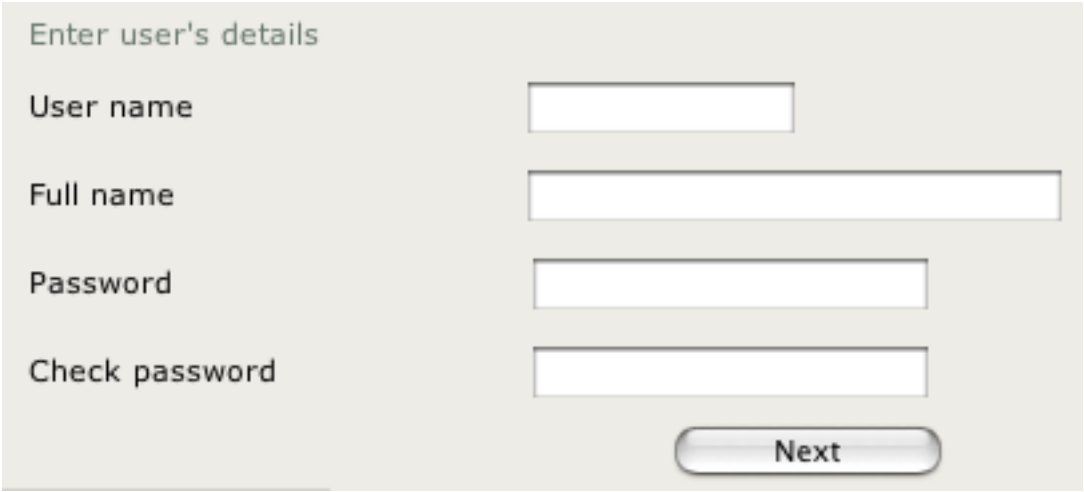
context://documentation/sitemap.xmap

```
 <map:pipeline internal-only="true">
    <map:match pattern="addUserTest-*.html">
       <map:aggregate element="root">
          ...
          <map:part src="cocoon:/addUserTest-{1}.px" element="content" strip-root="true"/>
       </map:aggregate>
       <map:transform src="resource://resources/transforms/pforms-violations.xsl"
          label="pforms-violations">
          <map:parameter name="formViolations" value="{session:__violations}"/>
       </map:transform>
       <map:transform src="resource://resources/transforms/pforms-default.xsl"/>
       <map:transform src="resource://resources/transforms/pforms2html.xsl"/>
       <map:call resource="outputPage"/>
    </map:match>

    <map:match pattern="**.px">
       <map:generate type="px" src="context://documentation/{1}.xml"/>
       <map:serialize/>
    </map:match>
 </map:pipeline>
```

There are several things to note here:

1. The page XML is generated using a *PXTemplateGenerator* to expand the Paloose variables which are used in back/forward progress of the flow.
2. After the page is built (in the aggregate) the Pforms are processed.
3. Even though there is no login there is still a session running.

Running this will produce a form similar to:



The next page describes the next stage and checking for errors.

36 Next Sequence and Checking for Errors

When the "next" button is pressed it sends the form data with the request:

```
http://<hostname>documentation/addUserTest.kont?__session=1174930335
```

The request is made up of the forms *form* tag's `@flow` attribute (`addUserTest`) and the "`kont`" extension. The session variable is added by the Continuations software. This request matches the pipeline:

context://documentation/sitemap.xmap

```
    <map:match pattern="addUserTest.kont">
       <map:call function="AddUserTest::add"/>
    </map:match>
</map:pipeline>
```

which runs the *add* method we saw previously. However his time it is directed to the 2 case of the switch statement which now has to output the next form in the sequence. If data is to be checked, this is where it is done. How you do this is up to you but the following seems to work quite well for this case. First declare a violations array:

context://resources/scripts/AddUserTest.php

```
<?php

require_once( PHP_DIR . "/flows/Continuations.php" );

class AddUserTest extends Continuations {

   /** Logger instance for this class */
   private $gLogger;

   private $gAddFormTestModel = array (
      "username" => "",
      "password" => "",
      "passwordCheck" => "",
      "fullname" => "",
   );

   private $dataRequired = array (
      "username" => "Must enter a username",
      "password" => "Must enter a password",
      "passwordCheck" => "Must enter a password check",
   );

   function __construct()
   {
       $this->gLogger =& LoggerManager::getLogger( __CLASS__ );
      parent::__construct( $this->gAddFormTestModel );
   }
```

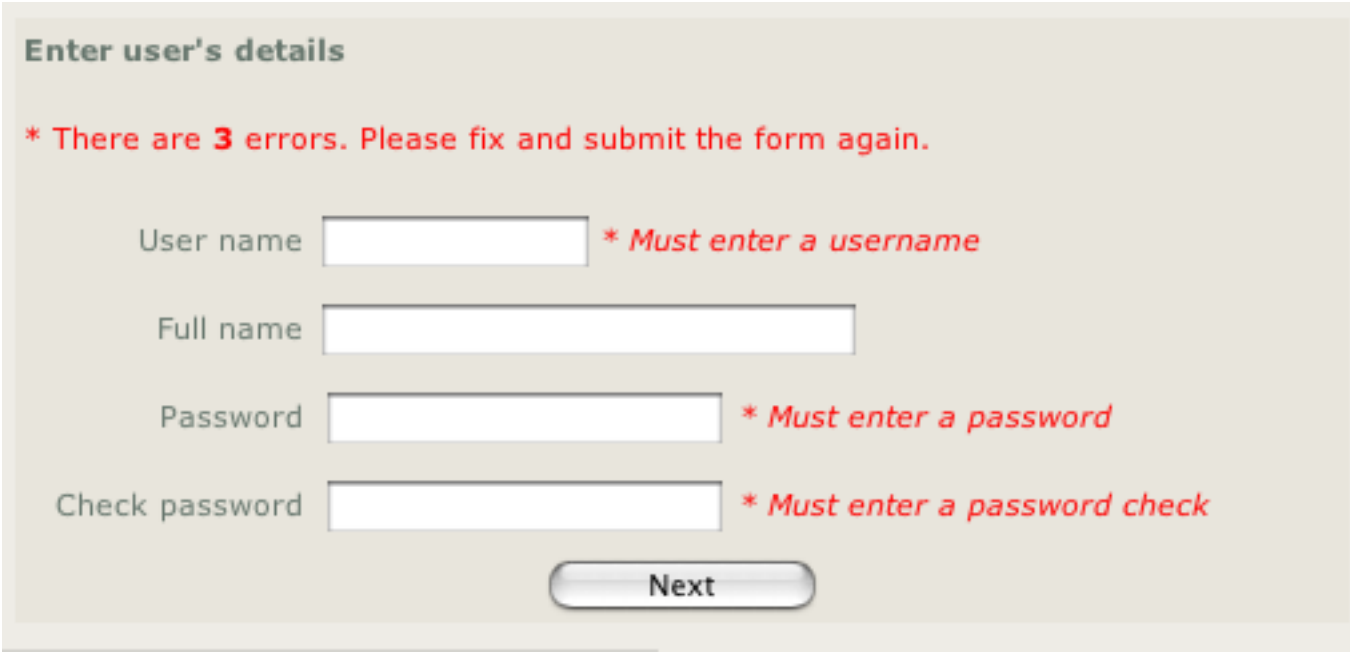
Next add the actual error check to the case 2 of the continuations selector:

context://resources/scripts/AddUserTest.php

```
    public function add()
    {
         global $gModules;

       $requestParameterModule = $gModules[ 'request-param' ];

       $finished = false;
       while ( !$finished ) {
          $errors = false;
          switch ( $this->gContinuation ) {

          case 0 :
             $this->sendPage(
```

```
                "addUserTest-1.html",        // The page to send to the client
                $this->gAddFormTestModel,    // The current data model to send
                ++$this->gContinuation,      // The continuation link for the next stage
                $this->gViolations );        // Array of errors (keyed by data model keys)
             $finished = true;
             break;

          case 1 :
             foreach ( $this->dataRequired as $key => $msg ) {
                if ( !isset( $this->gAddFormTestModel[ $key ] ) or
                        strlen( $this->gAddFormTestModel[ $key ] ) == 0 ) {
                  $this->gViolations[ $key ] = $msg;
                  $errors = true;
                }
             }

             // Now check for remaining errors. Adjust the order of the checks to suit local conditions.
             // The last check is what is displayed.
             if ( $errors === false ) {
                if ( strlen( $this->gAddFormTestModel[ 'password' ] ) < 8 ) {
                   $this->gViolations[ 'password' ] = "Password must be more than 8 characters long";
                   $errors = true;
                }
                if ( $this->gAddFormTestModel[ 'password' ] !=
                                      $this->gAddFormTestModel[ 'passwordCheck' ] ) {
                   $this->gViolations[ 'password' ] = "Password does not check";
                   $errors = true;
                }
             }
             if ( $errors ) {
                // Set back to previous stage and go round again (sends first page again with violations
                $this->gContinuation--;
                $finished = false;
             } else {
                // No errors so send next page
                $this->sendPage(
                   "addUserTest-2.html",
                   $this->gAddFormTestModel,
                   ++$this->gContinuation,
                   $this->gViolations );
                $finished = true;
              }
             break;

          ...
          }
       }
    }
```

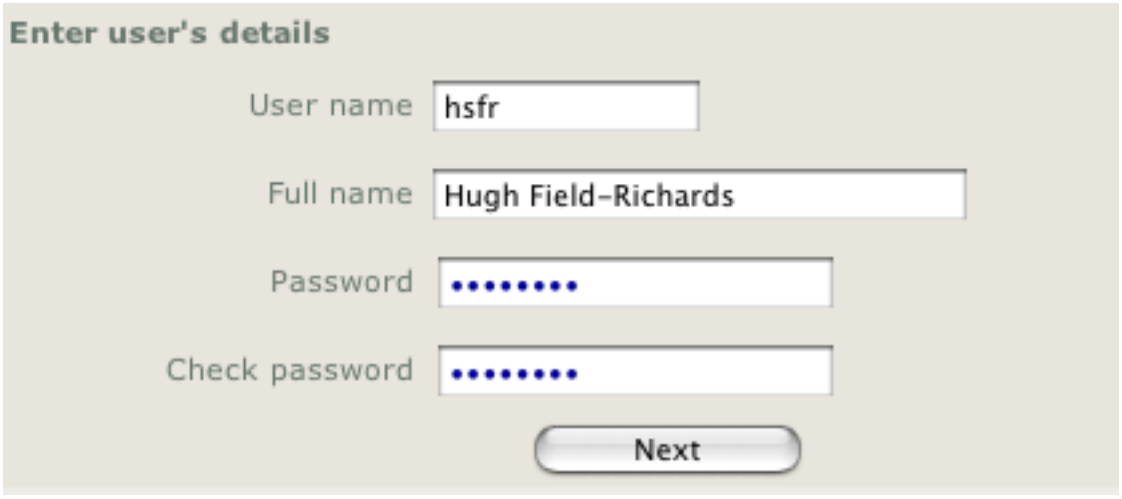
If we do not fill any fields in and submit the form then the first page is sent again with the violations marked:



The next page describes the confirmation of data.

36 Confirming the data

Let us assume that we have entered correct data, for example:



This is then sent to the second stage of the flow script which sends the following form:

context://dosumentation/addUserTest-2.xml

```
<pf:form id="addUserTestForm"
         flow="addUserTest"
         continuation="{flow:continuation.id}"
         session="{flow:__flowId}" >
   <pf:label>You input the following information:</pf:label>

   <pf:output ref="username">
       <pf:label>Username:</pf:label>
      <pf:value>{flow:username}</pf:value>
   </pf:output>

   <pf:output ref="fullname">
      <pf:label>Full name:</pf:label>
      <pf:value>{flow:fullname}</pf:value>
   </pf:output>

   <pf:submit class="button" id="prev">
      <pf:label>Back</pf:label>
      <pf:hint>Go to previous page</pf:hint>
   </pf:submit>

   <pf:submit class="button" id="next">
      <pf:label>Confirm</pf:label>
      <pf:hint>Confirm new user</pf:hint>
   </pf:submit>
</pf:form>
```

which will display:

Adding the data to a file can be carried out in a further stage which uses the SourceWritingTransformer. In those immortal words, "I will leave this as a simple exercise for the reader", subtext for "I haven't time to document it yet".

I have set up an example above so that you can see how it works (less the SourceWriting stage. The example below has the following extra code:

context://dosumentation/addUserTest-3.xml

```
<pf:form id="addUserTestForm"
            flow="addUserTest"
            continuation="{flow:continuation.id}"
            session="{flow:__flowId}" >
   <pf:label>You input the following information:</pf:label>

   <pf:output ref="username">
      <pf:label>Username:</pf:label>
      <pf:value>{flow:username}</pf:value>
   </pf:output>

   <pf:output ref="fullname">
      <pf:label>Full name:</pf:label>
      <pf:value>{flow:fullname}</pf:value>
   </pf:output>

   <pf:submit class="button" id="next">
      <pf:label>Finished</pf:label>
      <pf:hint>Back to documentation home page</pf:hint>
   </pf:submit>
</pf:form>
```

context://resources/scripts/AddUserTest.php

```
   public function add()
   {
      global $gModules;

      $requestParameterModule = $gModules[ 'request-param' ];

      $finished = false;
      while ( !$finished ) {
         $errors = false;
         switch ( $this->gContinuation ) {

         case 0 :
            $this->sendPage(
               "addUserTest-1.html",        // The page to send to the client
               $this->gAddFormTestModel,    // The current data model to send
               ++$this->gContinuation,      // The continuation link for the next stage
               $this->gViolations );        // Array of errors (keyed by data model keys)
            $finished = true;
            break;

         case 1 :
            foreach ( $this->dataRequired as $key => $msg ) {
               if ( !isset( $this->gAddFormTestModel[ $key ] ) or
                       strlen( $this->gAddFormTestModel[ $key ] ) == 0 ) {
                 $this->gViolations[ $key ] = $msg;
                 $errors = true;
               }
            }

            // Now check for remaining errors. Adjust the order of the checks to suit local conditions.
            // The last check is what is displayed.
            if ( $errors === false ) {
               if ( strlen( $this->gAddFormTestModel[ 'password' ] ) < 8 ) {
                  $this->gViolations[ 'password' ] = "Password must be more than 8 characters long";
                  $errors = true;
               }
               if ( $this->gAddFormTestModel[ 'password' ] !=
                           $this->gAddFormTestModel[ 'passwordCheck' ] ) {
                  $this->gViolations[ 'password' ] = "Password does not check";
                  $errors = true;
```

```
            }
          }
          if ( $errors ) {
             // Set back to previous stage and go round again (sends first page again with violations
             $this->gContinuation--;
             $finished = false;
          } else {
             // No errors so send next page
             $this->sendPage(
                "addUserTest-2.html",
                $this->gAddFormTestModel,
                ++$this->gContinuation,
                $this->gViolations );
             $finished = true;
           }
          break;

        case 2 :
          // No errors to record from optional data so send next page
          $this->sendPage(
             "addUserTest-3.html",
             $this->gAddFormTestModel,
             ++$this->gContinuation,
             $this->gViolations );
          $finished = true;
          break;

        case 3 :
          // Data confirmed return to documentation index
          $this->sendPage(
             "documentation.html",
             $this->gAddFormTestModel,
             ++$this->gContinuation,
             $this->gViolations );
          $finished = true;
          break;

       }
    }
}
```

37 Paloose Forms

The Paloose Forms (PForms) framework is loosely based on JXForms (now deprecated) which in turn was based on XForms. There are sufficient differences with PForms to make the latter to be of only loose help. It is worth reading the FAQ entry on flows for some background information on why Paloose Forms do not use CForms.

The use of PForms is described elsewhere, this page just gives the PForm elements.

37 Basic Structure

A Paloose form is enclosed by the `<form>` in the namespace ‘‘`http://www.paloose.org/schemas/Forms/1.0`‘‘:

```
<pf:form id="addUserForm" flow="addUser" continuation="{flow:continuation.id}" session="{flow:__flowId}">
   <pf:label>Text associated with the form</pf:label>
   ...
</pf:form>
```

where:

- *id* — the identity of this form (used mainly in the CSS,
- *flow* — the flow being used,
- *continuation* — the continuation identity used to navigate the flow,
- *session* — the current flow session id.

For more explanation of these see the PForms example. Within the form structure there are a set of items that reflect the various components within a form.

37 Simple Input Field (Input).

Input fields contain a single line entry field (cf HTML `<input>` input field). It has the following example structure:

```
<pf:input ref="username" class="usernameField">
   <pf:label>...</pf:label>
   <pf:value>...</pf:value>
   <pf:hint>...</pf:hint>
   <pf:violations/>
</pf:input>
```

where the attributes are:

- *ref* — how the flowscript references this field,
- *class* — the id for the CSS style.

37 Simple Password Field (secret).

Secret fields are identical to input fields except that text entered only displays as “*”. They have identical structure to input fields:

```
<pf:secret ref="password" class="passwordField">
   <pf:label>...</pf:label>
   <pf:value>...</pf:value>
   <pf:hint>...</pf:hint>
   <pf:violations/>
</pf:secret>
```

where the attributes are:

- *ref* — how the flowscript references this field,
- *class* — the id for the CSS style.

37 Hidden Field (hidden).

Hidden fields allow the transmission of information that is hidden from the user. The only enclosed data is the initial value:

```
<pf:hidden ref="password">
   <pf:value>...</pf:value>
</pf:hidden>
```

where the attributes are:

- *ref* — how the flowscript references this field,

37 Output Field (output).

Output fields allow display of data to the user in a read-only field for giving the user information. For example the following would display the value of the flow variable *username*:

```
<pf:output ref="username">
    <pf:label>Username:</pf:label>
    <pf:value>{flow:username}</pf:value>
</pf:output>
```

where the attributes are:

- *ref* — how the flowscript references this field,

37 Form button (submit).

Submit entries define how the form is treated, for example would produce an HTML form button:

```
<pf:submit class="button" id="next">
   <pf:label>Next</pf:label>
   <pf:hint>Go to optional details entry</pf:hint>
</pf:submit>
```

where the attributes are:

- *ref* — how the flowscript references this field,
- *class* — the id for the CSS style.

37 Multiple Select Fields.

When it is required to select several of a set of choices the multiple select is used. This is the equivalent of the checkbox HTML form field. Each choice is entered as a set of fields

```
<pf:select appearance="full|compact" ref="roles">
   <pf:value>...</pf:value>
   <pf:choices>
      <pf:item checked="false|true">
         <pf:label>...</pf:label>
```

```
            <pf:value>...</pf:value>
            <pf:hint>...</pf:hint>
         </pf:item>
         <pf:item>
            ...
         </pf:item>
      </pf:choices>
   </pf:select>
```

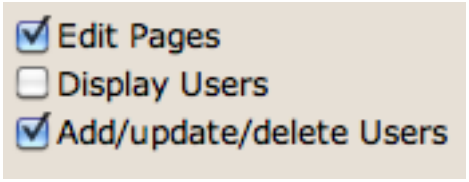
where the attributes are:

- *ref* — how the flowscript references this field,
- *appearance* — the style of the multiple selection. *full* denotes a set of checkboxes, and *compact* is a multiple selection list. If *appearance* is omitted then *compact* is assumed.
- *checked* — whether the checkbox is checked at page load time. If *checked* is omitted then *false* is assumed.

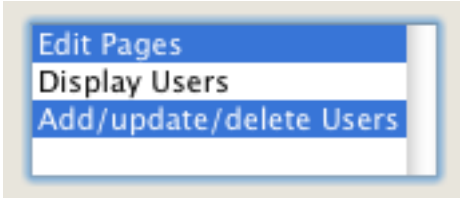
For example:

```
<pf:select appearance="full" ref="roles">
   <pf:value>editPage,manageUsers</pf:value>
   <pf:choices>
      <pf:item>
         <pf:label>Edit Pages</pf:label>
         <pf:value>editPage</pf:value>
         <pf:hint>Can edit pages within the Web site</pf:hint>
      </pf:item>
      <pf:item>
         <pf:label>Display Users</pf:label>
         <pf:value>displayUsers</pf:value>
         <pf:hint>Can display users</pf:hint>
      </pf:item>
      <pf:item>
         <pf:label>Add/update/delete Users</pf:label>
         <pf:value>manageUsers</pf:value>
         <pf:hint>Can manage users: change password, delete/add users, change roles</pf:hint>
      </pf:item>
   </pf:choices>
</pf:select>
```

would display as:



while a compact version would display as:



37 Single Select Fields.

When it is required to select several of a set of choices the multiple select is used. This is the equivalent of the checkbox HTML form field. Each choice is entered as a set of fields

```
<pf:select1 appearance="full|compact" ref="...">
   <pf:value>...</pf:value>
   <pf:choices>
      <pf:item>
         <pf:label>...</pf:label>
         <pf:value>...</pf:value>
         <pf:hint>...</pf:hint>
      </pf:item>
      <pf:item>
         ...
      </pf:item>
   </pf:choices>
</pf:select>
```

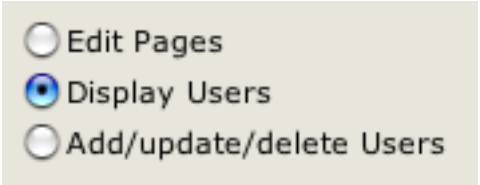
where the attributes are:

- *ref* — how the flowscript references this field,
- *appearance* — the style of the mutiple selection. *full* denotes a set of checkboxes, and *compact* is a multiple selection list.

For example:

```
<pf:select appearance="full" ref="roles">
   <pf:value>displayUsers</pf:value>
   <pf:choices>
      <pf:item>
         <pf:label>Edit Pages</pf:label>
         <pf:value>editPage</pf:value>
         <pf:hint>Can edit pages within the Web site</pf:hint>
      </pf:item>
      <pf:item>
         <pf:label>Display Users</pf:label>
         <pf:value>displayUsers</pf:value>
         <pf:hint>Can display users</pf:hint>
      </pf:item>
      <pf:item>
         <pf:label>Add/update/delete Users</pf:label>
         <pf:value>manageUsers</pf:value>
         <pf:hint>Can manage users: change password, delete/add users, change roles</pf:hint>
      </pf:item>
   </pf:choices>
</pf:select>
```

would display as:



37 Text Area

text area fields allow multi-line

```
<pf:textarea ref="...">
    <pf:label>...:</pf:label>
    <pf:hint>...</pf:hint>
    <pf:value>...</pf:value>
</pf:textarea>
```

where the attributes are:

- *ref* — how the flowscript references this field,

For example:

```
<pf:textarea ref="comments" class="commentField">
   <pf:value>{flow:comments}</pf:value>
   <pf:hint>Any special comments outside of the above</pf:hint>
</pf:textarea>
```

37 Label Field

Label fields contain text that is associated with the field. For example:

```
<pf:input ref="username" class="usernameField">
   <pf:label>User name</pf:label>
   ...
</pf:input>
```

which would typically output

37 Value Field

Value fields contain predefined data that is loaded into the field when the page is first displayed. For example:

```
<pf:input ref="username" class="usernameField">
   <pf:value>Please enter username</pf:value>
   ...
</pf:input>
```

would put the text "Please enter username" into the field when the page is loaded. If the PXTemplateGen is used then it is possible to use sitemap variables, for example:

```
<pf:input ref="username" class="usernameField">
   <pf:value>{flow:username}</pf:value>
   ...
</pf:input>
```

37 Hint Field

Hint field contains text that is output when the cursor is hovered above the field. For example:

```
<pf:input ref="username" class="usernameField">
   <pf:hint>The user's login username</pf:hint>
   ...
</pf:input>
```

37 Violations Field

Violations fields are empty place holders used when continuations require to report. Any tags within here will be ignored. For example:

```
<pf:input ref="username" class="usernameField">
   <pf:violations/>
   ...
</pf:input>
```

38 Optimising Paloose Performance

38 Some Background

First of all it is useful to look at the performance of a completely "vanilla" system. The test server I used was a Linux box with the following specification:

- Single processor AMD Athlon 64 FX-57 Processor (2.8GHz + 1M cache)
- 2G Memory
- NVIDIA GeForce FX to GeForce 8800 Graphics Processor
- 80G SATA drive
- Mandriva Powerpack 2008 Linux running Version 2.6.24.4 kernel.

Not the most highly specified machine but it served — its history is simple: my son had a bag of bits that he did not know what to do with, so I added a case and PSU and gained a new machine.

First of all to produce a bench mark I ran some tests (using siege) running on my local network. The server is connected to this network wirelessly via a Netgear DG834G router, while the siege machine, which was a MacPro G5 running Mac OS-X Leopard. Nothing very outstanding here but more than sufficient to get some representative results.

The first test was to run the siege test on Apache (using the same HTML version of the XML Paloose test file) and also the same with Tomcat running 2.1.10 Cocoon (caching on). These were run against an identical set of XML files and transforms in Paloose. The sitemap was:

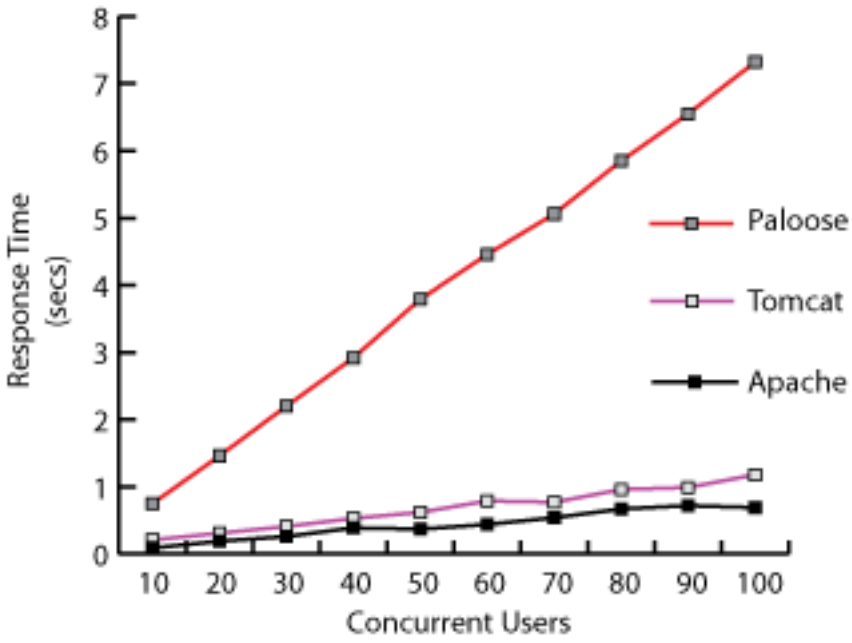
Test sitemap

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

   <map:components>

      <map:generators default="file">
         <map:generator name="file" src="resource://lib/generation/FileGenerator"/>
      </map:generators>

      <map:transformers default="xslt">
         <map:transformer name="xslt" src="resource://lib/transforming/TRAXTransformer">
            <map:use-request-parameters>true</map:use-request-parameters>
         </map:transformer>
         <map:transformer name="i18n" src="resource://lib/transforming/I18nTransformer">
            <map:catalogues default="index">
               <map:catalogue id="index" name="index" location="context://content/translations"/>
             </map:catalogues>
            <map:untranslated-text>untranslated text</map:untranslated-text>
         </map:transformer>
      </map:transformers>

     <map:serializers default="xml">
         <map:serializer name="html" mime-type="text/html"
               src="resource://lib/serialization/HTMLSerializer">
            <doctype-public>-//W3C//DTD HTML 4.01 Transitional//EN</doctype-public>
            <doctype-system>http://www.w3.org/TR/html4/loose.dtd</doctype-system>
            <encoding>iso-8859-1</encoding>
         </map:serializer>
         <map:serializer name="xhtml" mime-type="text/html"
               src="resource://lib/serialization/XHTMLSerializer">
            <doctype-public>-//W3C//DTD XHTML 1.0 Transitional//EN</doctype-public>
            <doctype-system>http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd</doctype-system>
            <encoding>iso-8859-1</encoding>
         </map:serializer>
         <map:serializer name="text" mime-type="text/plain"
               src="resource://lib/serialization/TextSerializer"/>
         <map:serializer name="xml" mime-type="text/xml"
               src="resource://lib/serialization/XMLSerializer"/>
      </map:serializers>

      <map:matchers default="wildcard">
         <map:matcher name="wildcard" src="resource://lib/matching/WildcardURIMatcher"/>
```

```
        <map:matcher name="regexp" src="resource://lib/matching/RegexpURIMatcher"/>
      </map:matchers>

      <map:readers default="resource">
         <map:reader name="resource" src="resource://lib/reading/ResourceReader"/>
      </map:readers>

      <map:selectors default="browser">
         <map:selector name="browser" src="resource://lib/selection/BrowserSelector">
            <browser name="explorer" useragent="MSIE"/>
            ...
            <browser name="netscape" useragent="Mozilla"/>
            <browser name="safari" useragent="Safari"/>
         </map:selector>
      </map:selectors>

   </map:components>

   <map:pipelines>

      <map:pipeline>

         <map:match pattern="**.html">
            <map:generate src="context://content/{1}.xml" label="xml-content"/>
            <map:transform src="context://resources/transforms/page2html.xsl" label="page-transform">
               <map:parameter name="page" value="{1}"/>
            </map:transform>
            <map:transform src="context://resources/transforms/stripNamespaces.xsl"/>
            <map:serialize type="html"/>
         </map:match>

      </map:pipeline>

      <map:handle-errors>
         <map:generate src="context://content/error.xml"/>
         <map:transform src="context://resources/transforms/error2html.xsl"/>
         <map:serialize type="html"/>
      </map:handle-errors>

   </map:pipelines>

</map:sitemap>
```

While not as simple as it could be it is reasonably representative of a single sitemap site. Note that there are no subsitemaps. The XML content was also suitably simple at about 200 lines of trivial XML. The XSL transforms were also suitably pared down from those that serve this Paloose site. In fact we are not looking for absolute speeds here, merely comparisons betwen the benchmark of Apache and Tomcat/Cocoon against Paloose.

As can be seen from the graph below Paloose does not score highly against Apache or a cached Tomcat/Cocoon. This is quite expected and a result of many factors: PHP, parsing the sitemaps each time, using DOM instead of SAX in the sitemap pipeline. The best we can say is that there is room for improvement and that all the sites that I have run have not needed the raw speed that Apache or Tomcat could deliver. It is also another reason why Apache should be used to serve the files hat Paloose does not deal with (CSS, graphics etc).



The next page looks at how we can increase the performance

39 Optimising Paloose Performance

39 Caching the Sitemap

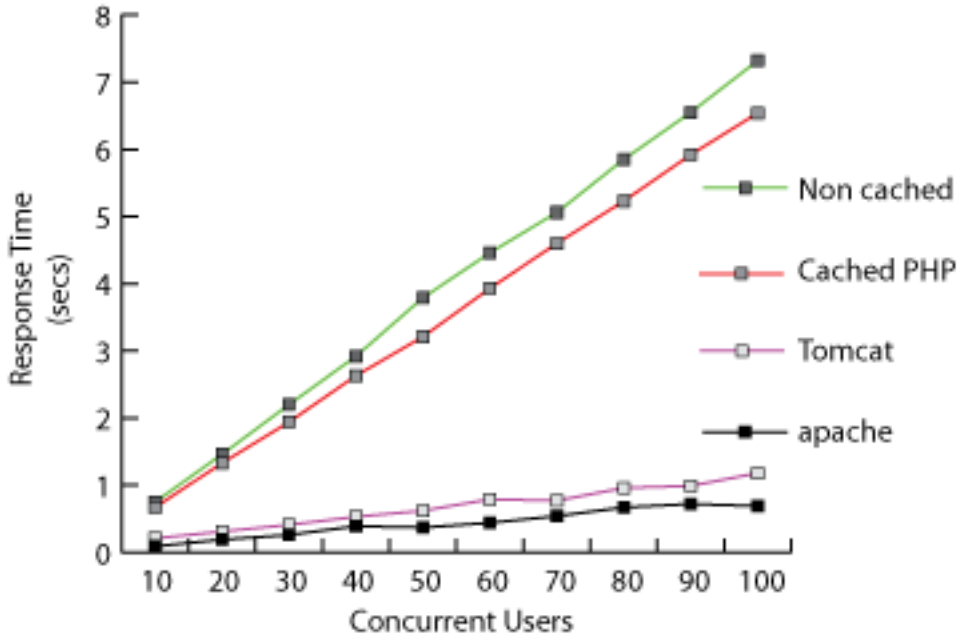
So how do we improve this situation? After much testing an interesting conclusion surfaced: that caching the sitemap in Paloose (while undoubtedly providing some speed increase) was not the most effective way of improving things.

I rewrote a substantial part of Paloose to cache the sitemap using precompiled versions. The sitemap parsers were changed to provide a code generation function to produce an equivalent PHP representation of the sitemap XML. For example the sitemap above would be compiled to the following PHP (or very similar):

```
<?php

   class CachedSitemap {

      private $gRequestParameters = array('url'=>'index.html','resource'=>'index.html',);
      private $gParameters = array();

      function __construct()
      {
      }

      public function run( $inURL, $inQueryString, $inInternalRequest )
      {
         global $gVariableStack;
         global $gSitemapStack;

         $sitemap = new Sitemap_3858cff740af348b8a52174be329505d();
         $gSitemapStack->push( $sitemap );
         $sitemap->run( $inURL, $inQueryString, $inInternalRequest );
         $gSitemapStack->pop();
      }
   }
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/generation/FileGenerator.php' );
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/transforming/TRAXTransformer.php' );
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/transforming/I18nTransformer.php' );
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/serialization/HTMLSerializer.php' );
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/serialization/XHTMLSerializer.php' );
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/serialization/TextSerializer.php' );
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/serialization/XMLSerializer.php' );
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/reading/ResourceReader.php' );
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/matching/WildcardURIMatcher.php' );
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/matching/RegexpURIMatcher.php' );
   require_once( '/var/www/html/simpleSite/../paloose-cached/lib/selection/BrowserSelector.php' );
   class Sitemap_3858cff740af348b8a52174be329505d {
      private $gOutputStream;

      function __construct()
      {
         $this->gOutputStream = new OutputStream( OutputStream::STANDARD_OUTPUT );
      }

      public function run( $inURL, $inQueryString, $inInternalRequest ) {
       global $gVariableStack;
    try { // Pipelines parse
   { // Pipeline parse
   if ( ( $matchArray = Match::match( 'WildcardURIMatcher', $inURL, '**.html' ) ) != NULL ) {
      $gVariableStack->push( $matchArray );
      $dom = GeneratorPipeElement::generate( 'FileGenerator', 'context://content/{1}.xml',
                'xml-content', $matchArray, $this->gRequestParameters );
   $this->gParameters = new Parameter();
   $this->gParameters->setParameterList( $this->gRequestParameters );
   $this->gParameters->setParameter( 'page', '{1}' );
   $dom = TransformerPipeElement::transform( 'TRAXTransformer',
   'context://resources/transforms/page2html.xsl',
   $dom, $label, $matchArray, $this->gParameters, $gVariableStack );
```

```
   $this->gParameters = new Parameter();
   $this->gParameters->setParameterList( $this->gRequestParameters );
   $dom = TransformerPipeElement::transform( 'TRAXTransformer',
   'context://resources/transforms/stripNamespaces.xsl',
   $dom, $label, $matchArray, $this->gParameters, $gVariableStack );
   $this->gParameters = new Parameter();
   $dom = SerializerPipeElement::serialize( 'HTMLSerializer',
   $dom, $label, $matchArray, $this->gParameters, $this->gOutputStream );
   $gVariableStack->pop(); }
   } // Pipeline parse
   } catch ( ExitException $e ) { // Pipelines parse
               throw new ExitException();
            } catch( UserException $e ) {
               // handle error pipeline
            } catch( RunException $e ) {
               // handle error pipeline
               $dom = GeneratorPipeElement::generate( 'FileGenerator', 'context://content/error.xml',
               '', $matchArray, $this->gRequestParameters );
   $this->gParameters = new Parameter();
   $this->gParameters->setParameterList( $this->gRequestParameters );

   $dom = TransformerPipeElement::transform( 'TRAXTransformer',
   'context://resources/transforms/error2html.xsl',
   $dom, $label, $matchArray, $this->gParameters, $gVariableStack );
   $this->gParameters = new Parameter();

   $dom =    $dom = SerializerPipeElement::serialize( 'HTMLSerializer',
   $dom, $label, $matchArray, $this->gParameters, $this->gOutputStream );}}
            }

?>
```

Not the prettiest code, but compiled code is not designed to be. Running this cached base system gave the following results:



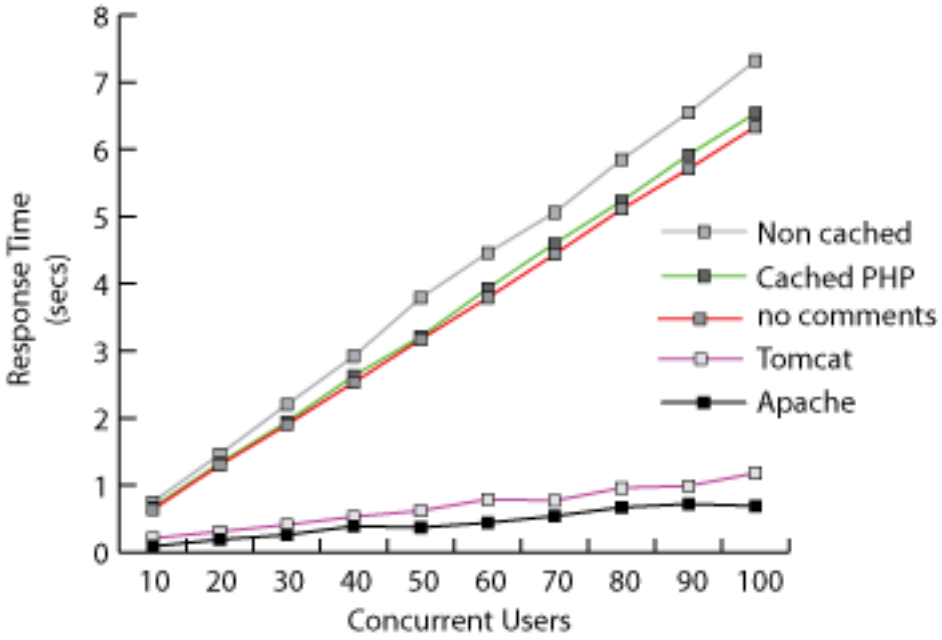
It was clear that exploring other avenues would be more fruitful. After a couple of weeks of trials I ended up with the conclusion that looking at the design of the Paloose and its PHP was the best way to continue. Changing the XML, XSLT and sitemap of a site really did not have as much effect as I wanted.

The next page describes these changes and their implications.

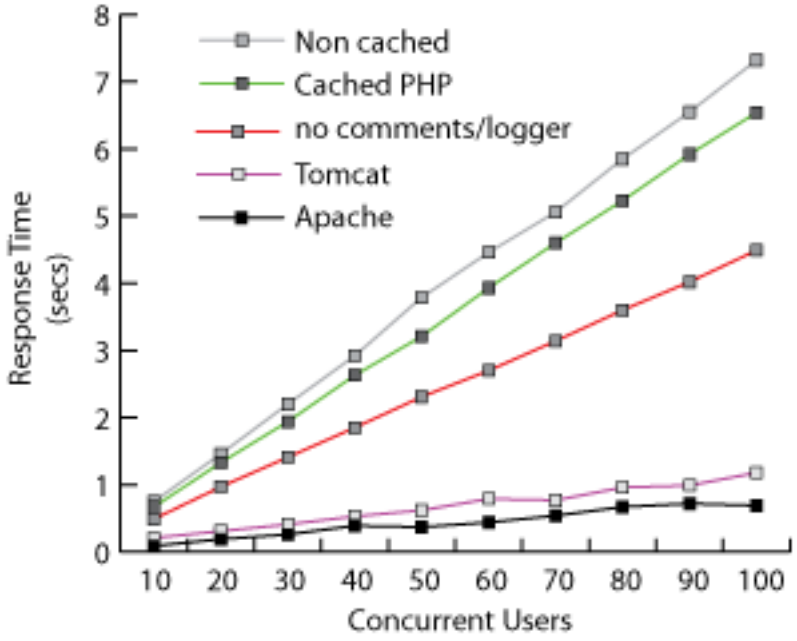
40 Optimising Paloose Performance

40 Optimising Paloose Code

To start with I produced a non-cached version of Paloose with no comments. As I suspected this gave little or no advantage over the cached version.



The next trial was to remove the Logging — certainly a little dangerous on a development system but possibly an allowable risk on a mature system when speed is essential. The result of this was illuminating. The speed increase was substantial.

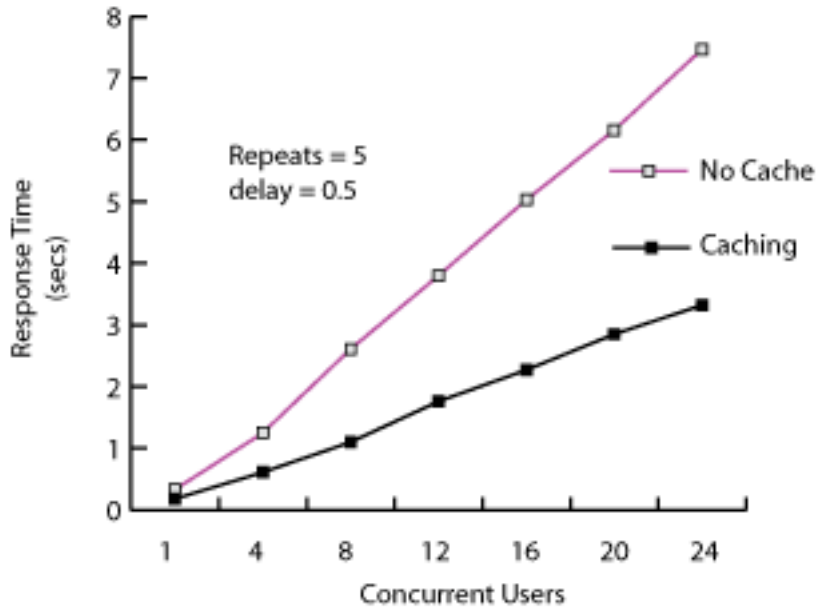


The main reason for this is the amount of code needed to be included from the `log4php` system. In particular every time a *require_once* statement is invoked it degrades the performance, not unexpected.

There were very few other optimisations that gave the same improvement of speed.

40 Using a Data Cache

The final way I tried with the caching was a data cache within the pipeline. The current version (after 1.3.0) has this scheme in some of the generators and transformers. The theory is that each stage is responsible for maintaining a cache of the data that it outputs. In the case of a transformer the output is taken from the cache dependent on several things: the input DOM being valid, the XSL file being up to date and the parameters being valid. Valid implies that there has been no change since the last access. I am not wholly convinced that the system is useful except in cases where there are a large amount of transformations to be done — either as a number of sequential transforms of a single large one. For example I tried loading my Hop Vine site catalogue page on a local server and which uses up to 6 transformers to run. The results are relatively encouraging:



However I am not sure whether the added complexity is worth it: any thoughts from anyone out there?

40 Conclusions for Paloose Web Site Design

So what is the best way to proceed? If you want huge speed increases that Tomcat/Apache give you then use them not Paloose. However it is possible to provide some guidance on small sites.

- **Do not use sub-sitemaps unless there is a real necessity.** Each time a new sub-sitemap is mounted it must be parsed. Not a huge overhead from experimental evidence but one nonetheless.
- **Do not declare more components in the sitemap than you need.** Each component requires some PHP code to be included (the component Class and its subsidiary classes). As we have seen this is the biggest overhead.

- **Use a "delogged" version of Paloose.** A little dangerous but it does give a good increase in speed. The question is: how do you get a version of Paloose without the logging? I am going to release a version of Paloose on each code release that does not use the logger and has no logging statements at all — treat with caution.

41 Caching Discussion

41 Introduction

First of all it is worth asking exactly why we might want to use caching. The basic answer is to achieve a speed increase and thus a performance increase (more hits per second or concurrent hits). In a system like Paloose (and Tomcat/Cocoon) performance increase can also come from alternate page serving techniques. In these, a static page server such as Apache is used (and essential) to take the burden from the more dynamic pages with which Paloose deals. Thus images are a good candidate for bypassing Paloose and being served directly from the Apache server.

So what are characteristics of a dynamic page over a static page? Simplistically they are:

- resources that change (a database query for example), or
- resources that are dependent on user input, or
- resources that are made up from fragments (transformed from XML documents)

Images (jpeg, png files etc) are static (in general) because they do not require any form of modification due to user input or database queries, and thus can be safely served by the Apache server.

Anything, therefore, that can speed up the process of transforming user data and other inputted variables has to be good. As a further complication, because of the stateless, on-demand nature of the servers there is also the question of the actual server code. In Paloose this is made worse by using a language such as PHP5 which is an interpreted language. Cocoon and Tomcat are a compiled into intermediate code language (not wholly always accurate but for the purposes of the argument true). Apache is a compiled solution and so is running without these restrictions. In an interpreted language such as PHP5 the code has to be translated each time into a runnable form. In modern systems there is a natural caching (persistence) which helps with tis process: frequently used code is kept in memory for use next time. However, it is impossible to rely on this being there all the time.

41 Caching Code

Caching the code is the primary way of overcoming the problems of interpreters. With Paloose this would clearly be possible, although I have not tried this it remains a potential option for future work. I have shown elsewhere that by judicious control of the basic Paloose code (rather than caching it) some considerable performance increase can be gained. However this is at the cost of code clarity (no comments) and missing functionality (no logging). While the former may be acceptable the latter probably is not.

41 Caching the Sitemap

Paloose works by interpreting the sitemap and building an internal structure of Paloose components and pipelines representing the sitemap. Unfortunately this is done each time a request is made giving a substantial performance penalty. One solution that I tried was to precompile the sitemap into a PHP5 representation which is then run (and can be cached). Curiously the increase in performance was not as much as compared to the Paloose code. One advantage of this technique is that it is not dependant on user input or changing XML pages or database queries.

41 Caching the Page Components

Within the pipelines, components take a variety of resources to make up the final deliverable page. These inputs are various:

- the input query from the user which consists of the requested resource (the page) and the query string of parameters.
- data as a results of data base queries.
- the XML or text fragments that make up the page.

The pipeline can be considered to be a state machine that outputs data dependant directly on the input. Unlike most (useful) state machines it has no persistent state between requests. Each request is considered to be fresh. The server/client arrangement with Web pages gets over this by using cookies. However this solution is not available in all cases (not everyone has cookies enabled). Thus we need to characterise a request purely on the basis on the input conditions for that request.

On top of the problems of changing inputs, the state of the server depends on:

- **The sitemap (has it changed since the last request?)** — We do not need to check this as a change her will cause all the other conditions to fail. If they do not then the cached data can be safely used.
- **The XML fragments (have they changed?)** — The most efficient way of achieving this is to note the latest modification time of the XML file and compare it with the previous one. However the previous time will have to be held in between requests in some form. However this is not the complete story for pipeline elements in the sitemap that do not take and external file (a transformer for example). In this case we need to inspect the inputted DOM, a little more tricky.
- **The XSL transformation file (has it been changed?)** — Again the most efficient was of doing this is via a timestamp.
- **The query string submitted with the request (how have these parameters changed?)** — we cannot use a timestamp here and so some form of hash is required.
- **Response from an SQL data base query (how may the results have changed?)** — things that are dependant on these types of external queries obviously cannot be cached suitably as they are outside control.

41 Checks and Balances

Adding a caching system causes extra code to be introduced. This extra code can offset the advantages that might be gained by using a cache system. So careful testing should be used when deciding to use a cache system.

Data caches in the Paloose pipeline might (and I only say might) be of benefit where external influences are greatest: for example if the XSL transformations are particularly large or there are many of them.

Warning

One final warning. Because the caching mechanism does not pick up all the changes in the source files (for example included XML or XSL files) it is important to turn off the caching when developing Paloose sites. It is very easy to make changes and not see them reflected in the HTML output. I have learned this the hard way. I also suggest that when a change has been made to a live site that the cache be cleared to allow the new changes to filter through.

41 Displaying All Composers

41 The XML File

The XML file contains the embedded SQL command for getting all the data:

displayComposers.html

```
<page:content>
   <t:heading level="1">SQL Display All Results</t:heading>

   <execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
      <query name="displayAll" database="composers">select * from composer</query>
   </execute-query>

   <t:p><link:link type="uri" ref="examples/sql/sql.html">Return to SQL index</link:link></t:p>
   <t:p><link:link type="uri" ref="examples/sql/logout.html">Logout</link:link> from admin pages.</t:p>
</page:content>
```

Note that only the content part of XML page is shown for brevities sake.

41 Extending the Sitemap

Only a single matcher in the sitemap is need for this, together with a declaration of the necessary transforms:

sitemap.xmap

```
<map:transformers default="xslt">
   <map:transformer name="xslt" src="resource://lib/transforming/TRAXTransformer" cachable="no">
      <map:use-request-parameters>true</map:use-request-parameters>
   </map:transformer>
   <map:transformer name="mysql" src="resource://lib/transforming/SQLTransformer">
       <map:parameter name="type" value="mysql"/>
       <map:parameter name="host" value="localhost:3306"/>
       <map:parameter name="user" value="hsfr"/>
       <map:parameter name="password" value="xxxxxxxx"/>
    </map:transformer>
</map:transformers>

...

<map:pipeline>

      <map:match pattern="sql.html">
         <map:aggregate element="root" label="aggr-content">
            <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
            <map:part src="cocoon:/sql.xml" element="content" strip-root="true"/>
         </map:aggregate>
         <map:call resource="outputPage"/>
      </map:match>

      <map:match pattern="displayAllComposers.html">
         <map:aggregate element="root" label="aggr-content">
            <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
            <map:part src="cocoon:/displayComposers.xml" element="content" strip-root="true"/>
         </map:aggregate>
         <map:transform type="mysql" label="sql-content">
            <map:parameter name="show-nr-of-rows" value="true"/>
         </map:transform>
         <map:transform src="context://resources/transforms/sql2xml.xsl" label="sql-transform"/>
         <map:call resource="outputPage"/>
      </map:match>

</map:pipeline>
```

41 Returned Data

After the *mysql* transformer the document is:

```
<page:content>
    <t:heading level="1">SQL Display All Results</t:heading>

    <default:row-set nrofrows="3" name="displayAll" xmlns="http://apache.org/cocoon/SQL/2.0">
      <default:row>
         <default:id>16</default:id>
         <default:name>Dvo\v r\‘ ak</default:name>
         <default:forenames>Antonin</default:forenames>
         <default:birth>1841-09-08</default:birth>
         <default:death>1904-05-01</default:death>
      </default:row>
      <default:row>
         <default:id>15 </default:id>
         <default:name>Mozart</default:name>
         <default:forenames>Wolfgang Amadeus</default:forenames>
         <default:birth>1756-01-27</default:birth>
         <default:death>1791-12-05</default:death>
      </default:row>
      <default:row>
         <default:id>13</default:id>
         <default:name>Barber</default:name>
         <default:forenames>Samuel</default:forenames>
         <default:birth>1910-03-09</default:birth>
         <default:death>1981-01-23</default:death>
      </default:row>
    </default:row-set>
    <t:p>
       <link:link type="uri" ref="examples/sql/sql.html">Return to SQL index</link:link>
    </t:p>
    <t:p>
       <link:link type="uri" ref="examples/sql/logout.html">Logout</link:link> from admin pages.
    </t:p>
</page:content>
```

41 Processing Data

To process this there is a transformer to change to the XML that the *outputPage* requires:

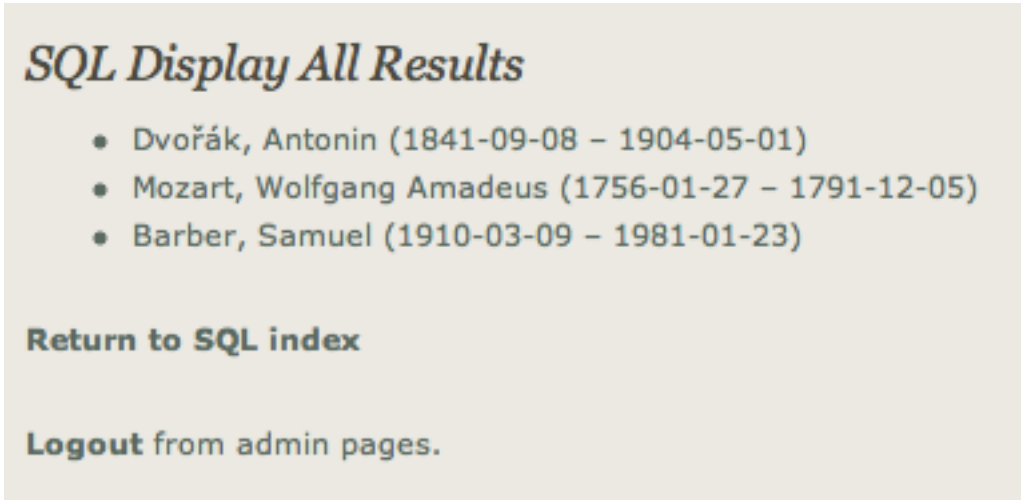
```
<?xml version="1.0" encoding="UTF-8"?>

<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:list="http://www.hsfr.org.uk/Schema/List"
   xmlns:t="http://www.hsfr.org.uk/Schema/Text"
   xmlns:sql="http://apache.org/cocoon/SQL/2.0"
   version="1.0">

   <xsl:template match="sql:row-set[ @name = 'displayAll' ]">
      <xsl:element name="list:list">
         <xsl:attribute name="type">unordered</xsl:attribute>
         <xsl:for-each select="sql:row">
            <xsl:element name="list:item">
               <xsl:value-of select="sql:name"/>
               <xsl:text>, </xsl:text>
               <xsl:value-of select="sql:forenames"/>
               <xsl:choose>
                  <xsl:when test="not( sql:birth='' and sql:death='' )">
                     <xsl:text> (</xsl:text>
                     <xsl:value-of select="sql:birth"/>
                     <xsl:text> &#x2013; </xsl:text>
                     <xsl:value-of select="sql:death"/>
                     <xsl:text>)</xsl:text>
                  </xsl:when>
                  <xsl:when test="not( sql:birth='' )">
                     <xsl:text> (b.</xsl:text>
                     <xsl:value-of select="sql:birth"/>
                     <xsl:text>)</xsl:text>
                  </xsl:when>
```

```
                  <xsl:when test="not( sql:death='' )">
                     <xsl:text> (d.</xsl:text>
                     <xsl:value-of select="sql:death"/>
                     <xsl:text>)</xsl:text>
                  </xsl:when>
               </xsl:choose>
            </xsl:element>
         </xsl:for-each>
      </xsl:element>
   </xsl:template>

   <!-- Soak up any remaining elements not processed by the above -->

   <xsl:template match="node()|@*" priority="-1">
      <xsl:copy>
         <xsl:apply-templates select="@*"/>
         <xsl:apply-templates/>
      </xsl:copy>
   </xsl:template>

</xsl:stylesheet>
```

After processing the document becomes (suitably indented by me):

```
<page:content>
   <t:heading level="1">SQL Display All Results</t:heading>

      <list:list xmlns:list="http://www.hsfr.org.uk/Schema/List" type="unordered">
         <list:item>Dvo\v r\‘ ak, Antonin (1841-09-08 --- 1904-05-01)</list:item>
         <list:item>Mozart, Wolfgang Amadeus (1756-01-27 --- 1791-12-05)</list:item>
         <list:item>Barber, Samuel (1910-03-09 --- 1981-01-23)</list:item>
      </list:list>

   <t:p><link:link type="uri" ref="examples/sql/sql.html">Return to SQL index</link:link></t:p>
   <t:p><link:link type="uri" ref="examples/sql/logout.html">Logout</link:link> from admin pages.</t:p>
   </page:content>
```

The output of this appears as:



41 Adding Composers

41 Add Composer Form

In order to add composers we need a form to collect the data. A simple Paloose form together with a suitable flow script is needed. The initial form is encoded in a file:

examples/sql/addComposer-1.xml

```
<?xml version="1.0" encoding="UTF-8"?>

<page:page
   xmlns:t="http://www.hsfr.org.uk/Schema/Text"
   xmlns:page="http://www.hsfr.org.uk/Schema/Page"
   xmlns:list="http://www.hsfr.org.uk/Schema/List"
   xmlns:link="http://www.hsfr.org.uk/Schema/Link"
   xmlns:pf="http://www.paloose.org/schemas/Forms/1.0">

   <page:meta>
      <page:title>Paloose &#x2014; SQL Example</page:title>
      <page:copyright>Copyright 2006 &#x2013; 2010 Hugh Field-Richards.
         All Rights Reserved.</page:copyright>
   </page:meta>

   <page:content>
      <t:heading level="1">SQL Add Composer</t:heading>

      <pf:form id="addComposerForm" flow="addComposer"
            continuation="{flow:continuation.id}" session="{flow:__flowId}">
         <pf:label>Enter composer's details:</pf:label>

         <pf:input ref="name" class="nameField">
            <pf:label>Composer name</pf:label>
            <pf:value>{flow:name}</pf:value>
            <pf:hint>The composer's surname</pf:hint>
            <pf:violations/>
         </pf:input>

         <pf:input ref="firstNames" class="firstNamesField">
            <pf:label>First names</pf:label>
            <pf:value>{flow:firstNames}</pf:value>
            <pf:hint>The composer's first names</pf:hint>
            <pf:violations/>
         </pf:input>

         <pf:input ref="birth" class="birthField">
            <pf:label>Birth date</pf:label>
            <pf:value>{flow:birth}</pf:value>
            <pf:hint>The composer's birth date (yyyy-mm-dd)</pf:hint>
            <pf:violations/>
         </pf:input>

         <pf:input ref="death" class="deathField">
            <pf:label>Death date</pf:label>
            <pf:value>{flow:death}</pf:value>
            <pf:hint>The composer's death date (yyyy-mm-dd)</pf:hint>
            <pf:violations/>
         </pf:input>

         <pf:submit class="button" id="next">
            <pf:label>Next</pf:label>
            <pf:hint>Check details</pf:hint>
         </pf:submit>
      </pf:form>

      <t:p><link:link type="uri" ref="examples/sql/sql.html">Return
         to SQL index</link:link></t:p>
      <t:p><link:link type="uri" ref="examples/sql/logout.html">Logout</link:link>
         from admin pages.</t:p>
   </page:content>
</page:page>
```

41 The Flow Script

There must be an associated flow script to support this form:

resources/scripts/AddComposer.php

```
<?php

   require_once( PHP_DIR . "/flows/Continuations.php" );

   // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

   class AddComposer extends Continuations {

      /** Logger instance for this class */
      private $gLogger;

      private $gAddComposerModel = array (
         "name" => "",
         "firstNames" => "",
         "birth" => "",
         "death" => "",
      );

      // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
      // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
      /**
       * Make a new instance of the continuation.
       *
       * @throws RunException if there is no current session in progress.
       */

      function __construct()
      {
               $this->gLogger = Logger::getLogger( __CLASS__ );
          parent::__construct( $this->gAddComposerModel );
      }

      // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
      // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

      function add()
      {
           global $gModules;

               $this->gLogger->debug( "Adding entry, continuation: " . $this->gContinuation );
         $requestParameterModule = $gModules[ 'request-param' ];
         $this->gViolations = array();

         $finished = false;
         while ( !$finished ) {
            $errors = false;
                  switch ( $this->gContinuation ) {

                     case 0 :
                        $this->sendPage(
                           "addComposer-1.html",      // The page to send to the client
                           $this->gAddComposerModel,  // The current data model to send
                           ++$this->gContinuation,    // The continuation link for the next stage
                           $this->gViolations );      // Array of errors (keyed by data model keys)
                        $finished = true;
                        break;

               // Further cases

               } // switch
         } // while
      } // AddComposer::add
   } // AddComposer
?>
```

41 Extending the Sitemap

The sitemap must declare the script:

examples/sql/sitemap.xmap

```
<map:flow language="php">
   <map:script src="context://resources/scripts/AddComposer.php"/>
</map:flow>
```

The sitemap add composer pipeline is in two parts: an external accessed one and an internal one:

examples/sql/sitemap.xmap

```
<map:pipeline>

     <map:match pattern="addComposer.html">
        <map:call function="AddComposer::add"/>
     </map:match>

     <map:match pattern="addComposer.kont">
        <map:call function="AddComposer::add"/>
     </map:match>

  </map:pipeline>

  <!-- Only called from the add user script AddComposer::add -->

  <map:pipeline internal-only="true">

     <map:match pattern="addComposer-*.html">
        <map:aggregate element="root" label="aggr-content">
           <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
           <map:part src="cocoon:/addComposer-{1}.px" element="content" strip-root="true"/>
        </map:aggregate>
        <map:transform src="resource://resources/transforms/pforms-violations.xsl" label="pforms-violations">
           <map:parameter name="formViolations" value="{session:__violations}"/>
        </map:transform>
        <map:transform src="resource://resources/transforms/pforms-default.xsl" label="pforms-default"/>
        <map:transform src="resource://resources/transforms/pforms2html.xsl" label="pforms-html"/>
        <map:call resource="outputPage"/>
     </map:match>

     <map:match pattern="menus.xml">
        <map:generate src="cocoon:/../menus.xml" label="menus-content"/>
        <map:serialize/>
     </map:match>

     <map:match pattern="*.xml">
        <map:generate src="cocoon:/{1}.xml" label="xml-content"/>
        <map:serialize/>
     </map:match>

     <map:match pattern="*.px">
        <map:generate type="px" src="cocoon:/{1}.xml" label="px-content"/>
        <map:serialize/>
     </map:match>

  </map:pipeline>
```

The important thing here is that the *PXTemplateGenerator* must be used for the flow pages.

Using the link “Create Composer” (http://¡host¿/examples/sql/addComposer.html) the form for entering a new composer is presented:


SQL Add Composer
Enter composer's details:
Composer name
First names
Birth date
Death date
Next
Return to SQL index
Logout from admin pages.

41 Confirming the Entered Data

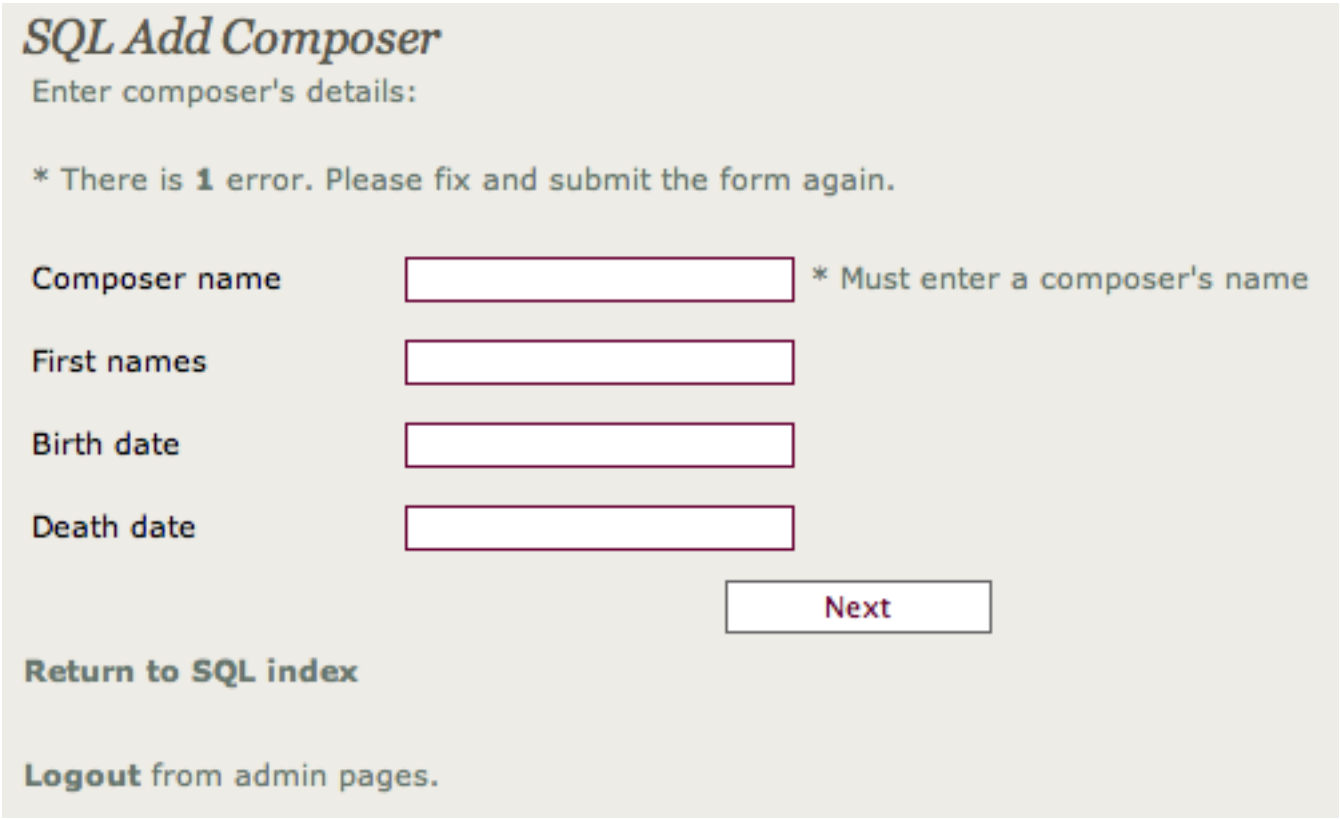
The first thing is to check the data being entered. We have to add the next case in the continuations to check on the entered data:

resources/scripts/AddComposer.php

```
class AddComposer extends Continuations {

   /** Logger instance for this class */
   private $gLogger;

   private $gAddComposerModel = array (
      "name" => "",
      "firstNames" => "",
      "birth" => "",
      "death" => "",
   );

   private $dataRequired = array (
      "name" => "Must enter a composer's name",
   );

   ...

   // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   /**
    */

   function add()
   {
      global $gModules;

      $this->gLogger->debug( "Adding entry, continuation: " . $this->gContinuation );
      $requestParameterModule = $gModules[ 'request-param' ];
      $this->gViolations = array();

      $finished = false;
      while ( !$finished ) {
         $errors = false;
          switch ( $this->gContinuation ) {

             case 0 :
                $this->sendPage(
                   "addComposer-1.html",        // The page to send to the client
                   $this->gAddComposerModel,    // The current data model to send
                   ++$this->gContinuation,      // The continuation link for the next stage
                   $this->gViolations );        // Array of errors (keyed by data model keys)
                $finished = true;
                break;

             case 1 :
                // Error processing here
                // First I have to check that there are contents in the required fields. I could leave
                // out the "isset" but some PHP systems will have strict error checking
                // and produce extraneous error messages on the page.
                foreach ( $this->dataRequired as $key => $msg ) {
                   if ( !isset( $this->gAddComposerModel[ $key ] ) or
                         strlen( $this->gAddComposerModel[ $key ] ) == 0 ) {
                     $this->gViolations[ $key ] = $msg;
                     $errors = true;
                   }
                 }

                 // Now check for remaining errors. Adjust the order of the checks
                 // to suit local conditions.
                 if ( $errors === false ) {
                    if ( $this->gAddComposerModel[ 'birth' ] >= $this->gAddComposerModel[ 'death' ] and
                          $this->gAddComposerModel[ 'birth' ] != "" and
                          $this->gAddComposerModel[ 'death' ] != "" ) {
                       $this->gViolations[ 'birth' ] = "Birth is later than death";
                       $errors = true;
```

```
                }
            }
            if ( $errors ) {
               $this->gContinuation--;
               $finished = false;
            } else {
              // No errors so send next page
               $this->sendPage(
                  "addComposer-2.html",
                  $this->gAddComposerModel,
                  ++$this->gContinuation,
                  $this->gViolations );
                $finished = true;
             }
            break;

        }
    }
}
```

If an error occurs (such as a blank form) then the following will be output:



41 Confirm Add Composer Form

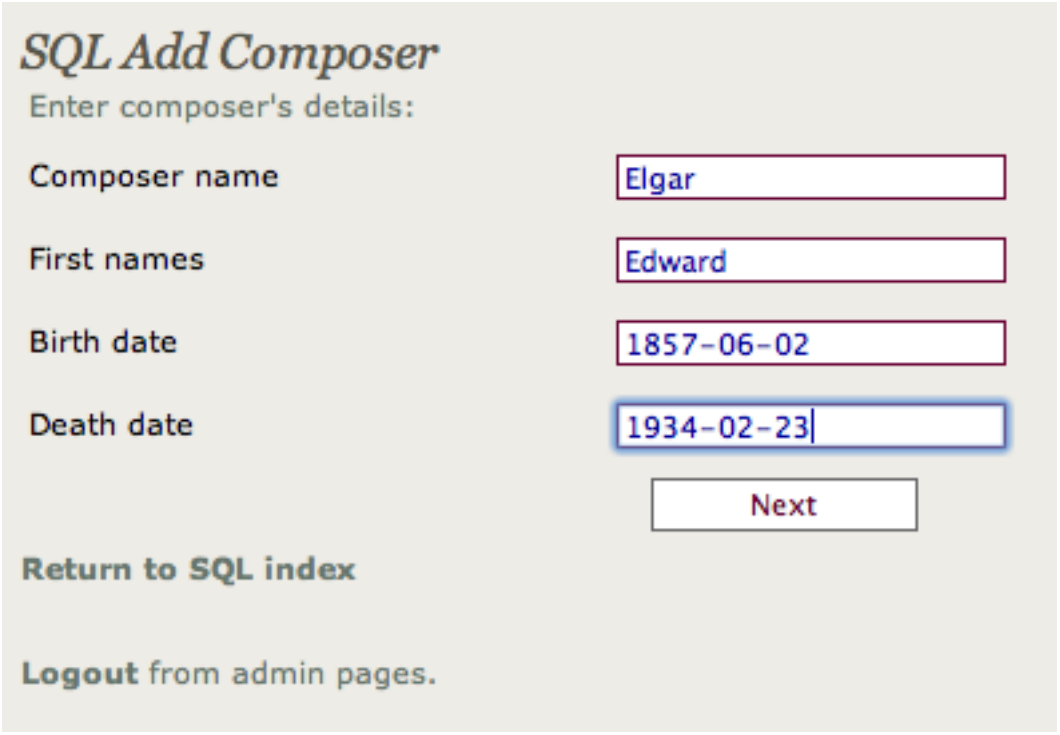
Assuming that there are no errors the flowscript redirects to the page, `addComposer-2.html`. The sitemap does not need to change but the XML file is:

examples/sql/addComposer-2.xml

```
<page:content>

   <t:heading level="1">SQL Add Composer</t:heading>

   <pf:form id="addComposerForm" flow="addComposer"
```

```
                continuation="{flow:continuation.id}"
                session="{flow:__flowId}">
        <pf:label>Press confirm to accept new composer:</pf:label>

        <pf:output ref="name">
           <pf:label>Composer name:</pf:label>
           <pf:value>{flow:name}</pf:value>
        </pf:output>

        <pf:output ref="firstNames">
           <pf:label>First names:</pf:label>
           <pf:value>{flow:firstNames}</pf:value>
        </pf:output>

        <pf:output ref="birth">
           <pf:label>Birth date:</pf:label>
           <pf:value>{flow:birth}</pf:value>
        </pf:output>

        <pf:output ref="death">
           <pf:label>Death date:</pf:label>
           <pf:value>{flow:death}</pf:value>
        </pf:output>

        <pf:submit class="button" id="prev">
           <pf:label>Back</pf:label>
           <pf:hint>Go to previous page</pf:hint>
        </pf:submit>

        <pf:submit class="button" id="next">
           <pf:label>Confirm</pf:label>
           <pf:hint>Confirm new composer</pf:hint>
        </pf:submit>
     </pf:form>

     <t:p>
        <link:link type="uri" ref="examples/sql/sql.html">Return to SQL index</link:link>
     </t:p>
     <t:p><link:link type="uri" ref="examples/sql/logout.html">Logout</link:link> from admin pages.</t:p>

  </page:content>
```

which provides a confirmation page. Entering the following data:



Gives the output:

41 Writing the Data

We have to add the next case in the continuations to handle the correct data:

resources/scripts/AddComposer.php

```
class AddComposer extends Continuations {

   function add()
   {
      global $gModules;

      $this->gLogger->debug( "Adding entry, continuation: " . $this->gContinuation );
      $requestParameterModule = $gModules[ 'request-param' ];
      $this->gViolations = array();

      $finished = false;
      while ( !$finished ) {
         $errors = false;
         switch ( $this->gContinuation ) {

            case 0 :
               ...
               break;

            case 1 :
               ...
               break;

            case 2 :
               // No errors to record from optional data so send next page
               $this->sendPage(
                  "addComposer-3.html",
                  $this->gAddComposerModel,
                  ++$this->gContinuation,
                  $this->gViolations );
                $finished = true;
                break;

         }
      }
   }
```

41 Constructing the SQL Command

Now that there is data to add a suitable SQL entry command has to be created:

examples/sql/addComposer-3.xml

```
   <page:content>
      <t:heading level="1">SQL Add Composer</t:heading>

      <execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
         <query name="createEntry" database="composers">INSERT INTO composer ( name, forenames, birth, death )
            VALUES ( "{flow:name}", "{flow:firstNames}", "{flow:birth}", "{flow:death}" )</query>
      </execute-query>

      <t:p><link:link type="uri" ref="examples/sql/sql.html">Return to SQL index</link:link></t:p>
      <t:p><link:link type="uri" ref="examples/sql/logout.html">Logout</link:link> from admin pages.</t:p>
   </page:content>
```

Note the use of the variables that take the flow script variables and are expanded when the *PXTemplateGenerator* inputs the file.

41 Extending the Sitemap

A suitable addition is required to the sitemap:

examples/sql/sitemap.xmap

```
<map:pipeline>

   <map:match pattern="addComposer.html">
      <map:act type="auth-protect">
         <map:parameter name="handler" value="adminHandler"/>
         <map:call function="AddComposer::add"/>
      </map:act>
   </map:match>

   <map:match pattern="addComposer.kont">
      <map:call function="AddComposer::add"/>
   </map:match>

</map:pipeline>

<!-- Only called from the add user script AddComposer::add -->

<map:pipeline internal-only="true">

   <map:match pattern="addComposer-3.html">
      <map:aggregate element="root" label="aggr-content">
         <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
         <map:part src="cocoon:/addComposer-3.px" element="content" strip-root="true"/>
      </map:aggregate>
      <map:transform type="mysql" label="sql-content"/>
      <map:transform src="context://resources/transforms/sql2xml.xsl" label="sql-transform"/>
      <map:call resource="outputPage"/>
   </map:match>

   <map:match pattern="addComposer-*.html">
      <map:aggregate element="root" label="aggr-content">
         <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
         <map:part src="cocoon:/addComposer-{1}.px" element="content" strip-root="true"/>
      </map:aggregate>
      <map:transform src="resource://resources/transforms/pforms-violations.xsl" label="pforms-violations">
         <map:parameter name="formViolations" value="{session:__violations}"/>
      </map:transform>
      <map:transform src="resource://resources/transforms/pforms-default.xsl" label="pforms-default"/>
      <map:transform src="resource://resources/transforms/pforms2html.xsl" label="pforms-html"/>
      <map:call resource="outputPage"/>
   </map:match>

</map:pipeline>
```

41 Processing Result

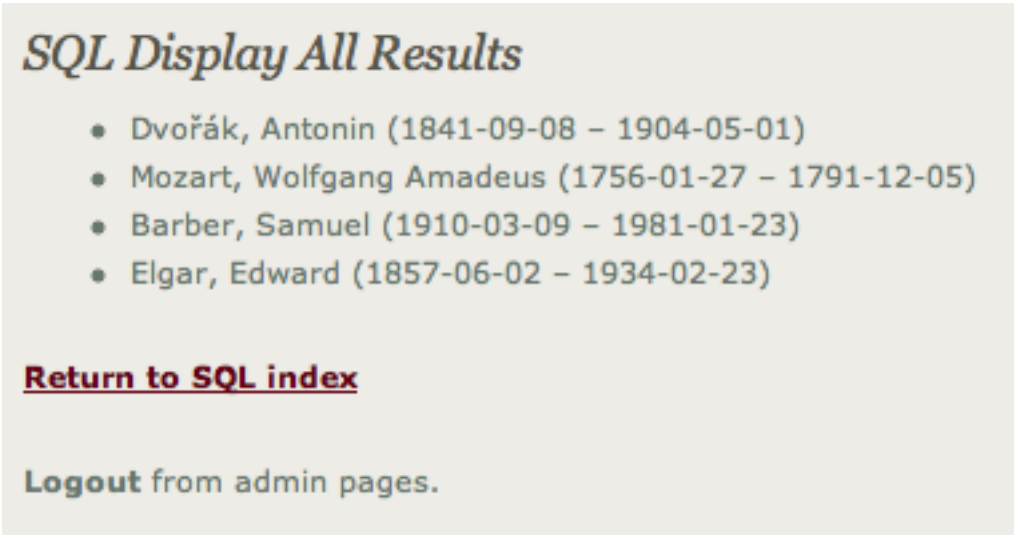
Note that we do not need the PForm transforms in this matcher. Only a transformer that contains the necessary transform to process the returned confirmation data from the SQL server about the success or faiulure of the write data command is required.

resources/transforms/sql2xml.xsl

```
<xsl:template match="sql:result[ . = 'true' ]">
      <xsl:element name="t:p">
         <xsl:text>Operation complete!</xsl:text>
      </xsl:element>
   </xsl:template>

   <xsl:template match="sql:result">
      <xsl:element name="t:p">
         <xsl:text>Problem with operation: </xsl:text>
         <xsl:value-of select="."/>
      </xsl:element>
   </xsl:template>
```

The transforms that are used are relatively crude, but effective. A succesful write would appear as:

Displaying the data now gives:



41 Deleting Composers

Deleting entries requires a similar approach to adding, but with a couple of interesting features. First we must construct a form to present the composers in a form that we can select one to delete. The content file is:

examples/sql/deleteComposer-1.xml

```
<page:content>
   <t:heading level="1">SQL Delete Entry</t:heading>

   <pf:form id="deleteComposerForm"
            flow="deleteComposer"
            continuation="{flow:continuation.id}"
            session="{flow:__flowId}">
      <pf:label>Select composer to delete:</pf:label>

         <execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
            <query name="selectDeleteComposer" database="composers">select * from composer</query>
         </execute-query>

      <pf:submit class="button" id="next">
         <pf:label>Next</pf:label>
         <pf:hint>Check details</pf:hint>
      </pf:submit>
   </pf:form>

   <t:p><link:link type="uri" ref="examples/sql/sql.html">Return to SQL index</link:link></t:p>
   <t:p><link:link type="uri" ref="examples/sql/logout.html">Logout</link:link> from admin pages.</t:p>
</page:content>
```

41 The Flow Script

The contents of the form are not known yet. The SQL query will bring back a list of the composers that can be used as a selector by the form. In order to process this we need a flow script:

resources/scripts/DeleteComposer.php

```
class DeleteComposer extends Continuations {

   /** Logger instance for this class */
   private $gLogger;

   private $gDeleteComposerModel = array (
      "selectDeleteComposer" => "",  // Should be same as the name in the form
   );

   // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   /**
    * Make a new instance of the continuation.
    *
    * @throws RunException if there is no current session in progress.
    */

   function __construct()
   {
       $this->gLogger = Logger::getLogger( __CLASS__ );
       parent::__construct( $this->gDeleteComposerModel );
   }

   // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   // -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   /**
    */

   function delete()
   {
      global $gModules;
```

```
    $this->gLogger->debug( "Delete composer, continuation: " . $this->gContinuation );
    $this->gViolations = array();

    $finished = false;
    while ( !$finished ) {
      $this->gLogger->debug( "Processing: " . $this->gContinuation );
      $errors = false;
      switch ( $this->gContinuation ) {

          case 0 :
              $this->sendPage(
                 "deleteComposer-1.html",      // The page to send to the client
                 $this->gDeleteComposerModel,  // The current data model to send
                 ++$this->gContinuation,       // The continuation link for the next stage
                 $this->gViolations );         // Array of errors (keyed by data model keys)
              $finished = true;
              break;

          case 1 :
             ...
              break;

          case 2 :
             ...
             break;

        } // switch
    } // while
} // function delete()
```

41 Extending the Sitemap

The additions to the sitemap are similar to the add composer case

examples/sql/sitemap

```
<map:flow language="php">
   <map:script src="context://resources/scripts/AddComposer.php"/>
   <map:script src="context://resources/scripts/DeleteComposer.php"/>
</map:flow>

...

<map:pipeline>

   <map:match pattern="deleteComposer.html">
      <map:call function="DeleteComposer::delete"/>
   </map:match>

   <map:match pattern="deleteComposer.kont">
      <map:call function="DeleteComposer::delete"/>
   </map:match>

</map:pipeline>

<!-- Only called from the add user script DeleteComposer::delete -->

<map:pipeline internal-only="true">

   <map:match pattern="deleteComposer-*.html">
      <map:aggregate element="root" label="aggr-content">
         <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
         <map:part src="cocoon:/deleteComposer-{1}.px" element="content" strip-root="true"/>
      </map:aggregate>
      <map:transform type="mysql" label="sql-content">
         <map:parameter name="show-nr-of-rows" value="true"/>
      </map:transform>
      <map:transform src="context://resources/transforms/sql2pform.xsl"/>
      <map:transform src="resource://resources/transforms/pforms-violations.xsl" label="pforms-violations">
         <map:parameter name="formViolations" value="{session:__violations}"/>
```

```
      </map:transform>
      <map:transform src="resource://resources/transforms/pforms-default.xsl" label="pforms-default"/>
      <map:transform src="resource://resources/transforms/pforms2html.xsl" label="pforms-html"/>
      <map:call resource="outputPage"/>
   </map:match>

</map:pipeline>
```

The returned data is processed by the transform *sql2pform.xsl* whose transforms are:

41 Processing Returned Data

resources/transforms/sql2pform.xsl

```
<xsl:template match="sql:row-set[ @name = 'selectDeleteComposer' ]">
   <xsl:element name="pf:select1">
      <xsl:attribute name="appearance">full</xsl:attribute>
      <xsl:attribute name="ref">
         <xsl:value-of select="@name"/>
      </xsl:attribute>
      <xsl:element name="pf:violations"/>
      <xsl:element name="pf:choices">
         <xsl:for-each select="sql:row">
            <xsl:element name="pf:item">
               <xsl:element name="pf:label">
                  <xsl:value-of select="sql:name"/>
                  <xsl:text>, </xsl:text>
                  <xsl:value-of select="sql:forenames"/>
               </xsl:element>
               <xsl:element name="pf:value">
                  <xsl:value-of select="sql:name"/>
               </xsl:element>
            </xsl:element>
         </xsl:for-each>
      </xsl:element>
   </xsl:element>
</xsl:template>

<!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

<xsl:template match="sql:row-set[ @name = 'confirmDeleteComposer' ]">
   <xsl:element name="pf:output">
      <xsl:attribute name="appearance">full</xsl:attribute>
      <xsl:attribute name="ref">
         <xsl:value-of select="@name"/>
      </xsl:attribute>
      <xsl:element name="pf:label">Composer name:</xsl:element>
      <xsl:element name="pf:value">
         <xsl:value-of select="sql:row/sql:name"/>
         <xsl:text>, </xsl:text>
         <xsl:value-of select="sql:row/sql:forenames"/>
      </xsl:element>
   </xsl:element>
</xsl:template>

<!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

<xsl:template match="sql:row-set[ @name = 'getComposerId' ]">
   <xsl:element name="pf:hidden">
      <xsl:attribute name="ref">composerId</xsl:attribute>
      <xsl:element name="pf:value">
         <xsl:value-of select="sql:row/sql:id"/>
      </xsl:element>
   </xsl:element>
</xsl:template>

<!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

<xsl:template match="node()|@*" priority="-1">
   <xsl:copy>
```

```
      <xsl:apply-templates select="@*"/>
      <xsl:apply-templates/>
   </xsl:copy>
</xsl:template>
```

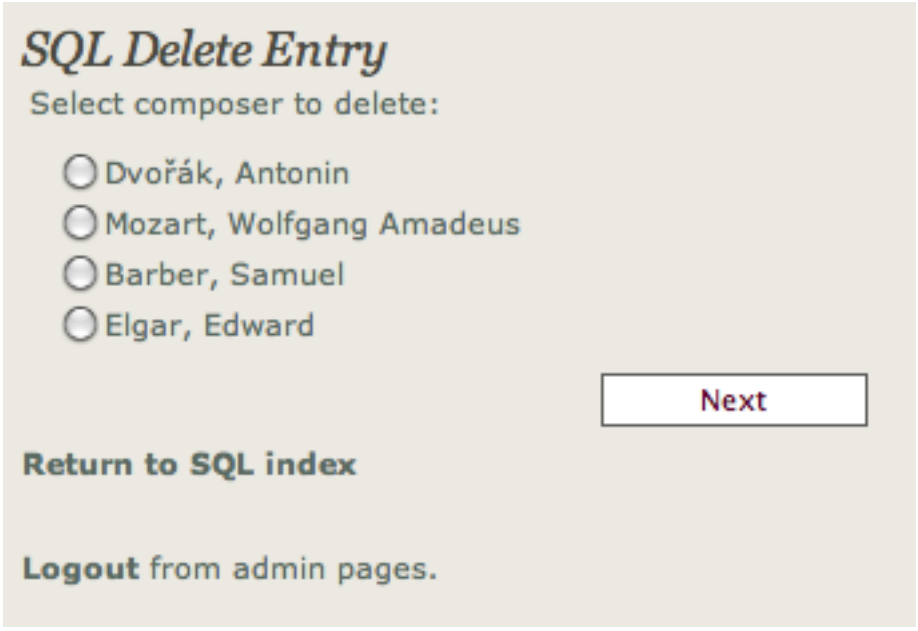
The purpose of this transform is to turn:

```
<default:row-set xmlns="http://apache.org/cocoon/SQL/2.0" nrofrows="4" name="selectDeleteComposer">
   <default:row>
      <default:id>16</default:id>
      <default:name>Dvok</default:name>
      <default:forenames>Antonin</default:forenames>
      <default:birth>1841-09-08</default:birth>
      <default:death>1904-05-01</default:death>
   </default:row>
   <default:row>
      <default:id>15</default:id>
      <default:name>Mozart</default:name>
      <default:forenames>Wolfgang Amadeus</default:forenames>
      <default:birth>1756-01-27</default:birth>
      <default:death>1791-12-05</default:death>
   </default:row>
   <default:row>
      <default:id>13</default:id>
      <default:name>Barber</default:name>
      <default:forenames>Samuel</default:forenames>
      <default:birth>1910-03-09</default:birth>
      <default:death>1981-01-23</default:death>
   </default:row>
   <default:row>
      <default:id>17</default:id>
      <default:name>Elgar</default:name>
      <default:forenames>Edward</default:forenames>
      <default:birth>1857-06-02</default:birth>
      <default:death>1934-02-23</default:death>
   </default:row>
</default:row-set>
```

into

```
<pf:select1 appearance="full" ref="selectDeleteComposer">
   <pf:violations/>
   <pf:choices>
      <pf:item>
         <pf:label>Dvok, Antonin</pf:label>
         <pf:value>Dvok</pf:value>
      </pf:item>
      <pf:item>
         <pf:label>Mozart, Wolfgang Amadeus</pf:label>
         <pf:value>Mozart</pf:value>
      </pf:item>
      <pf:item>
         <pf:label>Barber, Samuel</pf:label>
         <pf:value>Barber</pf:value>
      </pf:item>
      <pf:item>
         <pf:label>Elgar, Edward</pf:label>
         <pf:value>Elgar</pf:value>
      </pf:item>
   </pf:choices>
</pf:select1>
```

which is the appropriate PForm data to produce a select field:


SQL Delete Entry
Select composer to delete:
Dvořák, Antonin
Mozart, Wolfgang Amadeus
Barber, Samuel
Elgar, Edward
Next
Return to SQL index
Logout from admin pages.

41 Confirm Data Form

We need to confirm that the selected composer is the right one, so the PForm to do this is:

examples/sql/deleteComposer-2.xml

```
<page:content>

   <pf:form id="deleteComposerForm" flow="deleteComposer" continuation="{flow:continuation.id}"
      session="{flow:__flowId}">
      <pf:label>Press confirm to delete composer:</pf:label>

      <execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
         <query name="confirmDeleteComposer" database="composers">select * from composer
            where name = '{flow:selectDeleteComposer}'</query>
      </execute-query>

      <execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
         <query name="getComposerId" database="composers">select id from composer
            where name = '{flow:selectDeleteComposer}'</query>
      </execute-query>

      <pf:submit class="button" id="prev">
         <pf:label>Back</pf:label>
         <pf:hint>Go to previous page</pf:hint>
      </pf:submit>

      <pf:submit class="button" id="next">
         <pf:label>Confirm</pf:label>
         <pf:hint>Confirm delete composer</pf:hint>
      </pf:submit>
   </pf:form>

   <t:p>
      <link:link type="uri" ref="examples/sql/sql.html">Return to SQL index</link:link>
   </t:p>
   <t:p><link:link type="uri" ref="examples/sql/logout.html">Logout</link:link> from admin pages.</t:p>

</page:content>
```

There are two SQL queries here: the composers name for confirmation, and the identity to use for the actual delete command. The former will be used to display a simple text filed and the latter will be a hidden field.

41 Extending the Flow Script

The flow script needs to be changed to handle this case and detect errors (nothing selected):

resources/scripts/DeleteComposer.php

```
function delete()
{
     global $gModules;

    $this->gLogger->debug( "Delete composer, continuation: " . $this->gContinuation );
   $this->gViolations = array();

   $finished = false;
   while ( !$finished ) {
       $this->gLogger->debug( "Processing: " . $this->gContinuation );
     $errors = false;
      switch ( $this->gContinuation ) {

          case 0 :
             $this->sendPage(
                "deleteComposer-1.html",      // The page to send to the client
                $this->gDeleteComposerModel,  // The current data model to send
                ++$this->gContinuation,       // The continuation link for the next stage
                $this->gViolations );         // Array of errors (keyed by data model keys)
             $finished = true;
             break;
```

```
          case 1 :
             // Error processing here. Only have to check whether a composer has been selected.
            if ( strlen( $this->gDeleteComposerModel[ 'selectDeleteComposer' ] ) == 0  ) {
                $this->gViolations[ 'selectDeleteComposer' ] = "Need a composer to delete";
                 $errors = true;
             }

             if ( $errors ) {
                $this->gContinuation--;
                 $finished = false;
             } else {
                $this->sendPage(
                   "deleteComposer-2.html",
                   $this->gDeleteComposerModel,
                   ++$this->gContinuation,
                   $this->gViolations );
                $finished = true;
              }
              break;

          case 2 :
             ...

        } // switch
     } // while
  } // function delete()
```

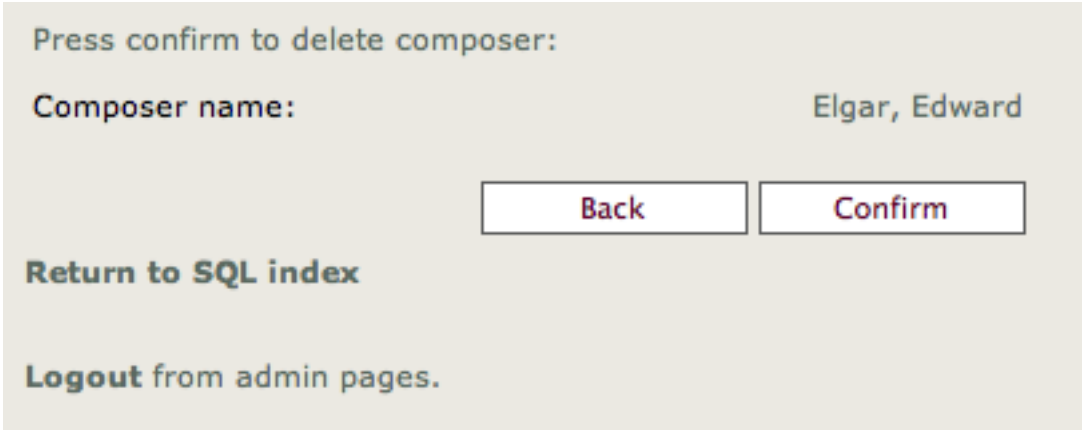
The same sitemap matcher is used as before. After selecting the composer and clicking "Next" the extracted data returned from the database:

```
<default:row-set xmlns="http://apache.org/cocoon/SQL/2.0" nrofrows="1" name="confirmDeleteComposer">
   <default:row>
      <default:id>17</default:id>
      <default:name>Elgar</default:name>
      <default:forenames>Edward</default:forenames>
      <default:birth>1857-06-02</default:birth>
      <default:death>1934-02-23</default:death>
   </default:row>
</default:row-set>

<default:row-set xmlns="http://apache.org/cocoon/SQL/2.0" nrofrows="1" name="getComposerId">
   <default:row>
      <default:id>17</default:id>
   </default:row>
</default:row-set>
```

This is transformed into

```
<pf:output appearance="full" ref="confirmDeleteComposer">
   <pf:label>Composer name:</pf:label>
   <pf:value>Elgar, Edward</pf:value>
</pf:output>

<pf:hidden ref="composerId">
   <pf:value>17</pf:value>
   </pf:hidden>
```

When processed this form appears as:


Press confirm to delete composer:
Composer name:
Elgar, Edward
Back
Confirm
Return to SQL index
Logout from admin pages.

41 Submitting SQL Delete command

Now that the right composer has been selected for deletion we have to construct a suitable SQL command and submit it. The flow variable *composerId* was taken from the hidden form field.

examples/sql/deleteComposer-3.xml

```
<page:content>
   <t:heading level="1">SQL Delete Composer</t:heading>

      <execute-query xmlns="http://apache.org/cocoon/SQL/2.0">
         <query name="runDeleteComposer"
            database="composers">DELETE FROM composer WHERE id='{flow:composerId}'</query>
      </execute-query>

   <t:p><link:link type="uri" ref="examples/sql/sql.html">Return to SQL index</link:link></t:p>
   <t:p><link:link type="uri" ref="examples/sql/logout.html">Logout</link:link> from admin pages.</t:p>
</page:content>
```

41 Extending the Flow Script

The flow script is modified:

resources/scripts/DeleteComposer.php

```
function delete()
{
     global $gModules;

    $this->gLogger->debug( "Delete composer, continuation: " . $this->gContinuation );
   $this->gViolations = array();

   $finished = false;
   while ( !$finished ) {
       $this->gLogger->debug( "Processing: " . $this->gContinuation );
     $errors = false;
      switch ( $this->gContinuation ) {

          case 0 :
             $this->sendPage(
                "deleteComposer-1.html",      // The page to send to the client
                $this->gDeleteComposerModel,  // The current data model to send
                ++$this->gContinuation,       // The continuation link for the next stage
                $this->gViolations );         // Array of errors (keyed by data model keys)
             $finished = true;
             break;

          case 1 :
             // Error processing here. Only have to check whether a composer has been selected.
            $this->gLogger->debug( "Checking: '" . $this->gDeleteComposerModel[ 'selectDeleteComposer' ] . "'" );
            if ( strlen( $this->gDeleteComposerModel[ 'selectDeleteComposer' ] ) == 0  ) {
                $this->gViolations[ 'selectDeleteComposer' ] = "Need a composer to delete";
                 $errors = true;
             }

             if ( $errors ) {
                $this->gContinuation--;
                 $finished = false;
             } else {
                $this->sendPage(
                   "deleteComposer-2.html",
                   $this->gDeleteComposerModel,
                   ++$this->gContinuation,
                   $this->gViolations );
                $finished = true;
              }
              break;

          case 2 :
             $this->sendPage(
                "deleteComposer-3.html",
                $this->gDeleteComposerModel,
```

```
                ++$this->gContinuation,
                $this->gViolations );
             $finished = true;
            break;

        } // switch
    } // while
} // function delete()
```

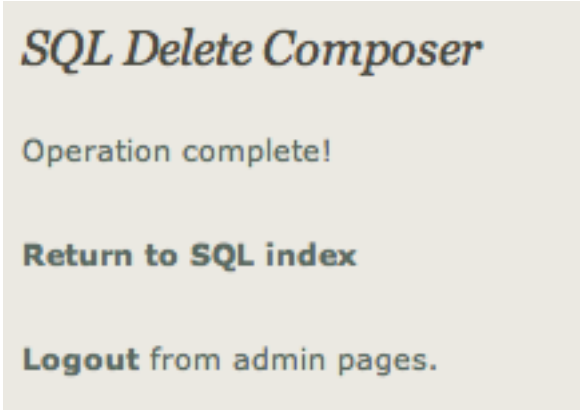
41 Extending the Sitemap

And the sitemap modified:

examples/sql/sitemap

```
<map:pipeline internal-only="true">

  <map:match pattern="deleteComposer-3.html">
      <map:aggregate element="root" label="aggr-content">
         <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
         <map:part src="cocoon:/deleteComposer-3.px" element="content" strip-root="true"/>
      </map:aggregate>
      <map:transform type="mysql" label="sql-content"/>
      <map:transform src="context://resources/transforms/sql2xml.xsl" label="sql-transform"/>
      <map:call resource="outputPage"/>
   </map:match>

   <map:match pattern="deleteComposer-*.html">
      <map:act type="auth-protect">
         <map:parameter name="handler" value="adminHandler"/>
         <map:aggregate element="root" label="aggr-content">
            <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
            <map:part src="cocoon:/deleteComposer-{1}.px" element="content" strip-root="true"/>
         </map:aggregate>
         <map:transform type="mysql" label="sql-content">
            <map:parameter name="show-nr-of-rows" value="true"/>
         </map:transform>
         <map:transform src="context://resources/transforms/sql2pform.xsl"/>
         <map:transform src="resource://resources/transforms/pforms-violations.xsl"
                  label="pforms-violations">
            <map:parameter name="formViolations" value="{session:__violations}"/>
         </map:transform>
         <map:transform src="resource://resources/transforms/pforms-default.xsl" label="pforms-default"/>
         <map:transform src="resource://resources/transforms/pforms2html.xsl" label="pforms-html"/>
         <map:call resource="outputPage"/>
      </map:act>
   </map:match>

</map:pipeline>
```

The matcher is simpler as we do not have to process any PForms. The result of a successful deletion is:



Having got the basic pages in we now need to restrict access to those page that change the database: adding and deleting. First of all we need a suitable login and logout page:

examples/sql/login.xml

```
<page:content>

    <t:heading level="1">Paloose Login</t:heading>

    <t:p>Please enter your login details for the SQL Example:</t:p>

    <form:form>
       <form:start url="checkLogin.html">Login</form:start>
       <form:field name="username" type="text">User name</form:field>
       <form:field name="password" type="password">Password</form:field>
    </form:form>

 </page:content>
```

examples/sql/logout.xml

```
<page:content>

    <t:heading level="1">Paloose SQL Example</t:heading>

    <t:p>You have now been logged out.</t:p>

     <t:p><link:link type="uri" ref="examples/sql/sql.html">Return to SQL Example index</link:link></t:p>
     </page:content>
```

We also need a suitable error notification page:

examples/sql/loginError.xml

```
<page:content>

    <t:heading level="1">Paloose SQL Example</t:heading>

    <t:p>Sorry that user/password combination does not seem to work, please try again:</t:p>

    <form:form>
       <!-- Will become the form's action attribute -->
       <form:start url="checkLogin.html">Login</form:start>
       <form:field name="username" type="text">User name</form:field>
       <form:field name="password" type="password">Password</form:field>
    </form:form>

</page:content>
```

We now have to add in the various authorisation bits into the siteamp. First declare the components:

pp/examples/sql/sitemap.xmap

```
<map:components>
    <map:generators default="file"/>
    <map:transformers default="xslt">
       <map:transformer name="mysql" src="resource://lib/transforming/SQLTransformer">
          <map:parameter name="type" value="mysql"/>
          <map:parameter name="host" value="localhost:3306"/>
          <map:parameter name="user" value="hsfr"/>
          <map:parameter name="password" value="fof-hyd"/>
          <!-- Only time I use this password so don't get excited! :-) -->
       </map:transformer>
       <map:transformer name="password" src="resource://lib/transforming/PasswordTransformer"/>
    </map:transformers>
   <map:actions>
       <map:action name="auth-protect" src="resource://lib/acting/AuthAction"/>
       <map:action name="auth-login" src="resource://lib/acting/LoginAction"/>
       <map:action name="auth-logout" src="resource://lib/acting/LogoutAction"/>
    </map:actions>
    <map:serializers default="xml"/>
    <map:matchers default="wildcard"/>
</map:components>
```

Then the authorisation manager:

pp/examples/sql/sitemap.xmap

```
<map:pipelines>

   <map:component-configurations>
      <map:authentication-manager>
         <!-- This is the means that Paloose checks whether a user is authenticated
            and is allowed to see pages protected within the pipeline. If the handler
            does not authenticate the use then go to redirect -->
         <map:handlers>
            <map:handler name="adminHandler">
               <!-- Run this if the user needs login (diverts to this sitemap) -->
               <map:redirect-to uri="cocoon:/login"/>
               <!-- The pipeline used to authenticate the user -->
               <map:authentication uri="cocoon:/authenticate-user.html"/>
            </map:handler>
         </map:handlers>
      </map:authentication-manager>
   </map:component-configurations>

   <!-- pipelines -->

</map:pipelines>
```

Finally the internal pipeline that processes the above:

pp/examples/sql/sitemap.xmap

```
<map:pipeline internal-only="true">

   <map:match pattern="authenticate-user.html">
      <map:generate src="context://configs/adminUsers.xml" label="xml-content"/>
      <map:transform src="context://resources/transforms/admin-getLoginQuery.xsl">
         <map:parameter name="username" value="{request-param:username}"/>
         <map:parameter name="password" value="{request-param:password}"/>
      </map:transform>
      <!-- Encrypt the password -->
      <map:transform type="password"/>
      <map:transform src="context://resources/transforms/admin-buildAuthenticateDOM.xsl"/>
      <map:serialize type="xml"/>
      <!-- The output here is a standard Cocoon authenticate structure
         and is returned to the authentication-manager -->
   </map:match>

   <!-- Login form action -->

   <map:match pattern="login">
      <!-- We come here when we need the user to log in. It produces a login
         form for the user to fill in. -->
      <map:aggregate element="root" label="aggr-content">
         <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
         <map:part src="cocoon:/login.xml" element="content" strip-root="true"/>
      </map:aggregate>
      <map:call resource="outputPage"/>
   </map:match>

   <map:match pattern="checkLogin.html">
      <!-- Match pattern must be the same match as that on the login form URL -->
      <map:act type="auth-login">
         <map:parameter name="handler" value="adminHandler"/>
         <map:parameter name="username" value="{request-param:username}"/>
         <map:parameter name="password" value="{request-param:password}"/>
         <!-- Logged in so must have given password and username -->
         <map:redirect-to uri="cocoon:/sql.html"/>
      </map:act>
      <!-- Oops, not logged in properly so give appropriate screen back -->
      <map:aggregate element="root" label="aggr-content">
         <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
         <map:part src="cocoon:/loginError.xml" element="content" strip-root="true"/>
      </map:aggregate>
      <map:call resource="outputPage"/>
   </map:match>
```

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

The format of the list of admin users is:

context://configs/adminUsers.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<authentication>
   <users>
      <user>
         <username>hsfr</username>
         <password>672b86ff.............33a92040c5</password>
         <data>
            <fullname>Hugh Field-Richards</fullname>
            <email>hsfr@hsfr.org.uk</email>
         </data>
      </user>
    </users>
    </authentication>
```

This is processed by the transform `admin-getLoginQuery.xsl`:

context://resources/transforms/admin-getLoginQuery.xsl

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

   <xsl:param name="password"/>
   <xsl:param name="username"/>

   <xsl:template match="//authentication">
      <xsl:element name="authentication" >
         <xsl:attribute name="username"><xsl:value-of select="$username" /></xsl:attribute>
         <xsl:attribute name="password"><xsl:value-of select="$password" /></xsl:attribute>
         <xsl:apply-templates mode="pass"/>
      </xsl:element>
   </xsl:template>

   <xsl:template match="@*|node()" mode="pass">
      <xsl:copy>
         <xsl:apply-templates select="@*|node()" mode="pass" />
      </xsl:copy>
   </xsl:template>

</xsl:stylesheet>
```

This is then run through the password transformer and then the build authenticate DOM transform:

context://resources/transforms/admin-buildAuthenticateDOM.xsl

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

   <xsl:variable name="gPassword" select="//authentication/@password" /> <!-- encrypted -->
   <xsl:variable name="gUsername" select="//authentication/@username" />

<!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

   <xsl:template match="authentication">
      <authentication>
         <xsl:apply-templates select="users"/>
      </authentication>
   </xsl:template>

<!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

   <xsl:template match="users">
      <xsl:apply-templates select="user"/>
   </xsl:template>

<!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
```

```
   <xsl:template match="user">
      <xsl:if test="normalize-space( username ) = $gUsername and normalize-space( password ) = $gPassword">
         <ID><xsl:value-of select="username"/></ID>
         <data>
            <ID><xsl:value-of select="username"/></ID>
            <username><xsl:value-of select="username"/></username>
            <password><xsl:value-of select="password"/></password>
            <fullname><xsl:value-of select="data/fullname"/></fullname>
            <xsl:if test="data/telephone">
               <telephone><xsl:value-of select="data/telephone"/></telephone>
            </xsl:if>
            <xsl:if test="data/email">
               <email><xsl:value-of select="data/email"/></email>
            </xsl:if>
         </data>
      </xsl:if>
   </xsl:template>

</xsl:stylesheet>
```

Which outputs the necessary data for the authentication manager. Individual pages are protected with the following additions:

examples/sql/sitemap.xmap

```
<map:pipeline>

   <map:match pattern="addComposer.html">
      <map:act type="auth-protect">
         <map:parameter name="handler" value="adminHandler"/>
         <map:call function="AddComposer::add"/>
      </map:act>
   </map:match>

   <map:match pattern="addComposer.kont">
      <map:call function="AddComposer::add"/>
   </map:match>

   </map:pipeline>
```

Only externally accessable pages ned be protected in the sitemap, all "*internal-only*" pipelines are protected naturally.

More information on authorising pages can be found in the documentation on authorising

42 Using Paloose as A Client

It is possible to use Paloose as a client to such Web applications/servers as WordPress and Joomla!. The process requires making a suitable interface to the server and instructing Paloose to act as a client.

42 Why use Paloose as a plugin to a CMS ?

A very important question. In general a simple CMS based web site will probably not need the functionality that Paloose adds. However with the prevalence and adoption of XML, a means of manipulating XML encoded data is very useful. For example a site might want to show data imported from another application such as a genealogy program that produces GEDCOM (either as XML or the GEDCOM format which Paloose understands). This is not held as SQL data and thus needs processing as pure XML. The data can be manipulated to select subsets of the data and then displayed as XHTML included in an existing CMS page. It would be possible to manipulate this data using a PHP-based XML transformation. However, Paloose allows you to write using XSL directly, a much richer and easier way of doing the transformations.

Note that this Paloose site uses Paloose as a server with Apache rather than using a CMS; but there would be no reason why it should not use a CMS such as Joomla!. Having said that, there is no imported data from external programs needing XML transfomations so it would be possible to use a CMS system alone. One important factor, however, is that the article held as XML can be transformed into other forms. One form is transforming the XML into LaTeX to get a printed version of this site as a book. This can be downloaded as a PDF version of the Paloose documentation.

42 Which systems are supported ?

Currently Paloose has plugins for the following apps:

- WordPress
- Joomla! CMS

43 Using Paloose with WordPress

Paloose can be used alongside WordPress to enhance the capability of both. WordPress is a popular application for the Web to produce blogs and general websites. It provides considerable hooks to extend it, both thematically and functionally. Paloose provides a means of adding to WordPress complex XML processing based on the Cocoon approach.

Until the plugin is loaded onto the WordPress site (which will happen when it has been fully tested) the following instructions are for those who want to try the plugin and are happy to load the latest version here and install it manually.

Please use at your own risk as this is a very beta version. More documentation is in preparation

There are several approaches to running the Paloose Plugin. These vary from the "get me running as quickly as possible", through the "I have already got a Paloose Server installed" to the "I need speed from the Paloose system". For an example of using this plugin see the example here.

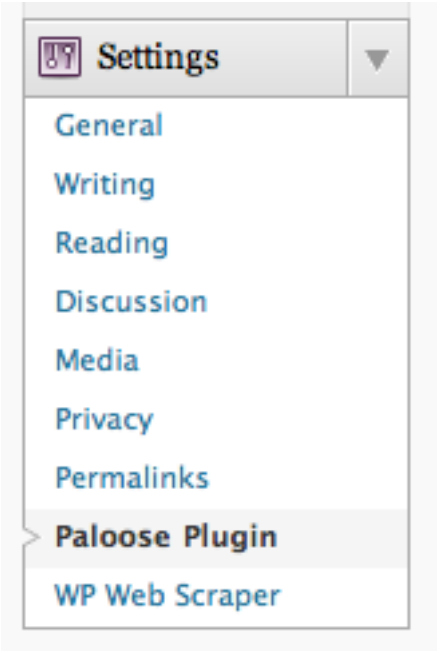
43 Install with no existing Paloose system

Download the latest version here. The download includes the latest Paloose and its compact version together with the latest Log4PHP that Paloose will work with. The directory structure of the plugin is:

```
paloose ----- log4php <-- Log4PHP subsystem
          |
          |-- paloose-system <-- Full Paloose system
          |
          |-- paloose-compact-system <-- Compacted Paloose system
          |
          |-- paloose-admin ----- options.php
          |
          |-- paloose-includes ----- functions.php
          |
          |-- paloose-scripts ----- palooseAdmin.js
          |
          |-- paloose-styles ----- palooseAdmin.css
          |
          |-- paloose.php <-- paloose plugin hook
          |
          |-- readme.txt <-- documentation for plugin
```

This complete directory should be put in the WordPress plugins directory (`wordpress/wp-content/plugins`). I intend to include in the WordPress plugin site as soon as I am happy it can be released. It will then be available on the WordPress dashboard:

(The version number and other details may be slightly different). When the plugin is then activated it should appear in the setting menu of the Dashboard:



Selecting the plugin will show a set of options to configure your site's local conditions. In general you will not have to change these for a simple installation. In a system that includes the paloose system (full, compact and logging) the typical values for the configuration variables are:

- **Directory where the paloose subsystem is stored** — "../paloose-system" (change this to "../paloose-compact-system" to use the more efficient but no logging version of Paloose).
- **Directory where the XML documents are held that Paloose uses** — "[base directory path of your XML repository]".
- **File (and path) where the Logger configuration information is stored** — "[File name and path of your log4php lagging configuration]".
- **Directory path where the log4php code is stored** — "../log4php".

All other parameters can be safely ignored or adjusted to suit local conditions: cache directory, root sitemap. These should remain the same regardless of whether you have an existing Paloose server fitted.

43 Install with existing Paloose server

There are occasions where you may need to have a Paloose server as well as a WordPress system working at the same time. In this case the server installation of Paloose can be used instead of the version in the plugin directory. Although you do not have to, you can removing the Paloose system in the plugin directory (`paloose-system`) and its compact version (`paloose-compact-system`). The 4 variables defined above must be changed to reflect the path of your existing Paloose installation.

43 Using the Paloose WordPress plugin

The plugin is called from a template file using the following syntax:

```
<?php echo paloose( [uri path string], [query string] ); ?>
```

where

- *[uri path string]* — the uri path that will be used to match a particular pipeline in the Paloose sitemap.

- *[query string]* — the associated query string with the URI.

For example if the template has the following line

```
<?php echo paloose( 'concerts.html', $_SERVER[ 'QUERY_STRING' ] ); ?>
```

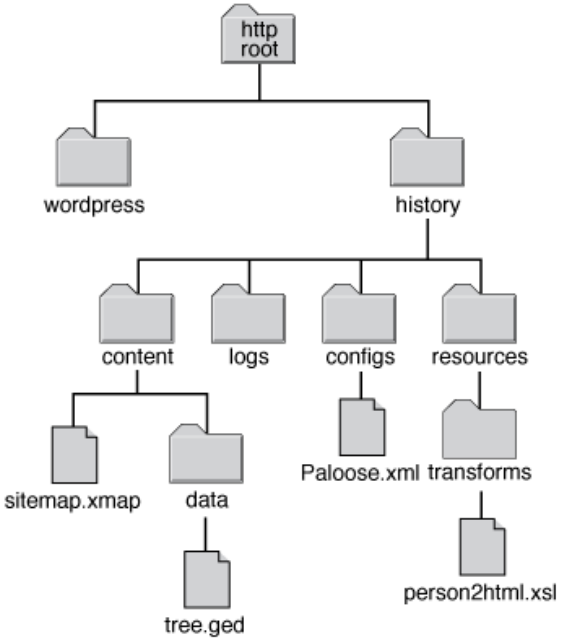
and if the calling URI is *http://¡localhost¿/concerts.html?year=2010*, the following pipeline would match and the request parameter *concertYear* would be set to "year=2010".

```
<map:match pattern="concerts.html">
   <map:generate src="context://content/data/concerts.xml" label="concerts-content"/>
   <map:transform src="context://resources/transforms/extractConcerts.xsl">
      <map:parameter name="who" value="{request-param:concertYear}"/>
   </map:transform>
   <map:serialize type="xml"/>
</map:match>
```

The original calling line would be replaced by the XML (XHTML) returned from Paloose. Note that the serializer must be XML *not* XHTML as Paloose returns a XHTML scrap, not a full HTML document.

44 WordPress Example

Consider the case where a family history site is being constructed based on WordPress. The family details are stored in a GEDCOM file and we would like to incorporate them into our site. Assume that a our site has the following structure:



There is nothing special about the directory structure, it is the one that I use on the Paloose servers and serves to isolate the various parts of the site. The relevant part of the GEDCOM (tree.ged) is (Frederick was a real person, my great-grandfather):

tree.ged

```
0 @I12@ INDI
1 NAME Frederick/FIELD-RICHARDS/
2 GIVN Frederick
2 SURN FIELD-RICHARDS
1 TITL Rev
1 SEX M
1 OCCU Clergyman
1 BIRT
2 DATE 2 NOV 1846
2 PLAC Hackney,,,,,
1 BAPM
2 DATE 3 DEC 1865
1 CONF
2 DATE 8 DEC 1865
1 ORDN
2 DATE 21 DEC 1872
2 NOTE Deacon
1 ORDN
2 DATE 11 JUN 1876
2 NOTE Priest
1 DEAT
2 DATE 10 APR 1879
1 BURI
2 DATE 17 APR 1879
2 PLAC Hastings Borough Cemetery,,,,,T34 - Division D Section E.
1 FAMS @F78@
1 FAMC @F81@
1 REFN FrederickField-Richards
1 OBJE
2 FILE :images:FrederickField-Richards.jpg
```

```
3 FORM jpg
1 OBJE
2 FILE :images:Field-Richards-Crest.tif
3 FORM jpg
```

The GEDCOM generator in the sitemap translates this to a simple XML representation that is injected into the pipeline and can be processed by the XSL. The above scrap is transformed into the XML:

```
<g:indi id="@I12@">
   <g:name data="Frederick/FIELD-RICHARDS/"/>
   <g:titl data="Rev"/>
   <g:sex data="M"/>
   <g:occu data="Clergyman"/>
   <g:birt>
      <g:date data="2 NOV 1846"/>
      <g:plac data="Hackney,,,,,"/>
   </g:birt>
   <g:bapm>
      <g:date data="3 DEC 1865"/>
   </g:bapm>
   <g:conf>
      <g:date data="8 DEC 1865"/>
   </g:conf>
   <g:ordn>
      <g:date data="21 DEC 1872"/>
      <g:note data="Deacon"> </g:note>
   </g:ordn>
   <g:ordn>
      <g:date data="11 JUN 1876"/>
      <g:note data="Priest"> </g:note>
   </g:ordn>
   <g:deat>
      <g:date data="10 APR 1879"/>
   </g:deat>
   <g:buri>
      <g:date data="17 APR 1879"/>
      <g:plac data="Hastings Borough Cemetery,,,,,T34 - Division D Section E."/>
   </g:buri>
   <g:fams ref="@F78@"/>
   <g:famc ref="@F81@"/>
   <g:refn data="FrederickField-Richards"/>
   <g:obje>
      <g:file data=":images:FrederickField-Richards.jpg">
         <g:form data="jpg"/>
      </g:file>
   </g:obje>
   <g:obje>
      <g:file data=":images:Field-Richards-Crest.tif">
         <g:form data="jpg"/>
      </g:file>
   </g:obje>
</g:indi>
```

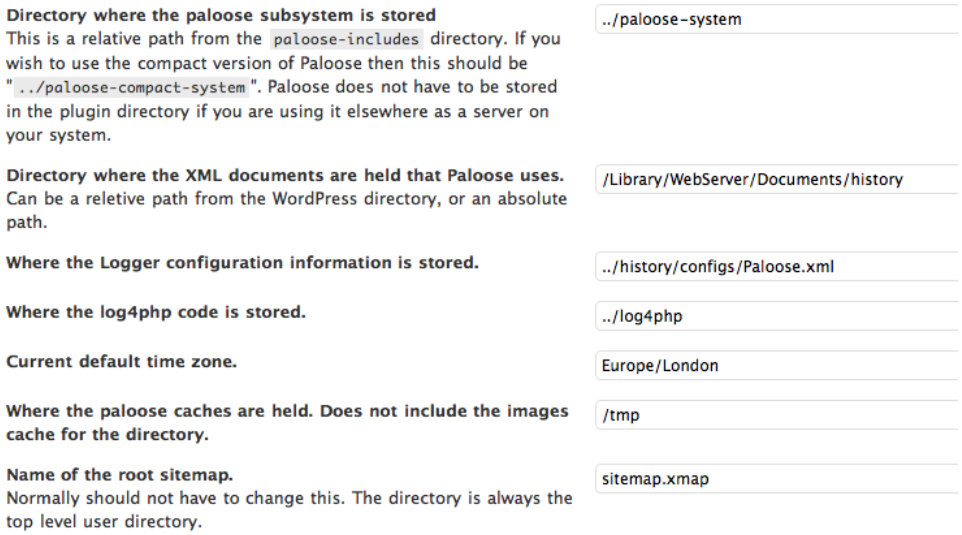
The requirement of the WordPress history site is to have a page for each member of the family with excerpts from the family GEDCOM data at the topm of the page. First we need a page template in a WordPress theme for the site. Assume that this page exists and the relevant links to the page exist via the Dashboard. The relevant temmplate file is

wp-content/themes/history/people.php

```
<?php
/*
Template Name: People
*/
?>

<?php get_header(); ?>
```

```
<div id="content" class="widecolumn">


<div id="main">

<?php echo paloose( 'people.html', $_SERVER[ 'QUERY_STRING' ] ); ?>

</div>
<?php get_footer(); ?>
```

The URL to access this is `http://[host name]/wordpress/history/people/?who=FrederickField-Richards`. First of all the Paloose plugin settings for the site must be entered to tell Paloose where the relevant files are stored:



The full Paloose (rather than the compact version) is being used. The data that Paloose uses in `history` is defined as an absolute path. Since we are using the full version of Paloose we must make sure that the Logging system is set up. The configuration file would typically look like:

configs/Paloose.xml

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<configuration xmlns="http://logging.apache.org/log4php/" threshold="all" INFO="false">

   <appender name="paloose" class="LoggerAppenderDailyFile">
      <param name="datePattern" value="Ymd" />
      <!-- Change this to suit local conditions, but make sure the right permissions
         are set for the logging folder (777) -->
      <param name="file" value="../history/logs/paloose_%s.log" />
      <layout class="LoggerLayoutTTCC">
         <param name="threadPrinting" value="true" />
         <param name="categoryPrefixing" value="true" />
         <param name="contextPrinting" value="true" />
         <param name="microSecondsPrinting" value="false" />
      </layout>
   </appender>

   <root>
      <level value="INFO" />
      <appender_ref ref="paloose" />
   </root>

   <logger name="Sitemap">
      <level value="INFO"/>
   </logger>

</configuration>
```

Obviously local conditions might change this. The log4php subsystem is stored at the same level as the Paloose system. However it does not have to be and both the latter and log4php can be stored in a more central location if required. If the compact version is used the logging system can be safely ignored.

The root sitemap is relatively simple, throwing all request to the `content directory`:

history/sitemap.xmap

```
<?xml version="1.0" encoding="UTF-8"?>

<?oxygen RNGSchema="/Library/Schemas/rng/sitemap/paloose-sitemap-1.3.4.rng" type="xml"?>
<?oxygen SCHSchema="/Library/Schemas/rng/sitemap/paloose-sitemap-1.3.4.rng"?>

<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

   <map:components>

      <map:generators default="file">
         <map:generator name="file" src="resource://lib/generation/FileGenerator" cachable="no"/>
         <map:generator name="px" src="resource://lib/generation/PXTemplateGenerator" cachable="no">
            <map:use-request-parameters>true</map:use-request-parameters>
         </map:generator>
      </map:generators>

      <map:transformers default="xslt">
         <map:transformer name="xslt" src="resource://lib/transforming/TRAXTransformer" cachable="no">
            <map:use-request-parameters>true</map:use-request-parameters>
         </map:transformer>
         <map:transformer name="log" src="resource://lib/transforming/LogTransformer"/>
      </map:transformers>

      <map:serializers default="xml">
         <map:serializer
               name="xhtml" mime-type="text/html"
               src="resource://lib/serialization/XHTMLSerializer">
            <doctype-public>-//W3C//DTD XHTML 1.0 Transitional//EN</doctype-public>
            <doctype-system>http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd</doctype-system>
            <encoding>iso-8859-1</encoding>
         </map:serializer>
         <map:serializer
               name="xml" mime-type="text/xml"
               src="resource://lib/serialization/XMLSerializer"/>
      </map:serializers>

      <map:matchers default="wildcard">
         <map:matcher name="wildcard" src="resource://lib/matching/WildcardURIMatcher"/>
      </map:matchers>

   </map:components>

   <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- PIPELINES -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

   <map:pipelines>

      <map:pipeline>
         <map:match pattern="**.html">
            <map:mount src="cocoon://content/sitemap.xmap"/>
         </map:match>

         <map:handle-errors>
            <map:generate type="px" src="context://content/error.xml"/>
            <map:transform src="context://resources/transforms/buildMenus.xsl"/>
            <map:transform src="context://resources/transforms/page2xhtml.xsl">
               <map:parameter name="page" value="error"/>
            </map:transform>
            <map:transform src="context://resources/transforms/stripNamespaces.xsl"/>
            <map:serialize type="xml"/>
         </map:handle-errors>
      </map:pipeline>
```

```
   </map:pipelines>

</map:sitemap>
```

Errors in the pipelines are handled here and return a simply formatted error message. All html requests are sent to a subsitemap which is:

history/content/sitemap.xmap

```
<?xml version="1.0" encoding="UTF-8"?>

<?oxygen RNGSchema="../../../Schemas/rng/sitemap-v06.rng" type="xml"?>

<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

   <map:components>
      <map:generators default="file">
         <map:generator name="gedcom" src="resource://lib/generation/GedComGenerator" cachable="no"/>
      </map:generators>
      <map:transformers default="xslt"/>
      <map:serializers default="xml"/>
      <map:matchers default="wildcard"/>
   </map:components>

   <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*- PIPELINES -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

   <map:pipelines>

      <map:pipeline>

         <map:match pattern="people.html">
            <map:generate type="gedcom" src="context://content/data/tree.ged"/>
            <map:transform src="context://resources/transforms/person2xhtml.xsl">
               <map:parameter name="who" value="{request-param:who}"/>
            </map:transform>
            <map:serialize type="xml"/>
         </map:match>

      </map:pipeline>

   </map:pipelines>

</map:sitemap>
```

All very straightforward with the pipeline returning data based on the GEDCCOM data and the transform:

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

<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:g="http://gedcom.org/dtd/gedxml55.dtd"
   version="1.0">

   <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   <!--
      Add any parameters from the sitemap here.
   -->

   <xsl:param name="who"/>

   <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
   <!--
      Add your global variables here as required. The default values are set here as well.
   -->

   <xsl:variable name="gWho" select="$who"/>
```

```
    <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
    <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

    <xsl:template match="/">
       <xsl:variable name="individualId" select="//g:refn[@data = $gWho]/parent::*[1]/@id"/>
       <xsl:variable name="birthDate" select="//g:indi[@id = $individualId]/g:birt/g:date/@data"/>
       <xsl:variable name="deathDate" select="//g:indi[@id = $individualId]/g:deat/g:date/@data"/>
       <xsl:variable name="title" select="//g:indi[@id = $individualId]/g:titl/@data"/>
       <xsl:variable name="name" select="//g:indi[@id = $individualId]/g:name/@data"/>

       <xsl:element name="div">
          <xsl:attribute name="id">mainFrame</xsl:attribute>
          <xsl:element name="div">
             <xsl:attribute name="class">personTitleTextPanel</xsl:attribute>
             <xsl:element name="div">
                <xsl:attribute name="id">personName</xsl:attribute>
                <xsl:value-of select="$name"/>
                <xsl:if test="not( string-length( $title ) = 0 )">
                   <xsl:value-of select="concat( ' (', $title, ')' )"/>
                </xsl:if>
             </xsl:element>
             <xsl:element name="div">
                <xsl:attribute name="id">personDates</xsl:attribute>
                <xsl:call-template name="common.outputBirthDeathDates">
                   <xsl:with-param name="inBirthDate" select="$birthDate"/>
                   <xsl:with-param name="inDeathDate" select="$deathDate"/>
                </xsl:call-template>
             </xsl:element>
          </xsl:element>
       </xsl:element>
    </xsl:template>

    <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->
    <!-- -*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-*-* -->

    <xsl:template name="common.outputBirthDeathDates">
       <xsl:param name="inBirthDate"/>
       <xsl:param name="inDeathDate"/>
       <xsl:choose>
          <xsl:when test="string-length( $inBirthDate ) = 0 and string-length( $inDeathDate ) = 0">
             <xsl:text> </xsl:text>
          </xsl:when>
          <xsl:when test="string-length( $inBirthDate ) = 0 ">
             <xsl:value-of select="concat( '(d.', $inDeathDate, ')' )"/>
          </xsl:when>
          <xsl:when test="string-length( $inDeathDate ) = 0 ">
             <xsl:value-of select="concat( '(b.', $inBirthDate, ')' )"/>
          </xsl:when>
          <xsl:otherwise>
             <xsl:value-of select="concat( '(', $inBirthDate, ' -- ', $inDeathDate, ')' )"/>
          </xsl:otherwise>
       </xsl:choose>
    </xsl:template>

 </xsl:stylesheet>
```

Obviously this is a highly simplistic example but the basic principle is common to most cases.

45 Using Paloose with Joomla!

Paloose can be used alongside Joomla! to enhance the capability of both. Joomla! is a popular Content Management System. It provides considerable hooks to extend it, both thematically and functionally. Paloose provides a means of adding to Joomla! complex XML processing based on the Cocoon approach.

Until the plugin is loaded onto the Joomla! site (which will happen when it has been fully tested) the following instructions are for those who want to try the plugin and are happy to load the latest version here and install it manually.

Please use at your own risk as this is a very beta version. More documentation is in preparation

45 Using the Paloose Joomla plugin

The plugin is called from a template or article file using the following syntax:

```
{paloose link="[uri path string]" query="[query string]"}
```

where

- *[uri path string]* — the uri path that will be used to match a particular pipeline in the Paloose sitemap.
- *[query string]* — the associated query string with the URI.

For example if the template has the following line

```
{paloose link="concerts.html" query="year=2009"}
```

the following pipeline would match and the request parameter *concertYear* would be set to "year=2009".

```
<map:match pattern="concerts.html">
   <map:generate src="context://content/data/concerts.xml" label="concerts-content"/>
   <map:transform src="context://resources/transforms/extractConcerts.xsl">
      <map:parameter name="who" value="{request-param:concertYear}"/>
   </map:transform>
   <map:serialize type="xml"/>
</map:match>
```

The original calling line would be replaced by the XML (XHTML) returned from Paloose. Note that the serializer must be XML *not* XHTML as Paloose returns a XHTML scrap, not a full HTML document.

46 Frequently Asked Questions

46 About the author

The Paloose project is currently run by me, Hugh Field-Richards. I recently retired after a (too) long career designing (amongst other things) computer hardware and software; I am now a professional music engraver, something I did as a hobby during my working life. I started Paloose to keep myself amused and to support my other interest, Hop Vine Music.

46 What's with the name?

The original name I chose was "Papoose"; in the UK this is defined as "*... a type of bag used to carry a child on the back*". Unfortunately it does not seem have this meaning in the US, indeed, with some people, it is a derogatory term. However, it occurred to me that a simple change of letter would give "Paloose", which is another term for the Appaloosa horse.

This is something dear to my heart as, until recently, I was privileged to have one of my own ('Zitty') who I rode until she reached the grand old age of 29 and then became a "house pet" until she died aged 33. I now have a fine quarter horse named "Fred" (Harlan Breeze, born 2005) who is coming along nicely as a promising Western horse.

46 What future for Paloose?

Paloose is released under the GPL-3. If anyone wishes to get involved then please EMAIL me. I intend to use it for all my small sites and will continue to do so until Cocoon is supported by the smaller ISPs that I use. It is also interesting (and slightly depressing) that the latest Cocoon does not seem to be as easy to use as Cocoon Version 2.1.x. I use this latter version on my local servers and have no plans (yet) to upgrade to Cocoon Version 2.2 or later. While I am prepared to admit that this is my problem, I know others who have had the same experience. As a result perhaps Paloose will fulfil the requirements of those who want a quick Cocoon-like engine for their small Web sites.

46 What editor/IDE do you use to develop Paloose?

From the start of Paloose I have always used the oXygen XML editor and IDE. I used it as part of my work before I retired and started Paloose. I had been involved with SGML and XML for many years and oXygen had always been the editor I turned to. When I wanted a suitable system to develop Paloose the choice became a "no-brainer" and I obtained a suitable licence. Whenever I have XML/XSL to develop it is the one I turn to. Having it syntax aware of PHP, CSS and other languages is a real bonus.

46 What Licence is Paloose issued under?

You should have received a copy of the GNU General Public License along with Paloose when you downloaded it. If not, see http://www.gnu.org/licenses/

46 What facilities do you intend to add to Paloose?

This is quite a difficult question as I have limited time and am not trying to do a complete Cocoon implementation in PHP5 — a pointless exercise anyway. Now that I have added a simple (limited) data caching scheme to the pipeline all of what I intended to add is complete. I am gradually refining the code and improving the documentation and comments. I am happy to look at anything that users suggest — just EMAIL me at hsfr@hsfr.org.uk and may add a further wrapper for Content Management Systems such as Joomla!.

Just for fun I am making a CMS system based on Paloose. It is about 50% finished and I only work on it when I am not doing else so it may take some time.

46 Can I use the Paloose logo on my site?

Yes please. I suggest the following "Powered by Paloose" logo or the smaller version. Scale it as you feel fit.

46 Do you intend to add PDF support via FOP?

The short answer to this is: no. It would mean writing a complete XSL-fo system in PHP5 which is something I really do not have the time (or inclination) to do. If you really want to produce PDF output then I suggest that you write a suitable XML to LaTeX transformer and then run that to get PDF.

I wrote a transformer (XML to LaTeX) to produce a paper PDF copy of this Paloose documentation Web site which runs within an Ant build file. It is not particularly difficult to generate XML to LaTeX transformers, although obviously it requires a knowledge of LaTeX. I would be happy to help anyone who has a problem.

46 What systems will it run on?

Basically any system that will run PHP5. Currently the list is Mac OS-X, Linux, and other similar Unix systems. I also know of a system based on Apache 2.0/PHP5 on Windows Server 2003.

46 Why don't you use SAX, like Cocoon, instead of DOM?

I thought about this quite hard initially. I concluded that using a DOM approach was far simpler. It has the penalty of being slower but Paloose is only intended for small volume/traffic sites and ultimate performance is not such an issue. I have seen no reason to revise this decision in the light of typical site usage. Changing to a SAX approach would mean a major code change and I really do not have the time at present as my Music Publishing venture (Hop Vine Music) and my horses have more call on my time.

46 How can I improve the performance of Paloose?

Assuming that you are already using the caching system and want more speed, it is possible, but at the expense of a little effort (and possible obscurity), to tailor the Paloose code and your own sitemaps and XSL. The implications and method are described on a separate page of the documentation in more detail.

46 What about caching in Paloose?

I have done considerable work on various ways of caching within Paloose and versions after 1.3.0 use pipeline data caching on certain generators and transformers. I confess that I am in a quandry over this as the gains from using a cache seem to be minimal and not really worth the effort except in some key cases — see my caching discussion for more on this. It is not the end of the story, but more performance increases will probably come from elsewhere.

46 Why did you not use CForms and JavaScript for flows?

It is useful to provide some background to some of the decisions that I made for flows and forms. This is where Paloose diverges from Cocoon most. The latter is based on Java and Javascript which was not available to me (conceptually, not practically, as Paloose is based solely on PHP5). Thus I had to base what I did on a pure PHP approach while including the concepts of Cocoon flows and forms. I made several false starts which arose from several design problems:

- How to store the information at each point of the flow?
- Which form template to use (CForms, XForms etc.)?
- How to allow forward and back (continuations) in the forms?

All of these problems seemed intractable at first with decisions made about one influencing, detrimentally, a decision made elsewhere. Mirroring the Cocoon forms template, CForms was initially desirable for commonality with Cocoon. But it soon became apparent that this was going to make the whole approach far too complicated for what I wanted. Much code was wasted in exploring this but I believe it was the right decision. In the end I based the Paloose forms (PForms) on the JXForms that older version of Cocoon used. (I had produced some sites with this in the past so I was reasonably familiar with it). PForms is not radically different from JXForms but it does not slavishly follow the latter. I believe that PForms is suitable for the restrictions I had set on Paloose and, as I have said elsewhere, if you have a site that requires all the facilities of CForms then you should probably be using Cocoon anyway.

Once I had settled on PForms there was the problem of how to implement the flow script, which in Cocoon uses Javascript. The Cocoon approach is to take a "snapshot" of the Javascript engine (a simplistic way of describing it) in order to maintain continuity between client requests. I was unable to find a sensible way of doing this with PHP5 (which does not mean to say that one does not exist). So I had to find a means of providing the user with a simple method of continuations based solely on a PHP5 approach. I believe I have achieved this while keeping to the spirit of Cocoon.

All these solutions, however successful, have made me diverge from the strict Cocoon approach so please read the documentation very carefully.

46 Who designs your own sites that use Paloose?

These have been done by my son Ian Field-Richards who is the GUI art director for DeviantArt and I am very grateful to him for the help he has given me in their design. However I have to take the blame for the Paloose site: any good points are due to him, the bad ones are all mine.

46 How do I generate multiple selection lists in PForms from SQL entries?

The addition of a simple transformer is all that is necessary; it is explained in more detail on this How-to.

46 How do I port Paloose Sites to Cocoon?

This is very easy with the only major change being the sitemap components. But beware, the Password, Gallery and PageHit transformers will not work unless you write your own in Java.

46 How do I port Cocoon Sites to Paloose?

This is really just the reverse of the above. The only issue might be one of performance so it is worth bearing in mind that Paloose is obviously slower than Cocoon.

46 How do I write Components for Paloose?

I have written an example component here (tgz) that is a "template" for writing generator, transformer and serializer components; also included is an example exceptions class used by the template. Current releases of Paloose include it in the folder `resources/templates/`. Also have a look at the various existing components that make up Paloose.

46 Do you have any support for iPhone?

Yes. The browser selector can detect the iPhone to allow separate pipelines to be run.

46 Help! I have tried everything but I am still getting page not found.

The usual problem here that you have not set up the `.htaccess` file up correctly. Remember that Apache will serve all your requests, including ones that are destined for Paloose unless you tell it otherwise. For example if you want requests for, say, `http://<hostname>/<site-path>/file.asm`, to be processed by Paloose then you would have to add the following line into your `.htaccess` file:

```
RewriteRule (.+)\.asm paloose.php?url=$1.asm [L,qsappend]
```

Remember though that the one type of file extension that this method does not handle is PHP, so putting

```
RewriteRule (.+)\.php paloose.php?url=$1.php [L,qsappend]
```

in the `.htaccess` file will cause an infinite loop.

46 Help! Why am I getting "Class 'XsltProcessor' not found" errors

99% of the time this shows that you are running PHP5 without the XML/XSL support that there should be. Try recompiling PHP5 with the following configuration parameters included

```
--with-xml --with-libxml-dir=<dir path> --with-xsl
```

46 Help! Why am I getting "Parse error: syntax error, unexpected '=', expecting '(' in /../../paloose/lib/Paloose.php on line xx" errors

This means you are still running PHP4. You need to run PHP5. Speak to your ISP to provide PHP5. The other reason may be that your `.htaccess` file has not been set properly to use PHP5.

46 Help! Why am I getting "Input file not found" as the only browser output?

See FAQ entry about the `.htaccess` file, this is the usual problem. Also make sure that you do not overwrite a system `.htaccess` that is required. That is why it is best to have your site in a separate folder.

46 Are there schemas for the XML used in this site?

Yes the basic schema is written in RELAX NG. They are not an essential to running Paloose or understanding, they are merely given out of interest; although I would urge the use of schemas to reduce errors in your XML files.

- The primary page schema is page.rng.

They are pretty versions of the raw RELAX NG which are easier to read. If anyone would like a copy of the transforms necessary to produce the pretty-printed version please EMAIL me and I will consider putting them up as a download.

46 Why do you not use schemas for the sitemaps?

This is a fairly contentious subject. Sitemaps are built in such expandable form that producing a schema in something like RELAX NG or XML-Schema is very difficult — they simply do not have a rich enough structure to do what I think is necessary without having to use Schematron additions. The closest that I got to producing one used DSD-2. I wrote an extensible set of schemas which worked fine, but are now sadly out of date. I also produced schemas for Dublin Core, Jakarta Ant and RDF (although RDF is a major problem because it is impossible to write a universal validating XML schema for it — it ain't the right sort of grammar and the current 2004 specification for RDF is flawed), all of which can be found on the DSD-2 home page. DSD-2 is the best schema language we never had, sad ....

46 Technical Trivia

The code that builds Paloose is a little rough and ready in places. This is my first PHP program (other than the odd embedded line within HTML) and it shows. I am more used to program in strongly typed languages having had a lifetime using Algol, Coral66, Algol68, Pascal, Java at al; as well as more obscure offerings such as OmniMark, ELLA, Abel, TeX. I have programmed in Perl for many years and so PHP was not a wholly new experience. I am sure that I have used 10 lines in places where PHP5 would let me use one; never mind — it suffices, the future will refine the code.

Index