The Paloose Web Engine (1.13.0)

Hugh Field-Richards

December 2022

Contents

Preface

Paloose was written when I retired after a (too) long career designing computer hardware and software. I am now a professional music engraver and Paloose was written to keep my technical muscles in trim and support a variety of Web sites, including Hop Vine Music[1] and the Paloose online documentation[2].

The original name chosen was "Papoose" which, in the UK, 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, a simple change of letter gave "Paloose" (or "Palouse"), which is another term for the Appaloosa horse; something dear to the my heart as, until recently, I was privileged to have had one of my own, "Zitty" (Astral Elite) who I rode until she reached the grand old age of 29 and then became a "house pet" until she died aged 33.

[1] http://www.hopvine-music.com/
[2] http://www.paloose.org/

Chapter 1

Introduction

Paloose is a simplified version of the Cocoon data-pipelined Web engine, using PHP5 instead of Java. It resulted from scratching a long standing personal itch: that there are very few ISPs who 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 to support several sites for over a decade 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 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.

Paloose 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 for 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. Although it is not essential, a knowledge of Cocoon is useful.

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

1.1 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 in Figure 1.1:

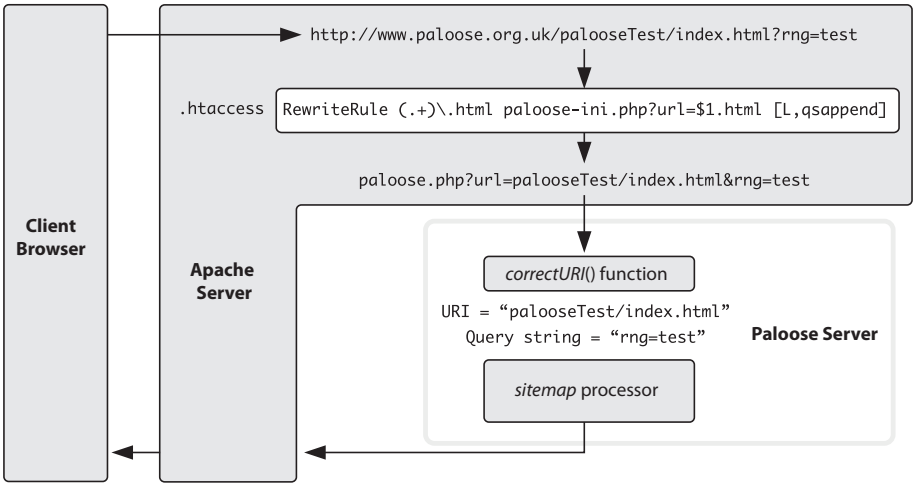


Figure 1.1: Server Client

1.2 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 shown in Figure 1.2.

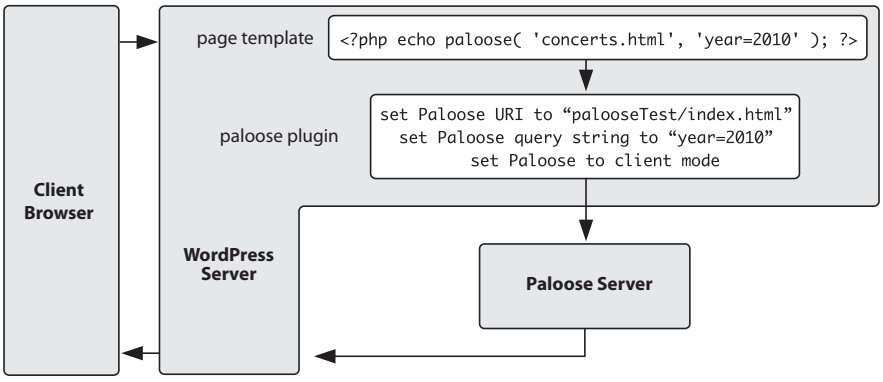


Figure 1.2: Paloose Client

It is worth mentioning that using Paloose as a client with WordPress is only necessary if a pipelined approach to the transform is required. It is perfectly possible, even desirable, for a single transform pipeline to be used without the Paloose plug-in by having the transform code (XSL) called directly from the embedded PHP within the WordPress page.

Chapter 2

Installing and Configuring

Installation is very simple — it is just making sure that the Paloose directory is in the right place. Unpack the download into the destination directory. For example, running on a Mac Mini running Mac OSX this would be in '`/Library/WebServer/Documents`' which 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
${ROOT_DIR}/paloose hsfr$
```

The only thing that must be done to get Paloose to run is to make sure that the web server (Apache usually) is running 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`'. Also it is necessary to turn off Sablotron support and use *libxml*.

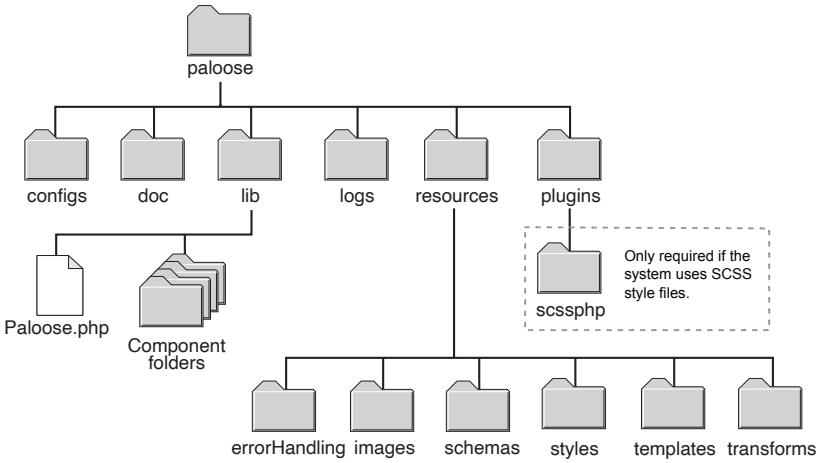


Figure 2.1: Paloose Folders

After installing Paloose as above, there should be a directory structure shown in Figure 2.1. The '`Paloose.php`' file the main entry point into Paloose from the file '`paloose-ini.php`' in the site's root folder.

2.1 Configuring Paloose

Configuring Paloose requires Paloose to be told where the various components are stored and is dependant on what system you are running. Note that a standard install requires very few changes. A server based system uses a simple calling program in your site home page. Using Paloose as a server will also require setting a suitable '`.htaccess`' file, see Section 3.

Unlike the server configuration, if Paloose as is being used a client, the set up of the system is done by the admin page of the respective server that is using Paloose as a client. For example with WordPress the "Settings → Paloose" Plugin menu item takes you to the configuration page where all the variables can be set. A '`.htaccess`' file is not needed either since that functionality is being done by the calling server.

2.1.1 Configuring Paloose as a Server to Apache

Paloose configuration is done through a single file. This file can be called anything (in the current sites case it is '`paloose-ini.php`' in the root directory of the site). If you wish to change the name then this change must be reflected in the *.htaccess* file, see Section 3.

Warning

This configuration information is only applicable to Paloose versions after 1.5.0. Earlier versions are no longer supported and are deprecated.

Configuration is basically a question of telling Paloose where everything is. There are a set of constant definitions which need setting for correct operation. If you leave them commented out Paloose will assign defaults, (see the '`lib/Paloose.php`' file to see default values.)

2.1.2 Configuration Variables (may need to be changed)

The following definitions can all be changed if required. The values shown below are those that are used in the Paloose site.

`PALOOSE_DIRECTORY`

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. It is possible to change between different versions of Paloose merely by changing this line. For example, if there is a compact version of Paloose running (no comments/logging) then the definition line might be '`define( 'PALOOSE_DIRECTORY', '../paloose-compact' );`'.

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

`PALOOSE_LOGGER_CONFIG`

Where the main *log4php* logger configuration file is kept. You can use Paloose pseudo protocols here. In the example below the log file configuration file is in user's site in the sub-directory '`configs/`'. Now that the logging code is separate from the main Paloose code the constant '`LOG4PHP_DIR`' points to where the root of the log4php code is held (relative to the site path).

```
define( 'PALOOSE_LOG4PHP_DIRECTORY', '../log4php' );
define( 'PALOOSE_LOGGER_CONFIGURATION_FILE',
        '../[site root directory]/configs/Paloose.xml' );
```

`PALOOSE_CACHE_DIR`

This is where the Paloose caches are held. It does not include the images cache directory. Paths are relative to the base directory of the Web site — where the '`paloose.php`' file is in your project. Make sure that this directory has the correct permissions — probably safe to allow write to all. In the following the cache is held relative to the Web site root.

```
define( 'PALOOSE_CACHE_DIR', 'resources/cache' );
```

`PALOOSE_ROOT_SITEMAP`

Normally should not have to change this. The directory is always the top level user directory. The site root directory is the location of your sites file and where the root sitemap sites (as well the configuration file that you are looking at).

```
define( 'PALOOSE_ROOT_SITEMAP', 'sitemap.xmap' );
define( 'PALOOSE_SITE_ROOT_DIRECTORY', '../pp' );
```

`PALOOSE_USER_EXCEPTION_xxx`

The error page to give back for User errors (errors in sitemap etc). They are not run time errors (such as missing page: code 404). If you override these to your area make sure that you use an absolute directory path — you can use the '`context://`' pseudo-protocol here.

Be careful of the path for 'USER_EXCEPTION_STYLE' — it looks like an absolute one, but is actually relative to the Apache root document directory. It must always have a leading path separator.

```
define( 'PALOOSE_USER_EXCEPTION_PAGE',
        'resource://resources/errorHandling/userError.html' );
define( 'PALOOSE_USER_EXCEPTION_TRANSFORM',
        'resource://resources/transforms/errorPage2html.xsl' );
define( 'PALOOSE_USER_EXCEPTION_STYLE',
        '/paloose/resources/styles/userError.css' );
```

PALOOSE_GALLERY_INDEX

IMAGEMAGICK_BIN

The file that contains the gallery information in each gallery directory. See Sections 7.11 and 14 for more information about the gallery. The '`ImageMagick`' directory is there if the PATH on the server is not set to pick up correct '`ImageMagick`' binaries.

```
define( 'PALOOSE_GALLERY_INDEX', 'gallery.xml' );
define( 'IMAGEMAGICK_BIN', '/usr/local/bin/' );
```

PALOOSE_INTERNAL_EXCEPTION_xxx

The error page to give back for internal (programming) errors (errors in sitemap etc). These should not need to be overridden. Be careful of the path for '`INTERNAL_EXCEPTION_STYLE`' — it looks like an absolute one, but is actually relative to the Apache root document directory. Must always have a leading path separator.

```
define( 'PALOOSE_INTERNAL_EXCEPTION_PAGE',
        'resource://resources/errorHandling/internalError.html' );
define( 'PALOOSE_INTERNAL_EXCEPTION_TRANSFORM',
        'resource://resources/transforms/errorPage2html.xsl' );
define( 'PALOOSE_INTERNAL_EXCEPTION_STYLE',
        '/paloose/resources/styles/internalError.css' );
```

2.1.3 Configuration Variables (you should not need to change)

The following definitions should not be changed unless you have a really good reason and are interested in Paloose deep-magic.

PALOOSE_SITEMAP_NAMESPACE

PALOOSE_SOURCE_NAMESPACE

PALOOSE_DIR_NAMESPACE

These should not need to be changed unless you plan to use another sitemap schema or want to change the source writing and directory results ('`DirectoryGenerator`').

```
define( 'PALOOSE_SITEMAP_NAMESPACE', 'http://apache.org/cocoon/sitemap/1.0' );
define( 'PALOOSE_SOURCE_NAMESPACE', 'http://apache.org/cocoon/source/1.0' );
define( 'PALOOSE_DIR_NAMESPACE', 'http://apache.org/cocoon/directory/2.0' );
```

PALOOSE_I18N_NAMESPACE

Change these with caution — they should only be changed for a different translation mechanism. It requires deep understanding of Paloose internals.

```
define( 'PALOOSE_I18N_NAMESPACE', 'http://apache.org/cocoon/i18n/2.1' );
```

PAGE_HIT_TRANSFORM

Only change this if you want to replace the current page hit mechanism.

```
define( 'PALOOSE_PAGE_HIT_TRANSFORM', 'resource://resources/transforms/PageHit.xsl' );
```

PALOOSE_NAMESPACE

This is the Paloose namespace that will never need to change unless there is a very good reason.[1]

```
define( 'PALOOSE_NAMESPACE', 'http://www.paloose.org/schemas/Paloose/1.0' );
```

Finally ...

[1] It won't change.

Definitely do not change the following lines unless you really, really know what you are doing. Users of versions 1.40 and earlier will note this as the most significant change in how Paloose is called.

```
ini_set( 'track_errors', '1' );
require_once( PALOOSE_DIRECTORY . '/lib/Paloose.php' );
$paloose = new Paloose();
$paloose->run( $_SERVER[ 'REQUEST_URI' ], $_SERVER[ 'QUERY_STRING' ] );
unset( $paloose );
```

Chapter 3

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".

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} hs3ir HelloWorld
${ROOT_DIR} hsfr$ cd HelloWorld
${ROOT_DIR}/HelloWorld hsfr$
```

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
${ROOT_DIR}/HelloWorld hsfr$
```

Create a '`content/page.xml`' file which is the content of the page. Note that there is no style information here at all, only a simplified structure of a page. 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>
```

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. Note that the code that this produces is based on the element. 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.

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 who 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 the 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. 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>
```

Having created 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[1] 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'.

[1] in which the '.htaccess' file sits.

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).

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 the templates folder "`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. There must always be a leading path separator.

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

Open a browser and go to the address '`http://[host-name]/HelloWorld/page.html`'. If all is well you should see something akin to Figure 3.1.

Figure 3.1: First Site

Chapter 4

Sitemaps

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.

4.1 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>
```

4.2 Sitemap Namespace

The Paloose namespace is identical to that used by Cocoon[1], 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

[1] `http://apache.org/cocoon/sitemap/1.0`

do this yourself note that the namespace is changed in the main Paloose configuration file in the user's site.

4.3 Components

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

4.3.1 Regular Expression Pattern Matching

Note

Note that 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[2]. 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>
```

4.4 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 three preset modules for variables: *global*, *error* and *request-param*.

[2]https://www.php.net/manual/en/reference.pcre.pattern.syntax.php

4.4.1 Global Variables

Global variables can be declared within each sitemap 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>
         <variable name="composer" value="Bach"/>
      </global-variables>
   </map:component-configurations>

   ...
```

Note

Note that this is a change from all versions prior to 1.13.0.

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.

These can be accessed by using '`{global:__0},{global:__1}, {global:__2}, ...`' respectively.

4.4.2 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" />
```

4.4.3 Handling Pipeline Errors

When an error is detected 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 Section 4.6.

4.4.4 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 the original Cocoon. Figures 4.1 and 4.2 should make the paths clear. Thus if a style file from the user's site is required the path would be "'`context://styles/file.css`'" (or "'`cocoon://../styles/file.css`'".

`context://`	get a resource from 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

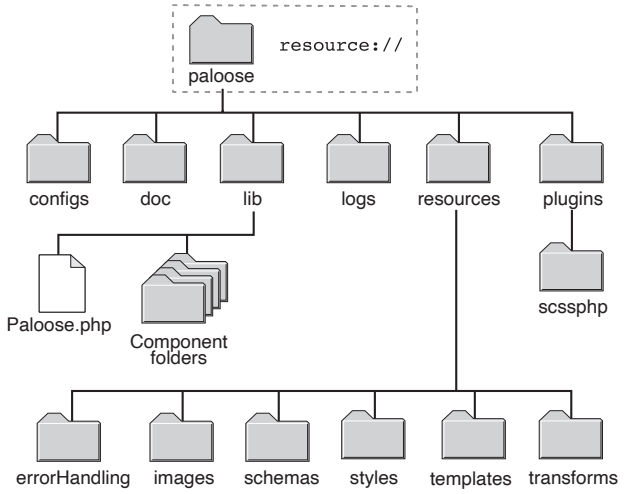


Figure 4.1: Protocol References within Paloose Structure

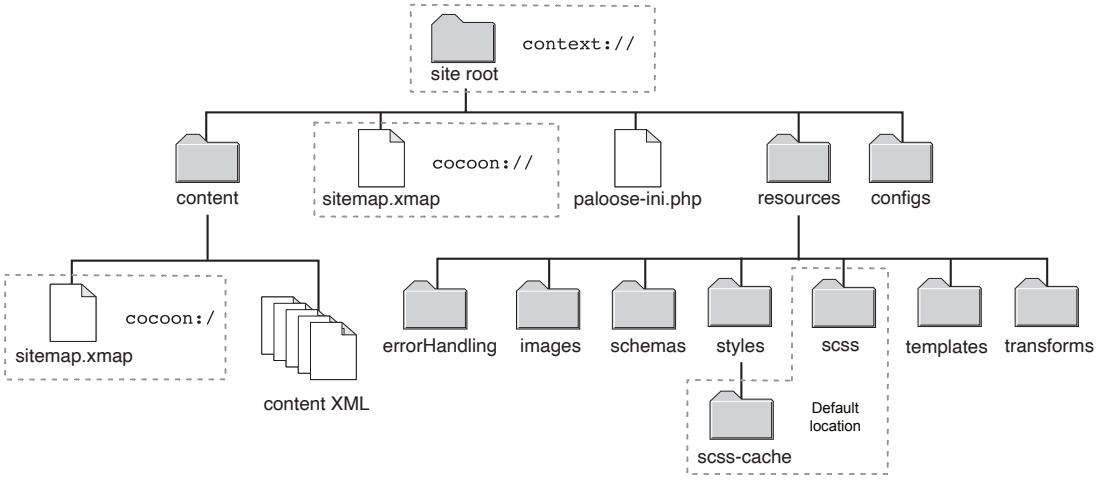


Figure 4.2: Protocol References within Site Structure

4.4.5 Mounting sitemaps (`mount`)

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 amount of inheritance of components so that they do not have to be re-defined. 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 of 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. ***Cocoon insists on this attribute being present (but can be an empty string). Paloose makes this attribute optional.***

Note

There will be a performance penalty if subsitemaps are nested too deep.

4.4.6 Redirecting Requests (`redirect-to`)

It is possible to redirect a request to Paloose to a completely different URI (on the same or different site). For example if the Paloose documentation was split onto a completely different server then we would 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.

4.5 Views

Views are means of interrupting the normal pipeline flow under the control of the user via the query 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`'.

4.6 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.

Note

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>
```

4.6.1 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.

```
<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>
```

Chapter 5

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.

BrowserSelector	selects pipeline fragment according to user's browser.
MobileSelector	selects pipeline fragment according to user's browser when mobile.
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 the request parameter from the query string.
ResourceExistsSelector	selects a pipeline on condition of whether a resource (file) exists.

5.1 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.

5.2 `BrowserSelector`

Declare the '`BrowserSelector`' component in the sitemap:

```
<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 the Paloose web 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>
```

5.3 MobileSelector

Declare the '`MobileSelector`' component in the sitemap:

```
<map:components>

  <map:selectors default="mobile">
      <map:selector name="mobile" src="resource://lib/selection/MobileSelector"/>
   </map:selectors>

</map:components>
```

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 the Paloose web 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="mobile">
         <map:when test="mobile">
            <map:transform src="context://resources/transforms/page2mobile.xsl" label="page-transform">
               <map:parameter name="page" value="home"/>
            </map:transform>
         </map:when>
         <map:otherwise>
            <map:transform src="context://resources/transforms/page2xhtml.xsl" label="page-transform">
               <map:parameter name="page" value="home"/>
            </map:transform>
         </map:otherwise>
      </map:select>
   </map:select>
   <map:transform src="context://resources/transforms/stripNamespaces.xsl"/>
   <map:serialize type="html" />
</map:match>
```

5.4 RequestParameterSelector

Declare the '`RequestParameterSelector`' component in the sitemap:

```
<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:parameter-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:

```
<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>
```

5.5 RegexpSelector

Declare the *RegexpSelector* component in the sitemap:

```
   <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:

```
      <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.

5.6 `VariableSelector`

Declare the '`VariableSelector`' component in the sitemap:

```
<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:

```
<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 is almost the same as the previous example.

5.7 ResourceExistsSelector

Declare the *ResourceExistsSelector* component in the sitemap:

```
<map:selectors default="resource-exists">
       <map:selector name="resource-exists"
                     src="resource://lib/selection/ResourceExistsSelector"/>
</map:selectors>
```

where

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

The following example shows how it can be used to select between two transforms. It is taken from an experimental CMS system based on Paloose and shows several other selectors as well:

```
<map:match pattern="page.html">
    <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/{global:defaultTheme}/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="requestId" value="{request-param:reqid}"/>
          <map:parameter name="siteIdent" value="{global:siteIdent}"/>
          <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://sites/{global:siteIdent}/resources/transforms/buildSite.xsl"
          label="buildSite-transform"/>

       <map:select type="regexp-selector">
          <map:parameter name="test-value" value="{request-param:reqid}"/>
          <map:when test="/nb.+/">
             <map:transform
src="context://sites/{global:siteIdent}/resources/transforms/widgets/buildSiteWidget-notebook.xsl"
label="buildNotebook-transform"/>
```

```
            <map:select type="resource-exists">
               <map:when
test="context://themes/{global:currentTheme}/resources/transforms/buildTheme.xsl">
                  <map:transform
src="context://themes/{global:currentTheme}/resources/transforms/buildTheme.xsl"
           label="buildTheme-transform"/>
               </map:when>
               <map:otherwise/>
            </map:select>
         </map:when>
         <map:otherwise/>
      </map:select>

      <map:select type="resource-exists">
         <map:when
            test="context://themes/{global:currentTheme}/resources/transforms/buildTheme.xsl">
            <map:transform
               src="context://themes/{global:currentTheme}/resources/transforms/buildTheme.xsl"
               label="buildTheme-transform"/>
         </map:when>
         <map:otherwise>
            <map:transform
               src="context://themes/{global:parentTheme}/resources/transforms/buildTheme.xsl"
               label="buildTheme-transform"/>
         </map:otherwise>
      </map:select>
      <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 src="context://system/resources/transforms/stripNamespaces.xsl"/>
      <map:serialize type="xhtml"/>
   </map:act>
</map:match>
```

Chapter 6

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 to inject into the pipeline.

6.1 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.

6.2 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

6.3 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`' (Section 6.4).

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.

6.3.1 Caching Support (available after Version 1.3.0)

The '`FileGenerator`' component supports caching of the data within the pipeline. The input file is normally checked for being well-formed (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 overridden 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>
```

6.4 `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>
```

6.4.1 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.

6.4.2 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

It is important to note 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 for '`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 overridden 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>
```

6.5 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 should be aggregated together. 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>
```

6.6 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 recursion 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.

If the results of the scan are empty a single element is returned to indicate an empty scan:

```
<dir:directory ... >
   <dir:empty>
</dir:directory>
```

6.7 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.

6.7.1 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 Â©
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="@S0@"/>
   </g:head>
   <g:subm id="@S0@">
      <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>
   ...
```

Chapter 7

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).

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.
VariableTransformer	expands any embedded Paloose variables within the text.
String2XMLTransformer	translate a string into an equivalent DOM structure.

7.1 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.

7.2 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`'.

7.2.1 Caching Support

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>
```

7.3 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>
```

7.3.1 Simple Example

Within the source XML include statements are entered simply as this following 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:

```
<?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>

<?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>

<?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>
```

7.4 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:

```
<?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 entity, '`testTitle`' is handled by the '`EntityTransformer`'.

Running the above through a suitable serialiser will display as:

Entity Test 1

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

Figure 7.1: Entity Output

7.5 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`

7.5.1 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`'.

7.5.2 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>
```

7.5.3 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)

```
<?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>
```

7.5.4 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:

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

The '`source:insert`' comes in several flavours:

7.5.4.1 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`':

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

7.5.4.2 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:

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

7.5.4.3 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.

7.5.4.4 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:

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

7.5.5 Source Delete

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

```
<source:delete>
   <source:source/>
</source:delete>
```

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>
```

7.6 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).

7.6.1 Error Reporting

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 * from composer </default:message>
   </default:sql-error>
</page:content>
```

7.6.2 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).

7.6.3 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.

7.6.4 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 might be:

```
<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 query attributes have changed 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.

7.6.5 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.

7.7 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 declaration 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>
```

7.8 ModuleWriteTransformer

The '`ModuleWriteTransformer`' updates the module variables from code within the input stream. A typical 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.

7.9 LogTransformer

The '`LogTransformer`' 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>
```

7.10 PasswordTransformer

The '`PasswordTransformer`' 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 '`PasswordTransformer`' the tag becomes:

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

7.11 GalleryTransformer

The Gallery Transformer provides a simple Picture Gallery addition for Paloose. The root sitemap needs to have '`GalleryTransformer`' declared as a component:

```
<map:components>
     <map:transformers default="xslt">
        <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:transformers>
```

where

root	the root of the gallery (a folder that holds the entire gallery). This can be overridden by the pipeline component, or by an explicit declaration in the XML content (see below).
image-cache	the cache where the scaled image and the thumbnails are kept. In this case in the '`resources/images/cache/`' folder within the root of the site.
max-thumbnail-width *max-thumbnail-height*	the size, in pixels, of the thumbnail image.
max-width, max-height	the maximum size, in pixels, of the displayed image. If either of the dimensions of the original image exceed these values then a suitable scaling is applied to the cached image.
resize	should the original image be scaled to the max values (1), or left as it is (0).

In order to use the transformer we need to put the following:

```
<map:match pattern="*.xml">
   <map:generate src="context://content/{1}.xml"/>
   <map:transform type="gallery"/>
   <map:transform src="context://resources/transforms/gallery2html.xsl"/>
   <map:serialize/>
</map:match>
```

Note that there is an XSL gallery transformer, '`gallery2html.xsl`'. An example and more details can be found in Section 14.

7.12 DirectoryTransformer

Directory transformers read a specified directory and insert it into the document according to a specified element. So a typical use of '`DirectoryTransformer`' would be (from an experimental CMS system):

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

<map:pipeline>
  <map:match pattern="page.html">
     <map:act type="auth-protect">
        <map:parameter name="handler" value="adminHandler"/>

        ...

        <map:select type="regexp-selector">
           <map:parameter name="test-value" value="{request-param:reqid}"/>
           <map:when test="/allPages/">
              <map:transform src="{global:adminDir}/resources/transforms/buildAdminSite.xsl"
                             label="buildSite-transform"/>
              <map:transform type="variables" label="buildVariables-transform"/>
              <map:transform type="directory" label="directory-transform"/>
              <map:transform src="{global:adminDir}/resources/transforms/buildPagesList.xsl"
                             label="buildPagesList-transform"/>
           </map:when>
           <map:otherwise>
              <map:transform src="{global:adminDir}/resources/transforms/buildAdminSite.xsl"
                             label="buildSite-transform"/>
           </map:otherwise>
        </map:select>

        ...

     </map:act>
  </map:match>
</map:pipeline>
```

The attributes are identical to the '`DirectoryGenerator`' component. Using the transformer is very simple (based on the CMS example above) using the <`pcms:folder`> element:

```
  <pcms:structure>
     <p>"{global:siteIdent}" Pages</p>
     <pcms:folder src="context://sites/{global:siteIdent}/articles" include="/^id-/"/>
  </pcms:structure>
```

where the attributes of <`pcms:folder`> are

src	the root directory to start scanning.
depth	(optional) the recursion 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.

This will give back a list of items whose name follows the pattern '`id-*****`' from the '`articles`' directory. The format of the output is identical to that from '`DirectoryGenerator`':

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.

If the results of the scan are empty a single element is returned to indicate an empty scan:

```
<dir:directory ... >
   <dir:empty>
</dir:directory>
```

7.13 VariableTransformer

The '`VariableTransformer`' expands any embedded Paloose variables within the text. A typical use would be (from a trial CMS system based on Paloose)

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

<map:pipelines>
   <map:pipeline>

      <map:match pattern="login.html">
         <map:generate src="{global:themesDir}/{global:currentTheme}/data/body.xml"
            label="raw-xml"/>
         <map:transform src="{global:systemDir}/resources/transforms/buildBody.xsl"
            label="buildBody-transform">
            <map:parameter name="requestId" value="login"/>
            <map:parameter name="siteIdent" value="{request-param:site}"/>
            <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:parameter name="admin" value="1"/>
         </map:transform>
        <map:transform src="{global:adminDir}/resources/transforms/buildAdminPanels.xsl"
            label="buildPanels-transform"/>
        <map:transform src="{global:adminDir}/resources/transforms/buildAdminSite.xsl"
            label="buildSite-transform"/>
        <map:transform
            src="{global:themesDir}/{global:currentTheme}/resources/transforms/buildTheme.xsl"
            label="buildTheme-transform"/>
        <map:transform type="variables" label="buildVariables-transform"/>
        <map:transform src="{global:systemDir}/resources/transforms/normaliseProperties.xsl"
            label="normalise-transform"/>
        <map:transform src="{global:systemDir}/resources/transforms/page2xhtml.xsl"
            label="page-transform"/>
        <map:serialize type="xhtml"/>
      </map:match>
   </map:pipeline>
</map:pipelines>
```

Within the page there is a widget file whose body of XML is

```
<pcms:structure>
   <p>"{global:siteIdent}" PCMS Site Copyright 2005 &ndash;
   2017 Hugh Field-Richards. All Rights Reserved</p>
</pcms:structure>
```

The Variable Transformer would replace the '`{global:siteIdent}`' with the value of the global value '`siteIdent`'.

7.14 String2XMLTransformer

There are occasions when it is necessary to translate a string into an equivalent DOM structure. Currently there is a single attribute

root-node the containing node of the string to be translated.

For example in the experimental Paloose CMS the TinyMCE editor[1] the updated (saved) content is sent to the server as a POST string. This is an HTML raw string that must be converted into a HTML DOM.

```
<map:pipeline>
   <map:match pattern="updatePage.html">
    <map:act type="auth-protect">
       <map:parameter name="handler" value="adminHandler"/>

       <map:aggregate element="root">
          <map:part src="cocoon:/updatePage-result.xml" element="updatePage-result"
             strip-root="true"/>
          <map:part src="cocoon:/updatePage-2.xml" element="updatePage-page"
             strip-root="true"/>
       </map:aggregate>
       <!-- Display result and edit page -->
    </map:act>
   </map:match>
</map:pipeline>

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

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

<map:match pattern="updatePage-result.xml">
 <map:generate src="{global:themesDir}/{global:currentTheme}/data/body.xml" type="px"
    label="raw-xml"/>

 <map:transform src="{global:adminDir}/resources/transforms/updatePage.xsl"
    label="updatePage-transform">
    <map:parameter name="whichArticle" value="{request-param:whichArticle}"/>
    <map:parameter name="editField" value="{request-param:editField}"/>
    <map:parameter name="requestId" value="edit"/>
    <map:parameter name="siteIdent" value="{request-param:site}"/>
    <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:parameter name="admin" value="1"/>
 </map:transform>
 <map:transform type="string2xml" label="string2xml-transform">
    <map:parameter name="root-node" value="pcms:editField"/>
 </map:transform>
 <map:transform src="{global:adminDir}/resources/transforms/normalizeEdit.xsl"
    label="updatePage-transform"/>
 <map:transform type="variables" label="buildVariables-transform"/>
 <map:transform type="sourceWrite" label="buildVariables-transform"/>
 <map:serialize type="xml"/>
</map:match>

<map:match pattern="updatePage-page.xml">
 <map:generate src="{global:themesDir}/{global:currentTheme}/data/body.xml" type="px"
    label="raw-xml"/>
 <map:transform src="{global:systemDir}/resources/transforms/buildBody.xsl"
    label="buildBody-transform">
    <map:parameter name="requestId" value="{request-param:reqid}"/>
    <map:parameter name="siteIdent" value="{request-param:site}"/>
    <map:parameter name="themesDir" value="{global:themesDir}"/>
```

[1] https://www.tinymce.com

```
    <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:parameter name="admin" value="1"/>
 </map:transform>
 <map:transform src="{global:adminDir}/resources/transforms/buildAdminPanels.xsl"
    label="buildPanels-transform"/>
 <map:transform src="{global:adminDir}/resources/transforms/buildAdminSite.xsl"
    label="buildSite-transform"/>
 <map:transform type="variables" label="buildVariables-transform"/>
 <map:transform type="directory" label="directory-transform"/>
 <map:transform src="{global:adminDir}/resources/transforms/buildPagesList.xsl"
    label="buildPagesList-transform"/>
 <map:serialize type="xml"/>
</map:match>
```

which builds an XML page which contains the POSTed string as an XML scrap for writing back into the article file being edited.

7.15 SCSSCompiler

The SCSS Compiler provides a mechanism to translate style files written to the SCSS format into CSS files. The actual compiler is the `SCSSPHP` compiler that is added as a plugin. Paloose will work fine if you don't need this facility and you can remove `SCSSPHP` from the plugins folder.

Note

Note that the `SCSSCompiler` code relies on the `SCSSPHP` (`https://scssphp.github.io/scssphp/`) module by Leaf Corcoran and is copyright to him. It should be installed in the Paloose plugin folder. This will happen naturally when a full Paloose installation is done. My thanks to him for this work.

Warning

The current version of `SCSSPHP` as loaded from the github site (V1.3) had a small problem: the `OutputStyle.php` file was not included in the `Compiler.php` file correctly (the actual error was "`could not find ../ScssPhp/ScssPhp/OutputStyle`"). I resolved this by moving the *OutputStyle* class into the `Compiler.php` file. The normal install of Paloose will always have this updated version included. It may be a quirk of my local server and a vanilla installation will work fine on your system — you have been warned! It has been reported.

The SCSS compiler is considered to be a transformer from its behaviour: it must not appear first or last in a pipeline line (i.e. not before a generator or after a serializer). It is declared in the transformers declaration of the sitemap as

```
<map:transformers default="xslt">
  <map:transformer name="scss" src="resource://lib/scss/SCSSCompiler">
     <map:parameter name="src" value="context://resources/scss"/>
     <map:parameter name="dst" value="context://resources/styles"/>
     <map:parameter name="compact" value="no"/>
  </map:transformer>
```

```
</map:transformers>
```
where

src	(optional) defines where your SCSS files are stored. This would normally be a folder within the resources folder at the same level as the CSS folder (defaults to "`context://resources/scss`" if not given).
dst	(optional) is the destination file for the CSS generated files. It is also the folder where the compilers cache folder (`scss_cache`) is held (defaults to "`context://resources/styles`" if not given).
compact	(optional) defines where the CSS generated is in a compact form (defaults to "`no`" if not given). Permitted values are "`yes`", "`true`" or "`1`" and "`no`", "`false`" or "`0`".

The default values given above should always be used unless there is a particular reason not to, the structure shown in Figure 4.2 should make this clear. Using the component is simple, place it anywhere within a sitemap pipeline as a transformer, for example

```
<map:match pattern="*.html">
    <map:aggregate element="root" label="aggr-content">
       <map:part src="context://content/breadcrumbs.xml"
                 element="breadcrumbs" strip-root="true"/>
       <map:part src="context://content/menus.xml"
                 element="menus" strip-root="true"/>
       <map:part src="cocoon:/{1}.xml"
                 element="content" strip-root="true"/>
    </map:aggregate>
    <map:transform src="context://resources/transforms/page2xhtml.xsl"
                   label="page-transform">
       <map:parameter name="page" value="{1}"/>
       <map:parameter name="baseDir" value="context://"/>
    </map:transform>
    <map:transform type="scss" label="scss-content"
    <map:serialize type="xhtml"/>
 </map:match>
```

which would compile any *scss* file in the *src* folder to the *dst* folder. If anything in the pipeline relies on generated CSS then the pipeline stage should be moved earlier in the pipeline (but not before the generation stage (the aggregate in the example following). If the *scss* file has not changed then the compiler uses the already existing css file.

```
<map:match pattern="*.html">
   <map:aggregate element="root" label="aggr-content">
      <map:part src="context://content/breadcrumbs.xml"
                  element="breadcrumbs" strip-root="true"/>
      <map:part src="context://content/menus.xml"
                element="menus" strip-root="true"/>
      <map:part src="cocoon:/{1}.xml"
                element="content" strip-root="true"/>
   </map:aggregate>
   <map:transform type="scss" label="scss-content"
   <map:transform src="context://resources/transforms/page2xhtml.xsl"
                  label="page-transform">
      <map:parameter name="page" value="{1}"/>
      <map:parameter name="baseDir" value="context://"/>
   </map:transform>
   <map:serialize type="xhtml"/>
</map:match>
```

Chapter 8

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 correct 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.

8.1 Component Declaration

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

Note

Note that this is a change from all versions prior to 1.13.0.

```
<map:serializers default="xml">
    <map:serializer name="html" mime-type="text/html" src="resource://lib/serialization/HTMLSerializer">
       <map:property name="doctype-public" value="-//W3C//DTD HTML 4.01 Transitional//EN"/>
       <map:property name="doctype-system" value="http://www.w3.org/TR/html4/loose.dtd"/>
       <map:property name="encoding" value="UTF-8"/>
    </map:serializer>
    ...
 </map:serializers>
```

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

8.2 HTMLSerializer

The HTML Serializer outputs a plain HTML stream suitable for most browsers. A typical use of '`HTMLSerializer`' would be

```
<map:components>
   ...
   <map:serializers default="xml">
      <map:serializer name="html" mime-type="text/html" src="resource://lib/serialization/HTMLSerializer">
         <map:property name="doctype-public" value="-//W3C//DTD HTML 4.01 Transitional//EN"/>
         <map:property name="doctype-system" value="http://www.w3.org/TR/html4/loose.dtd"/>
         <map:property name="encoding" value="UTF-8"/>
   </map:serializers>
</map:components>

<map:pipelines>
   <map:pipeline>

      <map:match pattern="*.html">
         <!-- generate -->
         <!-- some transformations -->
         <map:serialize type="html"/>
      </map:match>
   </map:pipeline>
</map:pipelines>
```

Given that the last transformer gave HTML as an output the '`HTMLSerializer`' would produce the following typical output:

```
<!DOCTYPE html
   PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
   "http://www.w3.org/TR/html4/loose.dtd">
<html>
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
      <!-- Remaining head tags from DOM -->
   </head>
   <body>
      <!-- Remaining body tags from DOM -->
   </body>
</html>
```

8.3 XHTMLSerializer

The XHTML Serializer is similar to the '`HTMLSerializer`' except that it produces valid XHTML as an XML document. For example:

```
<map:components>
   ...
   <map:serializers default="xml">
     <map:serializer name="xhtml" mime-type="text/html" src="resource://lib/serialization/XHTMLSerializer">
         <map:property name="doctype-public" value="-//W3C//DTD XHTML 1.0 Transitional//EN"/>
         <map:property name="doctype-system" value="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"/>
         <map:property name="encoding" value="UTF-8"/>
      </map:serializer>
   </map:serializers>
</map:components>

<map:pipelines>
   <map:pipeline>

      <map:match pattern="*.html">
         <!-- generate -->
         <!-- some transformations -->
         <map:serialize type="html"/>
      </map:match>
   </map:pipeline>
</map:pipelines>
```

Again, given a suitable HTML input this would produce:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html
   PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
   <head>
      <meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
      <!-- Remaining head tags from DOM -->
   </head>
   <body>
      <!-- Remaining body tags from DOM -->
   </body>
</html>
```

A typical use of the '`XHTMLSerializer`' would be:

```
<map:components>
   ...
   <map:serializers default="xml">
     <map:serializer name="xhtml" mime-type="text/html" src="resource://lib/serialization/XHTMLSerializer">
         <map:property name="doctype-public" value="-//W3C//DTD XHTML 1.0 Transitional//EN"/>
         <map:property name="doctype-system" value="http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd"/>
         <map:property name="encoding" value="UTF-8"/>
      </map:serializer>
   </map:serializers>

</map:components>

<map:pipelines>
   <map:pipeline>

      <map:match pattern="*.html">
         <!-- generate -->
         <!-- some transformations -->
         <map:serialize type="html"/>
      </map:match>
```

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

8.4 XMLSerializer

It is also possible to output the XML within the DOM in the pipeline directly. This is of use in the aggregation process when a document is made up of several XML parts. As an example consider this from the Paloose site content sitemap

```
<map:components>
   ...
   <map:serializers default="xml">
     <map:serializer name="xml" mime-type="text/xml" src="resource://lib/serialization/XMLSerializer">
         <map:property name="doctype-public" value="-//W3C//DTD XHTML 1.0 Strict//EN"/>
         <map:property name="doctype-system" value="http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd"/>
         <map:property name="encoding" value="UTF-8"/>
      </map:serializer>
      <map:serializer name="xml" src="resource://lib/serialization/XMLSerializer"/>
   </map:serializers>
</map:components>

<map:pipelines>
   <map:pipeline>

       <map:match pattern="**.html">
         <map:aggregate element="root" label="aggr-content">
            <map:part src="cocoon:/menus.xml" element="menus" strip-root="true"/>
            ...
         </map:aggregate>
         ...
         <map:serialize type="xhtml" />
      </map:match>

      <map:match pattern="menus.xml">
         <map:generate src="context://content/menus.xml" label="menus-content"/>
         <map:transform type="i18n">
            <map:parameter name="default-catalogue-id" value="menus"/>
         </map:transform>
         <map:serialize/>
      </map:match>

</map:pipelines>
```

The output menus matcher would give the following into the aggregator:

```
<?xml version="1.0"?>
<page:page xmlns:link="http://www.hsfr.org.uk/Schema/Link"
   xmlns:list="http://www.hsfr.org.uk/Schema/List"
   ... >
   ...
</page:page>
```

8.5 TextSerializer

There are occasions where it is useful to output pure text. For this the '`TextSerializer`' is used.

```
<map:components>
   ...
   <map:serializers default="xml">
```

```
      <map:serializer
         name="text"
         src="resource://lib/serialization/TextSerializer"/>
   </map:serializers>

</map:components>

<map:pipelines>
   <map:pipeline>

      <map:match pattern="*.html">
         <!-- generate -->
         <!-- some transformations -->
         <map:serialize type="text"/>
      </map:match>
   </map:pipeline>
</map:pipelines>
```

The output from this would be the text content from all the DOM document tags.

Chapter 9

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.
Cookies Actions	allows the pipeline access to the user's cookies (used in conjunction with ModuleWrite transformer.
Authorisation Actions	allows the pipeline to restrict use until some authorisation takes place (login etc). This is a set of actions: *AuthAction*, *LoginAction*, and *LogoutAction*.

9.1 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="cookies" src="resource://lib/acting/CookiesAction"/>
   <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>
```

9.2 `SendMailAction`

The SendMail Action 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	the IP address of the host to use which will deliver the EMAIL message.
smtp-user	the user name of the mail account.
smtp-password	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.

9.3 CookieslAction

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 '`ModuleWriteTransformer`' 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`').

9.4 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, '`auth-protect`', '`auth-login`' and '`auth-logout`'. 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>
```

Section 9.5 gives an extended example to show how the authorisation is used.

9.5 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 directory structure shown in Figure 9.1.

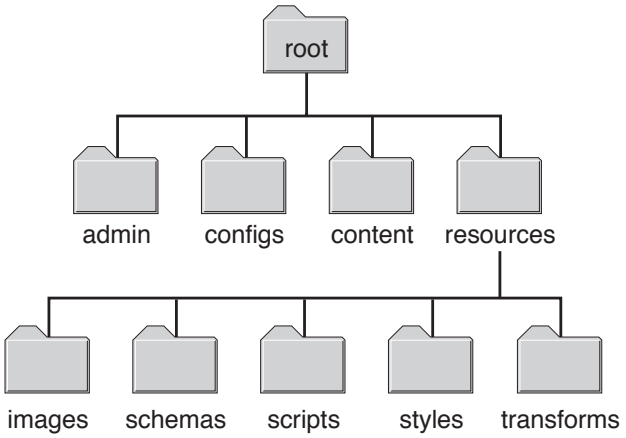


Figure 9.1: Authentication Directory

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:

9.5.1 Authorisation Handler

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

```
<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.***

9.5.2 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`'). A typical entry in the sitemap could be:

```
<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). Figure 9.2 illustrates the relationship of the above code.

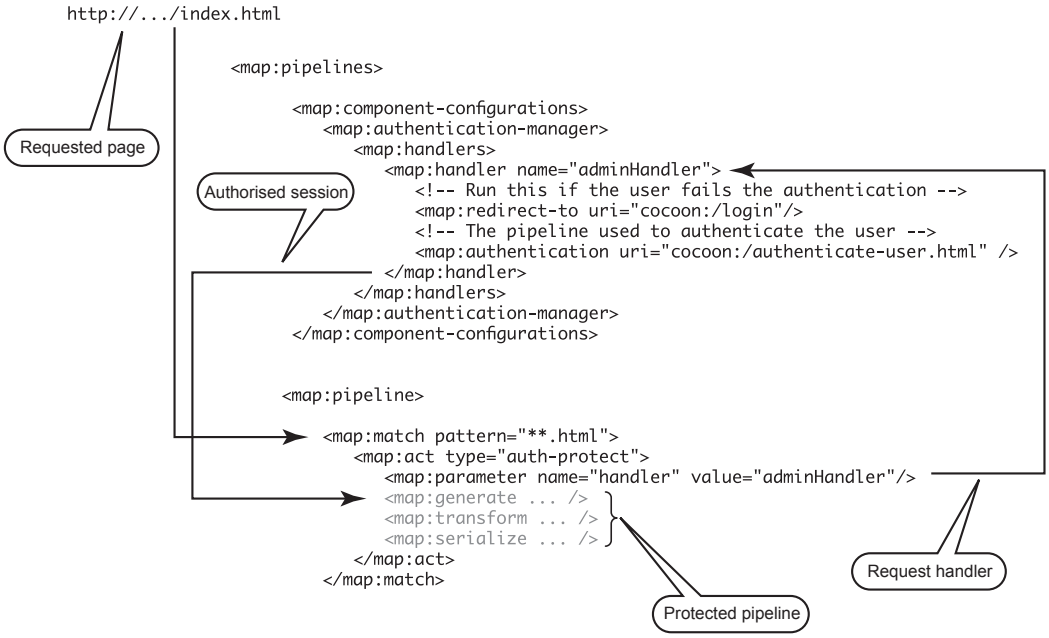


Figure 9.2: Administation Flow

The next section deals with the actual authorising mechanism.

9.6 Authorising the User

The '`adminHandler`' defined in the component configuration section of the sitemap is:

```
<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):

```
<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>

</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.

9.6.1 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.

```
<?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:

```
<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 to 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:

```
<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*.

9.6.2 User Login

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

```
<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:

```
<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. The above is used 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:

```
<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:

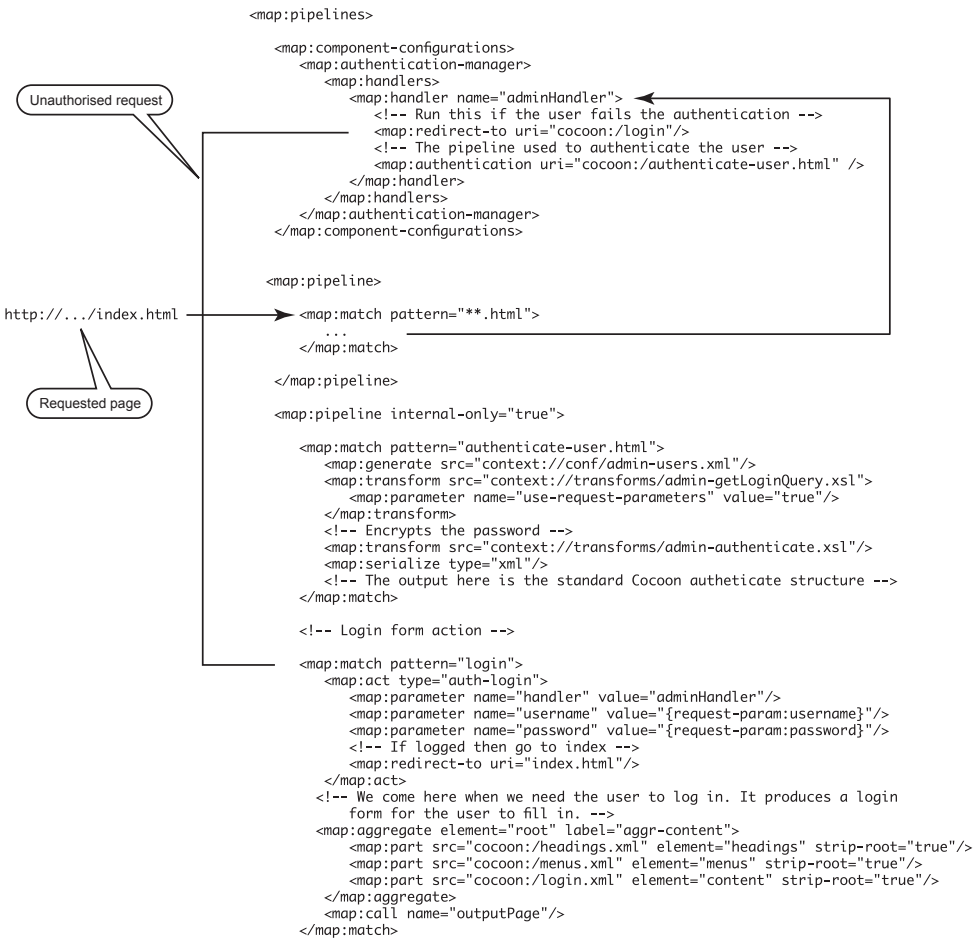


Figure 9.3: Login Flow

9.6.3 User Logout

Logout is relatively simple, for example:

```
<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.

Chapter 10

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.

10.1 Example

Consider a site which requires an admin system of restricted pages (the access system is described in more detail on the action page). The user's data (very simplistic) will consist of a *Username*, *Password* and *Full name*. So a typical data structure used by the login process etc would be:

```
<authentication>
   <users>
      <user>
         <username>hsfr</username>
         <password>xxxxxxxxxxxxxxxxxxxxxxxxxxxx</password>
         <data>
            <fullname>Hugh Field-Richards</fullname>
         </data>
      </user>
   </users>
</authentication>
```

10.1.1 Add User Form

Consider a means of entering a new admin user to an already password protected site. First a simple form is required (the file names are for the test accessible here). I am going to look at just the form not the surrounding content that the example consists of. First in the form we have:

```
<pf:form
      id="addUserTestForm"
      flow="addUserTest"
      continuation="{flow:continuation.id}"
      session="{flow:__flowId}">
   <pf:label>Enter user's details</pf:label>
```

where

- *id (addUserTestForm)* — The identity of this form used in referencing it from the CSS file.
- *flow (addUserTest)* — Defines which flow we are using.
- *continuation ({flow:continuation.id})* — Controls where we are in the flow.
- *session ({flow:__flowId})* — The current flow session (does not necessarily need login).

The two Paloose variables will need expanding on generation so we will need the *PXTemplateTransform* for this when we define the sitemap pipeline. Next we define a set of suitable fields for displaying the user entry form:

```
   <pf:input ref="username" class="usernameField">
      <pf:label>User name</pf:label>
      <pf:value>{flow:username}</pf:value>
      <pf:hint>The user's login username</pf:hint>
      <pf:violations/>
   </pf:input>
```

```
   <pf:input ref="fullname" class="fullnameField">
      <pf:label>Full name</pf:label>
      <pf:value>{flow:fullname}</pf:value>
      <pf:hint>The user's full name (optional)</pf:hint>
      <pf:violations/>
   </pf:input>

   <pf:secret ref="password" class="passwordField">
      <pf:label>Password</pf:label>
      <pf:value>{flow:password}</pf:value>
      <pf:hint>The user's login password</pf:hint>
      <pf:violations/>
   </pf:secret>

   <pf:secret ref="passwordCheck" class="passwordField">
      <pf:label>Check password</pf:label>
      <pf:value>{flow:passwordCheck}</pf:value>
      <pf:hint>The user's login password again to confirm</pf:hint>
      <pf:violations/>
   </pf:secret>
```

The value of each one is derived from the flow data ({*flow:password*} etc.). Finally we define the submit button which transmits the form to the server:

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

- *class (button)* — The class of this control used in referencing it from the CSS file.
- *id (next)* — How the flow script refers to this control.

The next page describes the flow script.

10.1.2 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:

```
<?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:

```
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:

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

We will also need the '`PXTemplateGenerator`':

```
<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:

```
<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 off 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:

```
   <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:

```
<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:

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

Running this will produce a form similar to:

Figure 10.1: Entry Form

10.1.3 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:

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

which runs the '`add`' method we saw previously. However this 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:

```
<?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:

```
   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:

Figure 10.2: Entry Errors

The next page describes the confirmation of data.

10.1.4 Confirming the data

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

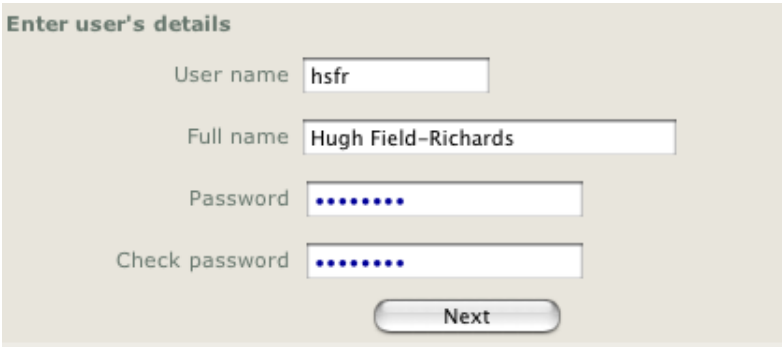


Figure 10.3: Correct Entry Data

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

```
<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:

Figure 10.4: Entry Confirmation

Adding the data to a file can be carried out in a further stage of the pipeline 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:

```
   <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>

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;

       }
    }
}
```

10.2 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.

10.2.1 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.

10.2.2 Simple Input Field

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.

10.2.3 Simple Password Field

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.

10.2.4 Hidden Field

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.

10.2.5 Output Field

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,

10.2.6 Form button

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.

10.2.7 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>
```

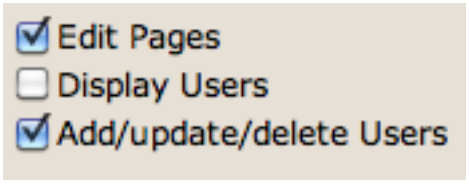
where the attributes are:

- *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.

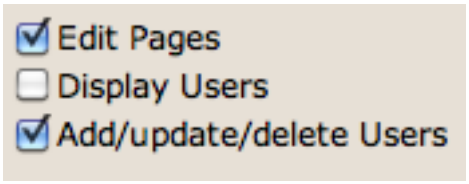
For example:

```
<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:



10.2.8 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>
```

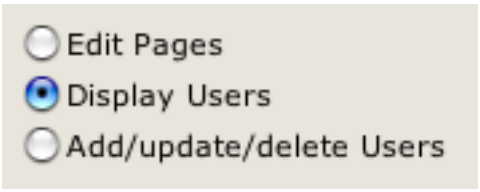
where the attributes are:

- *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:



10.2.9 Text Area

Note that 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>
```

10.2.10 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

10.2.11 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>
```

10.2.12 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>
```

10.2.13 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>
```

Chapter 11

Optimising Paloose Performance

11.1 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 was 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:

```
<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 Figure 11.1 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).

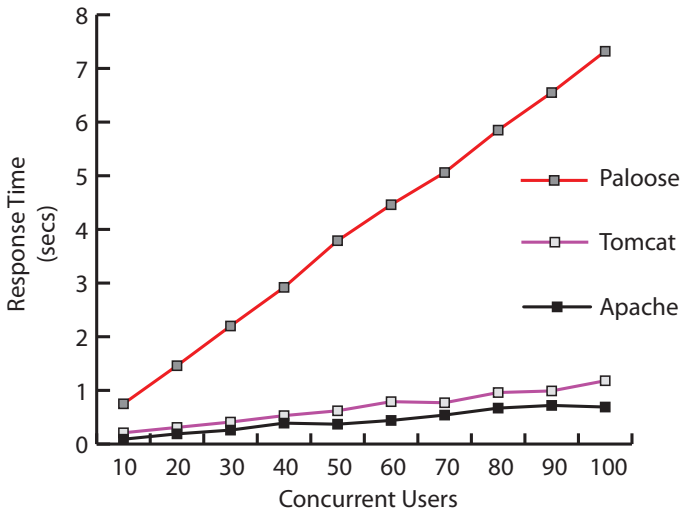


Figure 11.1: Paloose with no Optimisation

11.2 Optimising Paloose Performance

11.2.1 Caching the Sitemap

So how do is this situation improved? 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.

As an experiment a substantial part of Paloose was rewritten to cache the sitemap using a precompiled version. 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:

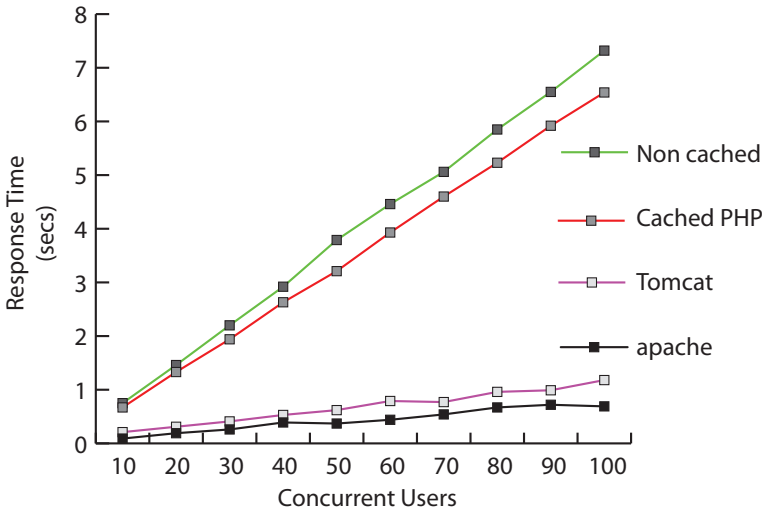


Figure 11.2: Pre-compiled sitemap

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.

11.3 Optimising Paloose Performance

11.3.1 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.

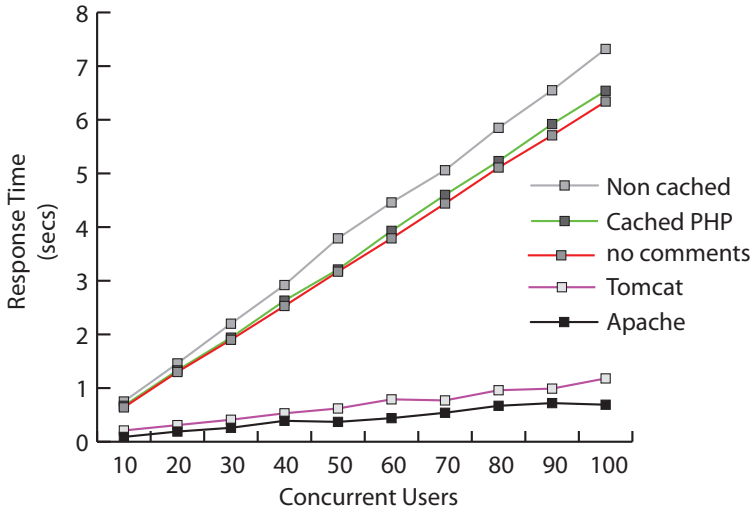


Figure 11.3: No cache with Uncommented PHP

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.

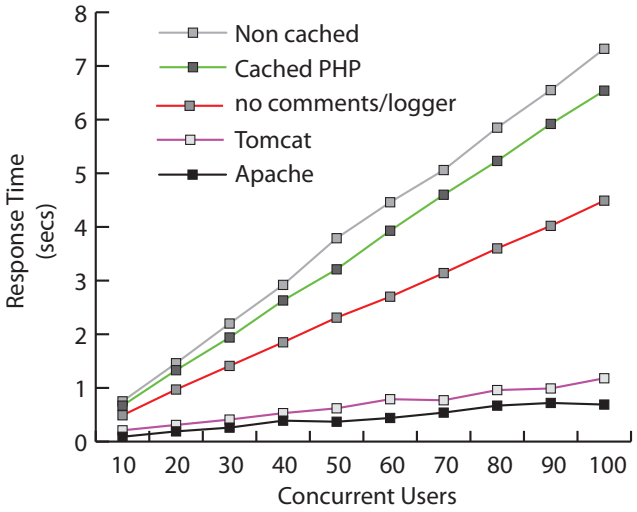


Figure 11.4: No cache with Uncommented PHP and no Logging

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.

11.3.2 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:

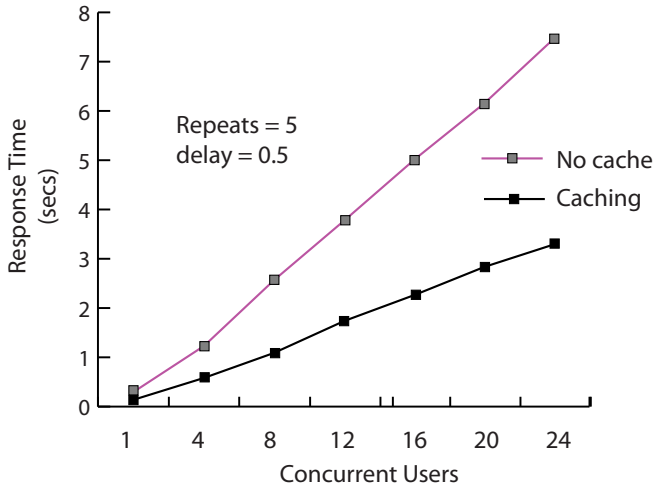


Figure 11.5: Response time, caching v. non-caching

11.3.3 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.

Chapter 12

Caching Thoughts

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 this 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.

12.1 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.

12.2 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.

12.3 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 dependent on these types of external queries obviously cannot be cached suitably as they are outside control.

12.4 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

Finally, 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.

Chapter 13

SQL Example

The following describes a worked example of using a SQL database with authorised pages. It shows how to construct forms to enter and remove data, and display that data. The example is based on a directory structure:

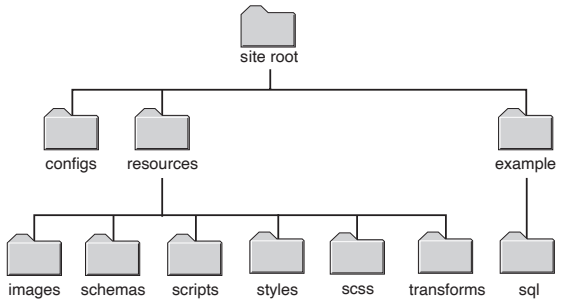


Figure 13.1:

13.1 The Database

The SQL database is a simple table of composers and their dates similar to the one I used in the SQL How-to:

```
mysql> use composers
Database changed
mysql> describe composer;
+------------+------------------+------+-----+---------+----------------+
| Field      | Type             | Null | Key | Default | Extra          |
+------------+------------------+------+-----+---------+----------------+
| id         | int(10) unsigned | NO   | PRI | NULL    | auto_increment |
| name       | varchar(40)      | NO   |     |         |                |
| forenames  | varchar(40)      | YES  |     | NULL    |                |
| birth      | date             | YES  |     | NULL    |                |
| death      | date             | YES  |     | NULL    |                |
+------------+------------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)

mysql> select * from composer;
+-------------+------------------+------------+------------+
| id | name   | forenames        | birth      | death      |
+-------------+------------------+------------+------------+
| 16 | Dvořák | Antonin          | 1841-09-08 | 1904-05-01 |
| 15 | Mozart | Wolfgang Amadeus | 1756-01-27 | 1791-12-05 |
| 13 | Barber | Samuel           | 1910-03-09 | 1981-01-23 |
+-------------+------------------+------------+------------+
3 rows in set (0.01 sec)

mysql>
```

Note that the character set used for the two text fields is UTF-8 UniCode, which allows the correct representation of names such as “Dvořàk”. The Paloose pipes use this encoding; the output serializers output according to the ‘`encoding`’ parameter.

13.1.1 The Index Page

The index page for the example is a simple list:

```
 <?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"
   xmlns:list="http://www.hsfr.org.uk/Schema/List"
   xmlns:pf="http://www.paloose.org/schemas/Forms/1.0"
   xmlns:link="http://www.hsfr.org.uk/Schema/Link"
   xmlns:paloose="http://www.paloose.org/schemas/Paloose/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 Example</t:heading>

      <t:p>This example allows you to set up and query a database. They are
         all working with a data base "composers".</t:p>

      <list:list type="unordered">
         <list:item><link:link type="uri" ref="addComposer.html">Create
             composer</link:link></list:item>
         <list:item><link:link type="uri" ref="deleteComposer.html">Delete
             composer</link:link></list:item>
      </list:list>

      <list:list type="unordered">
         <list:item><link:link type="uri" ref="displayAllComposers.html">Display
             entire database</link:link></list:item>
      </list:list>

      <t:p><link:link type="uri" ref="logout.html">Logout</link:link> from
             admin pages.</t:p>

   </page:content>
</page:page>
```

13.2 The Sitemap

The sitemap to read this file is fairly simple at this level. First the necessary components (it is worth noting that some of these declarations can be placed in the calling sitemap when this is a subsitemap. I have elected to turn off the caching here.

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

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

The only resource here is a means of outputting the final assembled XML page.

```
<map:resources>

   <map:resource name="outputPage">
      <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="xhtml"/>
   </map:resource>

</map:resources>
```

When I worked the original example I declared a series of views to help with the debugging.

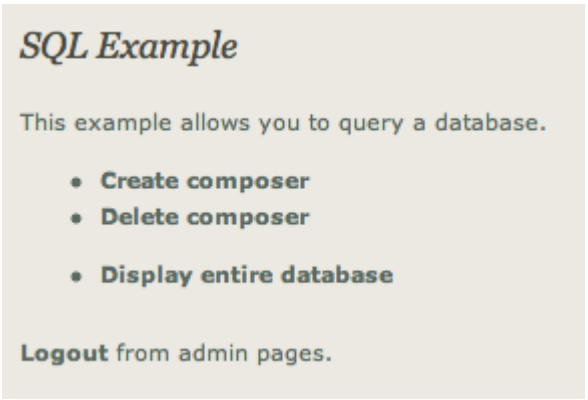
```
   <map:views>

      <map:view name="aggr" from-label="aggr-content">
         <map:call resource="outputXML"/>
      </map:view>

      <map:view name="xml" from-label="xml-content">
         <map:call resource="outputXML"/>
      </map:view>

      <map:view name="px" from-label="px-content">
         <map:call resource="outputXML"/>
      </map:view>

      <map:view name="transform" from-label="page-transform">
         <map:call resource="outputXML"/>
      </map:view>

      <map:view name="menus" from-label="menus-content">
         <map:call resource="outputXML"/>
      </map:view>

      <map:view name="pforms-1" from-label="pforms-violations">
         <map:call resource="outputXML"/>
      </map:view>

      <map:view name="pforms-2" from-label="pforms-default">
```

```
         <map:call resource="outputXML"/>
      </map:view>

      <map:view name="pforms-3" from-label="pforms-html">
         <map:call resource="outputXML"/>
      </map:view>

      <map:view name="sql" from-label="sql-content">
         <map:call resource="outputXML"/>
      </map:view>

      <map:view name="sql-transform" from-label="sql-transform">
         <map:call resource="outputXML"/>
      </map:view>

</map:views>
```

The first page has a very simple pipeline:

```
      <map:pipelines>

         <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:pipeline>

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

            <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:pipeline>

      </map:pipelines>

   </map:sitemap>
```

Using the same styles as the Paloose site the above would appear as:



13.3 Displaying All Entries

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

```
<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.

13.3.1 Extending the Sitemap

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

```
<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>
```

13.3.2 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>
```

13.3.3 Processing Data

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

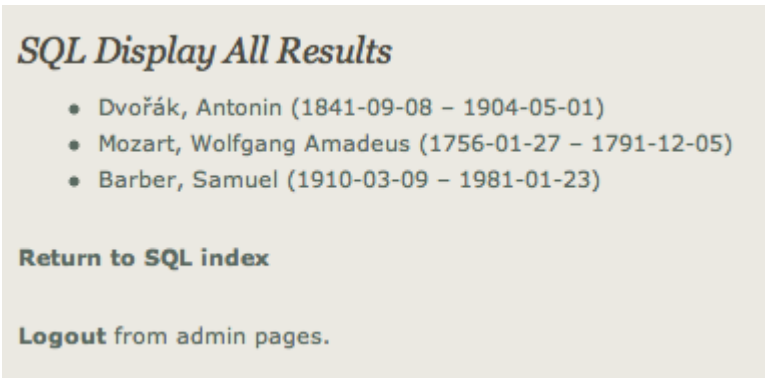
```
<?xml version="1.0" encoding="UTF-8"?>

<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:



13.4 Adding Entries

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:

```
<?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>
```

13.4.1 The Flow Script

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

```
<?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
?>
```

13.4.2 Extending the Sitemap

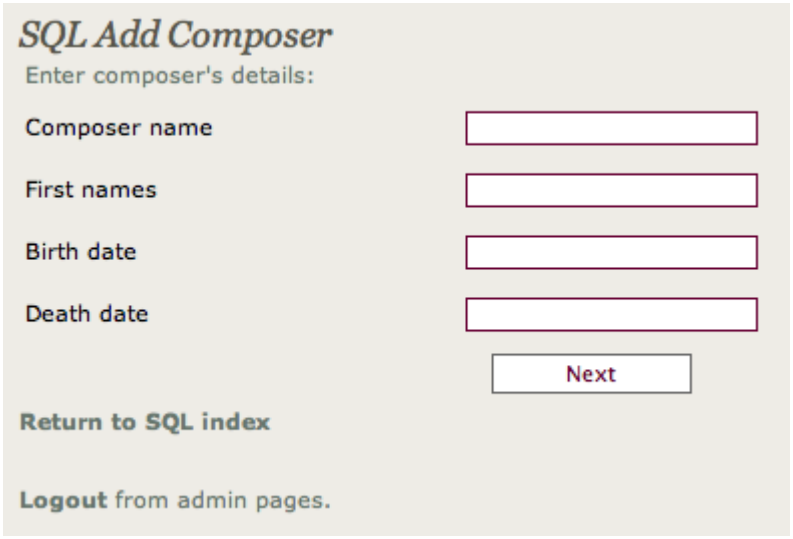
The sitemap must declare the script:

```
<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:

```
<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:



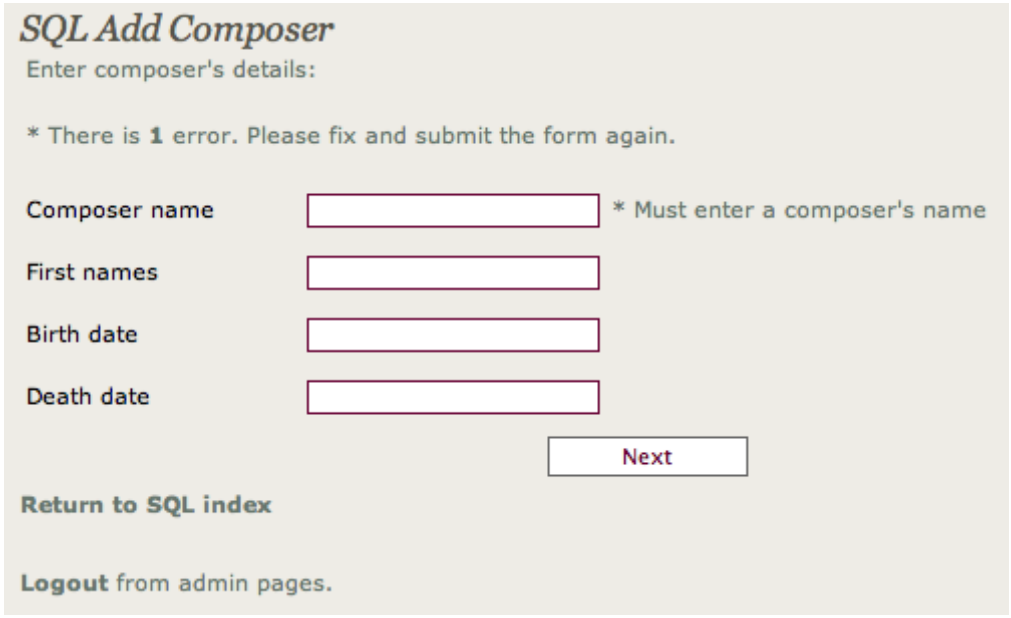
13.4.3 Confirming the Entered Data

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:

```
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:



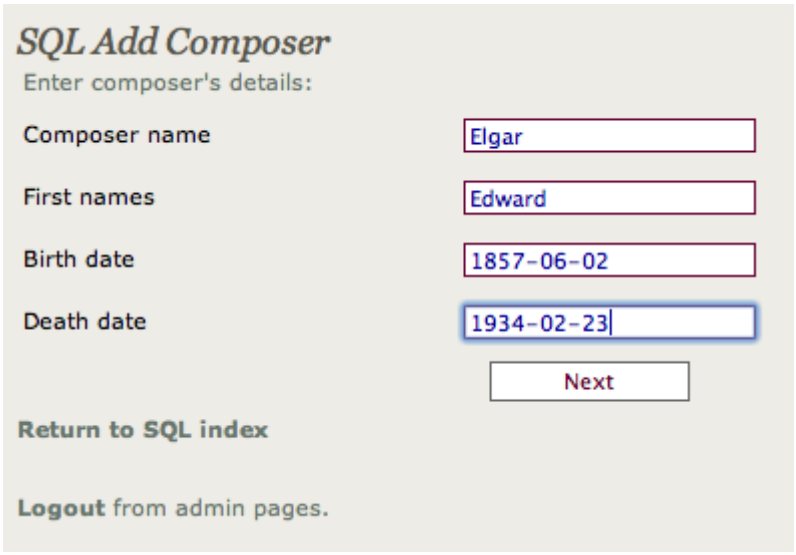
13.4.4 Confirm Add Entry Form

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:

```
<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:

13.4.5 Writing the Data

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

```
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;

         }
      }
   }
```

13.4.6 Constructing the SQL Command

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

```
   <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.

13.4.7 Extending the Sitemap

A suitable addition is required to the sitemap:

```
<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>
```

13.4.8 Processing Result

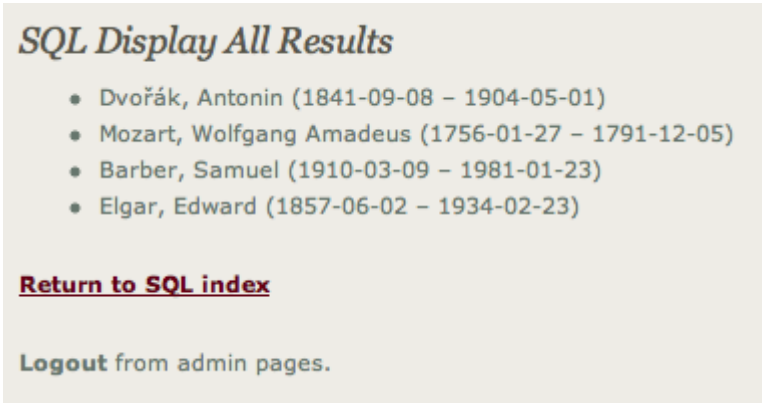
Note that we do not need the PForm transforms in this matcher. Only a transformer that contains the necessary transform to process the returned confirmation data from the SQL server about the success or faiulure of the write data command is required.

```
<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 successful write would appear as:

Displaying the data now gives:



13.5 Deleting Entries

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:

```
<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>
```

13.5.1 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:

```
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()
```

13.5.2 Extending the Sitemap

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

```
<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 below.

13.5.3 Processing Returned Data

```
<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:

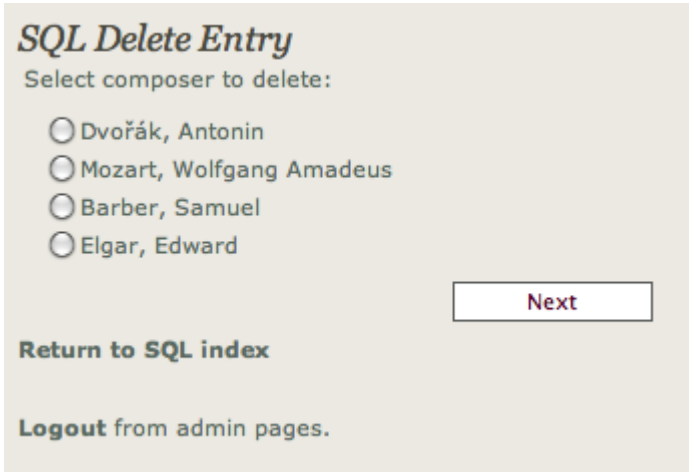
```
<default:row-set xmlns="http://apache.org/cocoon/SQL/2.0"
                 nrofrows="4"
                 name="selectDeleteComposer">
   <default:row>
      <default:id>16</default:id>
      <default:name>DvoÅŹÃąk</default:name>
```

```
      <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>DvoÅŽÃąk, Antonin</pf:label>
         <pf:value>DvoÅŽÃąk</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:



13.5.4 Confirm Data Form

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

```
<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.

13.5.5 Extending the Flow Script

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

```
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:

13.5.6 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.

```
<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>
```

13.5.7 Extending the Flow Script

The flow script is modified:

```
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()
```

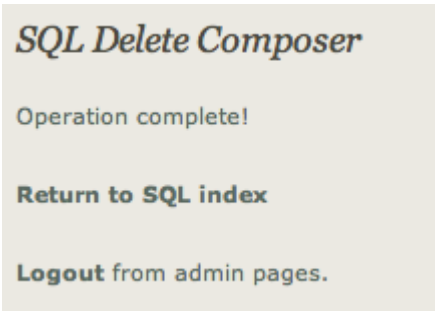
13.5.8 Extending the Sitemap

And the sitemap is modified:

```
<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:

```
<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>
```

```
<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:

```
<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:

```
<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:

```
<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:

```
<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:

```
<?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`':

```
<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:

```
<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:

```
<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 accessible pages need be protected in the sitemap, all "*internal-only*" pipelines are protected naturally. More information on authorising pages can be found in Section 9.4 giving an example on authorising.

Chapter 14

Gallery Transformer Example

Consider the following sitemap definition to implement a gallery:

```
<map:pipelines>

    <map:pipeline>

       <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:/{1}.xml" element="content" strip-root="true"/>
          </map:aggregate>
          <map:transform src="context://resources/transforms/page2xhtml.xsl"
                         label="page-transform">
             <map:parameter name="page" value="{1}"/>
          </map:transform>
          <map:serialize type="xhtml"/>
       </map:match>

    </map:pipeline>

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

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

       <map:match pattern="gallery.xml">
          <map:generate src="context://content/gallery.xml" label="xml-content"/>
          <map:transform type="gallery" label="gallery-transform">
             <map:parameter name="root" value="context://gallery/gallery-1"/>
          </map:transform>
          <map:transform src="context://resources/transforms/gallery2html.xsl"
                         label="page-transform"/>
          <map:serialize/>
       </map:match>

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

    </map:pipeline>

</map:pipelines>
```

14.1 Defining the Gallery Index File

Within each folder of your gallery you must place a file that describes that level of the gallery. The default file name for this is '`gallery.xml`'. If you wish to change this (globally for all gallery folders) then change the entry in the '`paloose.php`' that you have in the root of your web site. There is a constant '`GALLERY_INDEX`' defined here in the line

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

Change this line to suit your site.

The contents of this file will be melded into your XML file as it is processed in the pipeline. The format of this file is relatively straight forward. For example (from a local stables site):

```
<paloose:gallery xmlns:paloose="http://www.paloose.org/schemas/Paloose/1.0"
      xmlns:t="http://www.hsfr.org.uk/Schema/Text"
      xmlns:link="http://www.hsfr.org.uk/Schema/Link">
   <paloose:name>Photos</paloose:name>
   <paloose:dir></paloose:dir>
   <paloose:breadcrumb>
      <paloose:name src="">GPF</paloose:name>
   </paloose:breadcrumb>
   <paloose:description>
      <t:p>
         We will add galleries as we get them. If you have any to put up please email them to
         <link:link type="email" ref="xxx@xxx.xxx"/>. Any format is acceptable, but
         please do not process them before you send them. If they are too many or too big
         then let us have them on CDROM.</t:p>
   </paloose:description>
   <paloose:folders>
      <paloose:folder src="general">General Views around the farm (lessons etc)</paloose:folder>
      <paloose:folder src="RDA">Some of our RDA activities</paloose:folder>
      <paloose:folder src="racing">Our racing successes!</paloose:folder>
      <paloose:folder src="horses">Some of our horses</paloose:folder>
   </paloose:folders>
   <paloose:images type="multi"/>
</paloose:gallery>
```

Note

For correct running of the gallery transformer (since version 1.1.1b1), it is important that the Paloose namespace '`(http://www.paloose.org/schemas/Paloose/1.0)`' is used.

There will eventually be a schema defining this to make it easier to understand the above and to assist in creating them. However a quick look will show that it is relatively simple in format. The description information can be any element form, dependent on your following '`gallery2html`' transformer. Image information is displayed in a similar fashion to the above as:

```
<paloose:gallery>
   <paloose:name>Farm Views</paloose:name>
   <paloose:dir>general/farm</paloose:dir>
   <paloose:breadcrumb>
      <paloose:name src="">GPF</paloose:name>
      <paloose:name src="general">General</paloose:name>
      <paloose:name src="general/farm">The Farm</paloose:name>
   </paloose:breadcrumb>
   <paloose:description>
      <t:p>Some selected views around the farm. As you can see the countryside
         around here is beautiful with even a hack for an hour staying close to the
      farm or its immediate surrounds.</t:p>
   </paloose:description>
```

```
   <paloose:folders/>
   <paloose:images type="multi">
      <paloose:image name="gpf-1.jpg"></paloose:image>
      <paloose:image name="gpf-2.jpg"></paloose:image>
      <paloose:image name="gpf-3.jpg"></paloose:image>
      <paloose:image name="gpf-4.jpg"></paloose:image>
   </paloose:images>
</paloose:gallery>
```

Note that the folder in this case is null. It is perfectly possible to mix folders and images.

14.2 Adding the Gallery to your XML Content

In order to use the transformer a single element is placed in your content file that you wish to place the gallery. For example:

```
<page:content xmlns:page="http://www.hsfr.org.uk/Schema/Page">
   <paloose:gallery xmlns:paloose="http://www.paloose.org/schemas/Paloose/1.0"/>
</page:content>
```

Note that the *page:content* is not obligatory on your site. You could embed the gallery line within your own XML. That is just how it is used on one of my existing sites. We can use the attribute *root* to override the parameter in the component declaration above.

14.3 Processing the Transformer Output

After running through the transformer this element is replaced by the contents of the '`gallery.xml`' file. How this information is processed is really up to the user, but the example of the Guinness Park Farm site would be instructive. We need two files: a transformer and a style file. The transformer takes the above XML and turns it into HTML suitable for display using the style file. First the declarations (mainly namespaces, which are used in the GPF site. Another site could use something else for the content structure. The only proviso is that they have to be the same, and that there should be a namespace declaration for Paloose.

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

<xsl:stylesheet
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:graphic="http://www.hsfr.org.uk/Schema/Graphic"
   xmlns:list="http://www.hsfr.org.uk/Schema/List"
   xmlns:link="http://www.hsfr.org.uk/Schema/Link"
   xmlns:text="http://www.hsfr.org.uk/Schema/Text"
   xmlns:news="http://www.hsfr.org.uk/Schema/News"
   xmlns:paloose="http://www.paloose.org/schemas/Paloose/1.0"
   xmlns:page="http://www.hsfr.org.uk/Schema/Page"
   version="1.0">
```

The following is an included XSL file processing the text (all coming from the namespaces above. Other sites would change this if required.

```
<xsl:include href="text2xhtml.xsl"/>
```

I enclose the whole gallery in a simple panel:

```
<xsl:template match="//paloose:gallery">
   <xsl:element name="div">
      <xsl:attribute name="class">galleryPanel</xsl:attribute>
```

```
      <xsl:apply-templates/>
   </xsl:element>
      </xsl:template>
```

The name of the gallery sits within a simple panel:

```
<xsl:template match="paloose:name">
   <xsl:element name="div">
      <xsl:attribute name="class">galleryNamePanel</xsl:attribute>
      <xsl:value-of select="."/>
   </xsl:element>
      </xsl:template>
```

We can safely eat the directory information that is the relative directory underneath the root for the displayed gallery folder.

```
<xsl:template match="paloose:dir"/>
```

The breadcrumb trail is assembled:

```
<xsl:template match="paloose:breadcrumb">
      <xsl:element name="div">
         <xsl:attribute name="class">galleryBreadcrumbPanel</xsl:attribute>
         <xsl:for-each select="paloose:name">
            <xsl:element name="a">
               <xsl:attribute name="href">
                  <xsl:choose>
                     <xsl:when test="@src = ''" >
                        <xsl:value-of select="'gallery.html'"/>
                     </xsl:when>
                     <xsl:otherwise>
                        <xsl:value-of select="concat( 'gallery.html?src=', @src )"/>
                     </xsl:otherwise>
                  </xsl:choose>
               </xsl:attribute>
               <xsl:value-of select="."/>
            </xsl:element>
            <xsl:if test="not( position() = last() )">
               <xsl:text> &gt;&gt; &lt;/xsl:text>
            </xsl:if>
         </xsl:for-each>
      </xsl:element>
   </xsl:template>
```

The description of the gallery folder sits within a simple panel. The text is processed by the included text templates included at the beginning (change to suit site):

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

Process the folders in to a simple vertical list (I use a small graphic to the left of the gallery title):

```
<xsl:template match="paloose:folders">
   <xsl:if test="paloose:folder">
      <xsl:element name="div">
         <xsl:attribute name="class">galleryFoldersPanel</xsl:attribute>
         <xsl:for-each select="paloose:folder">
            <xsl:element name="div">
               <xsl:attribute name="class">galleryFolderPanel</xsl:attribute>
               <xsl:element name="a">
                  <xsl:attribute name="href">
                     <xsl:choose>
```

```
                        <xsl:when test="@src = ''" >
                           <xsl:value-of select="'gallery.html'"/>
                        </xsl:when>
                        <xsl:otherwise>
                           <xsl:value-of select="concat( 'gallery.html?src=', @src )"/>
                        </xsl:otherwise>
                     </xsl:choose>
                  </xsl:attribute>
                  <table>
                     <tr>
                        <td><xsl:element name="img">
                           <xsl:attribute name="src">AlbumIcon.png</xsl:attribute>
                        </xsl:element></td>
                        <td><xsl:value-of select="."/></td>
                     </tr>
                  </table>
               </xsl:element>
            </xsl:element>
         </xsl:for-each>
      </xsl:element>
   </xsl:if>
      </xsl:template>
```

The images are shown in a set of image tags containing the main image, the thumbnail image, the cache directory where they are stored, and the description (plain string at present).

```
<xsl:template match="paloose:images">
   <xsl:choose>
      <xsl:when test="@type='multi'">
         <!-- Displaying a set of thumbnails -->
         <xsl:if test="paloose:image">
            <xsl:element name="div">
               <xsl:attribute name="class">galleryImagesPanel</xsl:attribute>
               <xsl:for-each select="paloose:image">
                  <xsl:call-template name="outputImageAndLink">
                     <xsl:with-param name="inNode">
                        <xsl:value-of select="."/>
                     </xsl:with-param>
                  </xsl:call-template>
               </xsl:for-each>
            </xsl:element>
         </xsl:if>
      </xsl:when>
      <xsl:otherwise>
         <!-- Displaying single image -->
         <xsl:variable name="thisImageName"><xsl:value-of select="@name"/></xsl:variable>
         <xsl:element name="div">
            <xsl:attribute name="class">galleryImagePanel</xsl:attribute>
            <xsl:call-template name="outputImage">
               <xsl:with-param name="inImage">
                  <xsl:value-of select="//paloose:image[ @name = $thisImageName ]/@img"/>
               </xsl:with-param>
               <xsl:with-param name="inNextImageName">
                  <xsl:value-of
            select="//paloose:image[ @name = $thisImageName ]/following-sibling::*[1]/@name"/>
               </xsl:with-param>
               <xsl:with-param name="inPrevImageName">
                  <xsl:value-of
            select="//paloose:image[ @name = $thisImageName ]/preceding-sibling::*[1]/@name"/>
               </xsl:with-param>
               <xsl:with-param name="inImageDescription">
                  <xsl:value-of select="//paloose:image[ @name = $thisImageName ]"/>
               </xsl:with-param>
               <xsl:with-param name="inGalleryDir">
                  <xsl:value-of select="//paloose:dir"/>
               </xsl:with-param>
            </xsl:call-template>
         </xsl:element>
      </xsl:otherwise>
   </xsl:choose>
```

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

Outputting a single image:

```
<xsl:template name="outputImage">
   <xsl:param name="inImage"/>
   <xsl:param name="inNextImageName"/>
   <xsl:param name="inPrevImageName"/>
   <xsl:param name="inImageDescription"/>
   <xsl:param name="inGalleryDir"/>

   <xsl:variable name="fullImage">
      <xsl:value-of select="concat( 'resources/images/cache/', $inImage )"/>
   </xsl:variable>

   <xsl:element name="div">
      <xsl:attribute name="class">galleryPrevNextPanel</xsl:attribute>
      <xsl:element name="div">
         <xsl:attribute name="class">galleryPrevButtonPanel</xsl:attribute>
         <xsl:choose>
            <xsl:when test=" $inPrevImageName != ''">
               <xsl:element name="a">
                  <xsl:attribute name="href">
                     <xsl:value-of select="concat( 'gallery.html?name=', $inPrevImageName,
                        '&src=', $inGalleryDir )"/>
                  </xsl:attribute>
                  <xsl:value-of select="'<<<'"/>
               </xsl:element>
            </xsl:when>
            <xsl:otherwise>
               <xsl:value-of select="' '"/>
            </xsl:otherwise>
         </xsl:choose>
      </xsl:element>
      <xsl:element name="div">
         <xsl:attribute name="class">galleryNextButtonPanel</xsl:attribute>
         <xsl:choose>
            <xsl:when test=" $inNextImageName != ''">
               <xsl:element name="a">
                  <xsl:attribute name="href">
                     <xsl:value-of select="concat( 'gallery.html?name=', $inNextImageName,
                        '&src=', $inGalleryDir )"/>
                  </xsl:attribute>
                  <xsl:value-of select="'>>>'"/>
               </xsl:element>

            </xsl:when>
            <xsl:otherwise>
               <xsl:value-of select="' '"/>
            </xsl:otherwise>
         </xsl:choose>

      </xsl:element>
      </xsl:element>

      <xsl:element name="div">
         <xsl:attribute name="class">galleryImageDescriptionPanel</xsl:attribute>
         <xsl:value-of select="$inImageDescription"/>
      </xsl:element>

      <xsl:element name="div">
         <xsl:attribute name="class">galleryImagePanel</xsl:attribute>
         <xsl:element name="img">
            <xsl:attribute name="src"><xsl:value-of select="$fullImage"/></xsl:attribute>
         </xsl:element>
      </xsl:element>
   </xsl:template>
```

Outputting a gallery of images:

```
   <xsl:template name="outputImageAndLink">
      <xsl:param name="inNode"/>
      <xsl:variable name="thumbnailImage"><xsl:value-of select="@thumb"/></xsl:variable>
      <xsl:variable name="imageName"><xsl:value-of select="@name"/></xsl:variable>
      <xsl:variable name="cachedImage"><xsl:value-of select="@img"/></xsl:variable>
      <xsl:variable name="fullThumbnailImage">
         <xsl:value-of select="concat( 'resources/images/cache/', $thumbnailImage )"/>
      </xsl:variable>
      <xsl:variable name="galleryDir">
         <xsl:value-of select="//paloose:dir"/>
      </xsl:variable>
      <xsl:element name="span">
         <xsl:attribute name="class">galleryThumbnailImage</xsl:attribute>
         <xsl:element name="a">
            <xsl:attribute name="href">
               <xsl:value-of select="concat( 'gallery.html?name=', $imageName,
                  '&src=', $galleryDir )"/>
            </xsl:attribute>
            <xsl:element name="img">
               <xsl:attribute name="src">
                  <xsl:value-of select="$fullThumbnailImage"/>
               </xsl:attribute>
            </xsl:element>
         </xsl:element>
      </xsl:element>
   </xsl:template>
```

Finally pass everything not eaten above.

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

</xsl:stylesheet>
```

All the above looks a bit frightening, but it is really straight-forward. It can be changed to meet local requirements which is why it is not part of Paloose. The Paloose Gallery transformer only gets the information, it does not do anything with it for display — as should be the case with any XML/XSL engine like Cocoon or Paloose; separation of duties is the thing.

The only thing remaining is the CSS some of which is shown below (again taken from the old stables site):

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

   .galleryNamePanel {
      margin: 15px 0px 0px 0px;
      font-weight: bold;
      font-size: 120%;
   }

   .galleryBreadcrumbPanel {
      margin: 10px 10px 10px 10px;
      padding: 3px 3px 3px 3px;
      border: 1px dashed #414b37;
      color: black;
   }

   .galleryDescriptionPanel {
      font-family: Verdana, Arial, Helvetica, sans-serif;
      font-size: 100%;
      margin: 0px 0px 10px 0px;
```

```
}

.galleryFoldersPanel {
   padding: 10px 0px 10px 0px;
   border-top: 1px solid #414b37;
}

.galleryFolderPanel {
   padding: 5px 0px 0px 0px;
}

.galleryImagesPanel {
   text-align: center;
   border-top: 1px solid #414b37;
   margin: 0px 0px 10px 0px;
}

.galleryThumbnailImage img {
   margin: 10px 10px 0px 10px;
   border: 2px solid white;
}

.galleryImagePanel img {
   margin: 0px 0px 10px 0px;
   border: 2px solid white;
   text-align: center;
}

.galleryImageDescriptionPanel {
   margin: 0px 0px 10px 0px;
   padding: 0px 0px 0px 0px;
   text-align: center;
   font-style: italic;
   font-size: 110%;
   font-family: Georgia, Times, serif;
}

.galleryPrevNextPanel {
   margin: 15px 0px 0px 0px;
   width: 600px;
   height: 20px;
}

.galleryPrevButtonPanel {
   position:relative;
   text-align: left;
   width: 100px;
   top: 0px;
   font-weight: bold;
   font-size: 120%;
   float:left;
}

.galleryNextButtonPanel {
   position:relative;
   width: 100px;
   top: 0px;
   text-align: right;
   font-weight: bold;
   font-size: 120%;
   float:right;
}
```

OK, so I am not going to win prizes for the best way of doing things, but there we are, it works.

Chapter 15

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.

15.1 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.

It is also worth noting that if the transforms are relatively simple this can be incorporated into the WordPress or Joomla! code for a particular page directly rather than use Paloose — a more satisfactory approach (and faster).

Which systems are supported?

Currently Paloose has plugins for the following apps:

- WordPress
- Joomla!

15.2 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.

15.2.1 Install with no existing Paloose system

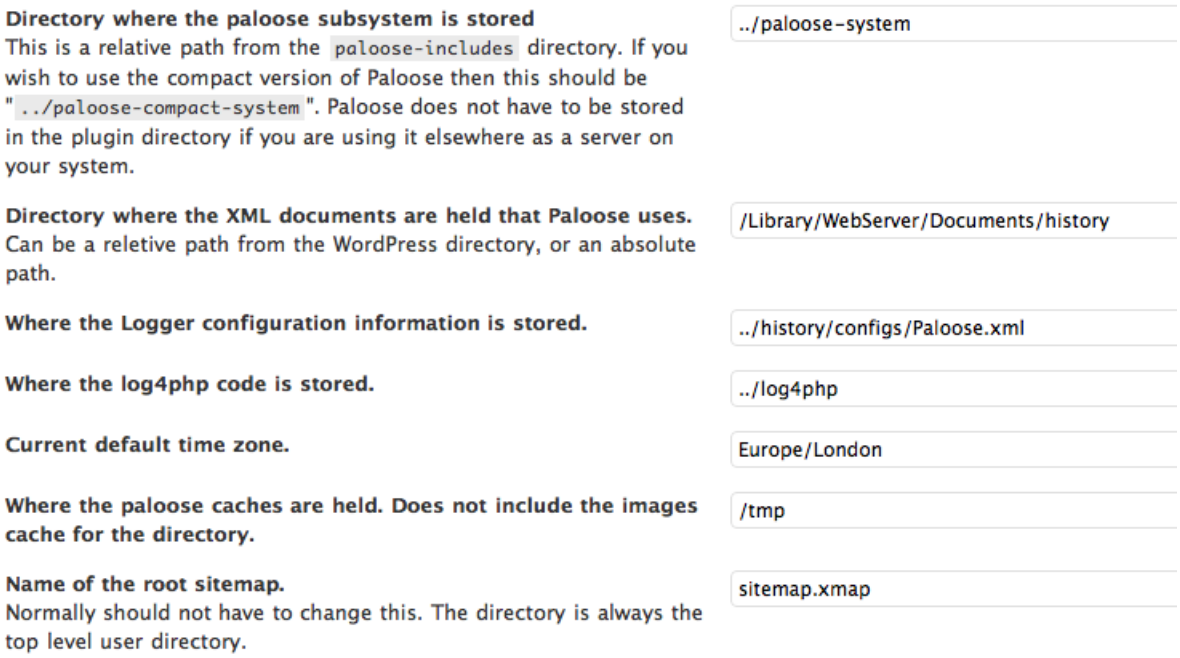
Download the latest version here. The download includes the latest Paloose and its compact version together with the latest Log4PHP that Paloose will work with. The directory structure of the plugin is:

```
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 [1]. 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:

[1] wordpress/wp-content/plugins



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.

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.

15.2.2 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.

15.3 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 ahown in Figure 15.1. 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):

```
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
```

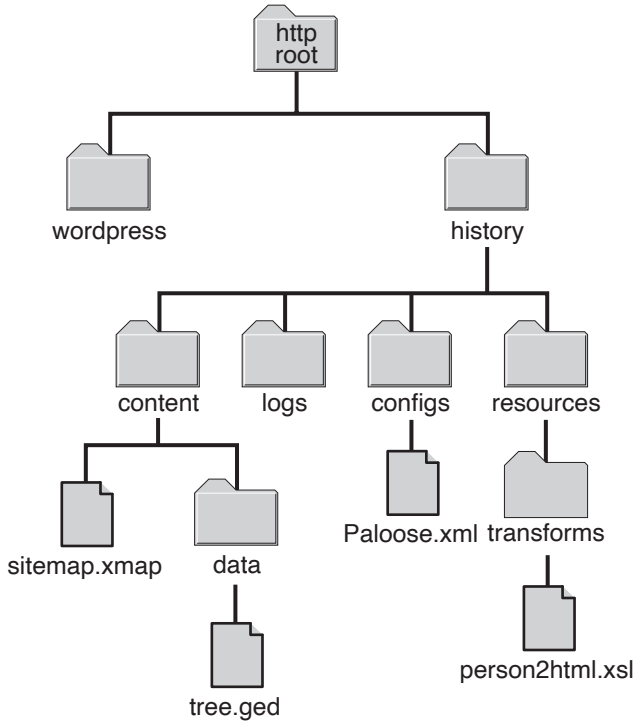


Figure 15.1: Example Folder Structure

```
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:

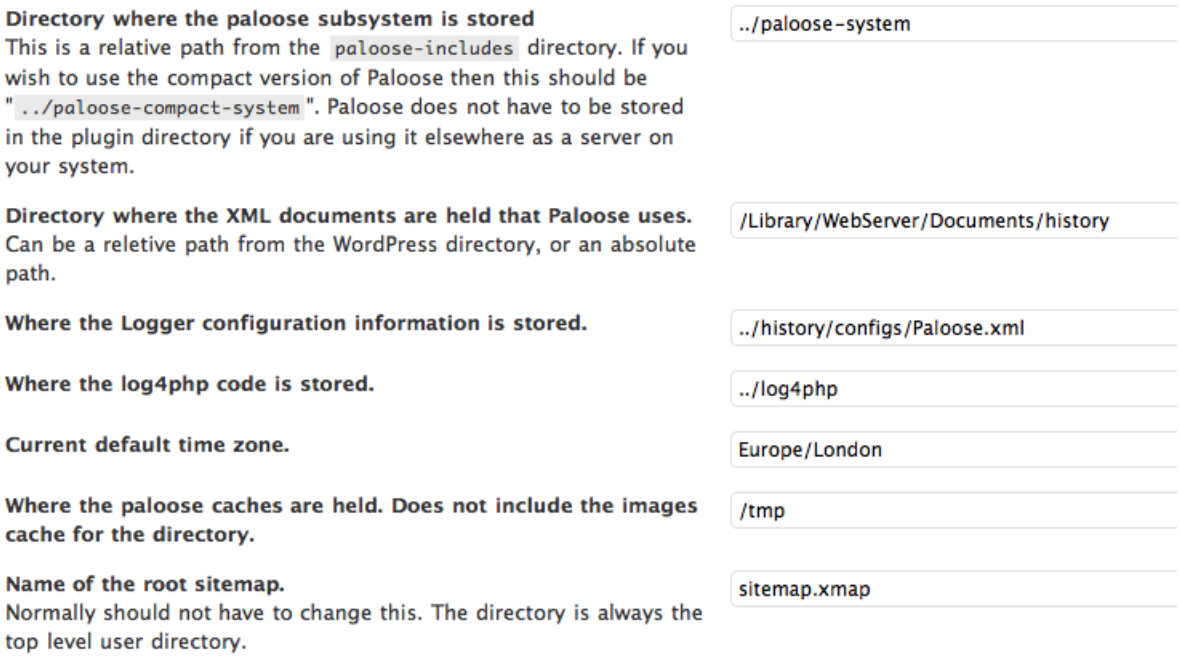
```
<g:indi id="@I12@">
   <g:name data="Frederick/FIELD-RICHARDS/"/>
   <g:titl data="Rev"/>
   <g:sex data="M"/>
   <g:occu data="Clergyman"/>
   <g:birt>
      <g:date data="2 NOV 1846"/>
      <g:plac data="Hackney,,,,,"/>
   </g:birt>
   <g:bapm>
      <g:date data="3 DEC 1865"/>
   </g:bapm>
```

```
      <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 template file is

```
<?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:

```
<?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`:

```
<?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:

```
<?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.

15.4 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

15.4.1 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.

Chapter 16

Troubleshooting

16.1 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.

16.2 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.

16.3 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
```

16.4 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.

Chapter 17

Frequently Asked Questions

17.1 What future for Paloose?

I use it for all my small sites and will continue to do so until Cocoon[1] is supported by the smaller ISPs that I use. It is also interesting (and slightly depressing) that the latest Cocoon (V2.2.0) does not seem to be as easy to use as previous Cocoon Versions. I used this earlier version on my local servers but did not upgrade because I found that versions after 2.2 have become significantly more complex for simple sites (using Maven[2], Spring[3] etc). So perhaps Paloose will fulfil the requirements of those who want a quick Cocoon-like engine for their small Web sites.

I am currently developing a CMS based on Paloose, but this remains very much a personal project and it is unlikely to be released in the near future. It is possible to edit site pages using the system but it is very early days.

17.2 What editor/IDE do you use to develop Paloose?

From the start of Paloose I have always used the oXygen XML editor and IDE[4], using it commercially before I retired and started Paloose. I had been involved with SGML and XML for many years and oXygen was always 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. I am hugely grateful to oXygen's developers Syncro Soft for the support they have given me while developing Paloose.

17.3 What Licence is Paloose issued under?

Paloose is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

[1] http://cocoon.apache.org
[2] http://maven.apache.org
[3] https://spring.io
[4] https://www.oxygenxml.com

Paloose is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

17.4 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 it is complete apart from the occasional pipeline component to add functionality. 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`'.

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. It is working on my local servers and I may update some of my personal sites to use it but that is someway off.

17.5 Do you intend to add PDF support via FOP?

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 basis for the LaTeX based system of the 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.

17.6 What systems will it run on?

Basically any system that will run PHP5. Currently the list is OS-X, Linux, and other similar Unix systems. I also know of a system based on Apache 2.0/PHP5 on Windows Server 2003.

17.7 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 family have more call on my time.

17.8 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 Section 11 in more detail.

17.9 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 Section 12 for more on this. It is not the end of the story, but more performance increases will probably come from elsewhere.

17.10 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 Section 10 very carefully.

17.11 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.

17.12 How do I port Paloose Sites to Cocoon?

This is relatively easy with the only major change being the sitemap components. But beware, some components do not exist in Cocoon and you would have to write your own in Java.

17.13 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.

17.14 How do I write Components for Paloose?

I have written an example component here (tgz) that is a "template" for writing generator, transformer and serialiser 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.

17.15 Do you have any support for iPhone etc?

Yes. The browser selectors (*BrowserSelector* and *MobileSelector*) can detect the various mobile systems to allow separate pipelines to be run.

17.16 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[5] or XML-Schema[6] is very difficult — they simply do not have a rich enough structure to do what I think is necessary without having to use Schematron[7] additions. I do have an experimental schema that uses RELAX-NG and Schematron that I occasionally tinker with but I suspect is not complete for all variations of possible sitemaps.

However I have produced a set of schemas for the current version of Paloose sitemaps. It has not been extensively tested, although I have run several sitemaps past it, including one CMS system based on Paloose, and it seems ok (so far). They are bundled with the current Paloose

[5] `http://www.relaxng.org/`
[6] `http://www.w3.org/XML/Schema`
[7] `https://www.schematron.com/`

distribution files on the download page. If you just want to see their structure then they are bundled with the Paloose distro as well. Remember that these are definitely work in progress so they provided "as is" for anyone to experiment with. They are in folders within Paloose, '`paloose/resources/schemas`'.

The closest that I got to producing one used the Document Structure Description 2.0 (DSD-2)[8]. 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, all of which can be found on the DSD-2 home page. However RDF is a major problem, it is impossible to write a universal validating XML schema for it — it ain't the right sort of grammar and the 2004 specification for RDF is flawed, but that is another story.

17.17 Technical Trivia

The code that builds Paloose is a little rough in places. This was 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 and PL/1, as well as some not so strongly typed such as Basic, Fortran, Perl, C and C++; and more obscure offerings such as XSL, OmniMark, ELLA, Abel and LaTeX. Latterly I have been using Swift and Objective-C. I had 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, however I am gradually refining the code when I have the time.

[8] `https://www.brics.dk/DSD/dsd2.html`

Chapter 18

Version History

0.0.0	27th March 2006	Created
0.0.1	10th April 2006	Major change to all to use log4php
0.0.2	19th April 2006	Change directory structure and start adding error/exceptions properly
0.1.0	24th May 2006	First issue (Hop Vine site and Paloose site both use it)
0.2.0	13th July 2006	Views mechanism sorted to work on more components
0.3.0	20th July 2006	i18n translation completely rewritten (handles dates (not time yet))
0.4.0	1st August 2006	PageHit and Gallery Transformer added
0.5.0	12th August 2006	Preliminary TidyGenerator added
0.5.1	13th August 2006	Changed base class for WildcardURIMatcherException
0.5.2	21st August 2006	Added XInclude in FileGenerator
0.6.0	12th September 2006	Added browser selector
0.6.1	4th October 2006	Date/time localisation changes
0.6.2	16th October 2006	Minor cosmetic changes
0.6.3	19th October 2006	Minor change to the way date is processed in I18nTransformer.
0.7.0	24th October 2006	Added RegexpURIMatcher. Cleaned up use of matchers. Changed WildcardURIException to be a more general matcher exception, MatcherException. Cleaned up comments that were misleading (still some more to sort out unfortunately).
0.8.0	26th October 2006	Updated HTMLSerializer Added XHTMLSerializer, TextSerializer, XMLSerializer.
0.9.0	2nd November 2006	Added Resources/Call mechanism. Cleaned up several areas of bad code and documentation. Cured several bugs in the error reporting. Made sure all pipe building components used same creation process in PipeElement class.
0.9.1	15th November 2006	Temporarily removed PipeElementInterface.
0.9.2	16th November 2006	Restored PipeElementInterface with suitable corrections to dependant classes.
0.9.2b1	17th November 2006	Strong PHP checking allowed for (and all subsequent betas at this level)
0.10.0	21st November 2006	Major work making the code more robust after turning on local server full PHP error reporting
0.10.1	22nd November 2006	Corrected uninitialised variable $badip in PageHitTransformer

0.11.0	5th December 2006	Added POST information into RequestParameterModule. Removed global variable for error messages (now passed through exceptions). Added SendMail action (Action, Actions, SendMailAction classes added). Cured bug in ErrorPage that occasionally gave errors in display. Many minor cosmetic changes. Major code documentation changes (still more to do).
0.11.1	5th December 2006	global $gConfiguration missing in TidyGenerator.
0.12.0	10th December 2006	Converted to use output buffer Added redirect-to pipeline element More code documentation changes Added internal-only pipelines
0.13.0	1st December 2006 ff	Major clean up of documentation and code
0.14.0	11th January 2007	Adding authorisation/session/continuations mechanism (not complete)
0.14.1	16th January 2007	Cured minor bug in ResourceReader (unset variable). Cured bug in Mount where sitemap variables were not expanded. Cured bug in I18nTransformer (wrong check, unset variable in time evaluate).
0.15.0	12th February 2007	Changed the way subsitemap declarations work. You can now declare components within appropriate subsitemap, not always put them in the master sitemap. Some minor documentation changes.
1.0.0b1	26th March 2007	Added PXTemplateGen generator for expanding embedded sitemap variables. Added SourceWriteTransformer for diverting XML to external file (cf Cocoon). Added PasswordTransformer for encrypting passwords in Authorisation framework. Added Authorisation actions: AuthAction, AuthenticationActions, LoginAction and LogoutAction. Added Paloose Form framework. Added Paloose Continuations/Flow framework. Cured problem where sitemap pseudo-protocols parameters were not being expanded. More minor code documentation updates.
1.0.0b2	27th March 2007	Added LogTransformer for pipeline logging.
1.0.0b3	30th March 2007	Created new class hierarchy for the pipeline elements. Only of note if you have created your own pipeline components. The inheritance is now from an intermediate class not the PipeElement class. For example TransformerPipeElement for all transformers. This affects generators, actions, transformers, selectors, readers, and serializers. Cured bug in Readers that would cause exception on duplicate Reader name in subsitemaps, rather than redefine.
1.0.0b4	4th April 2007	Cured bug in GalleryTransformer.
1.0.0b5	23rd April 2007	Made error reporting a little cleaner for XML/XSLT problems (not well formed etc). Minor cosmetics.
1.0.0b6	24th April 2007	Corrected some untidiness in the base classes for components (Transformers, Generators etc).
1.0.0b7	2nd May 2007	Added DirectoryGenerator.
	29th July 2007	Better error reporting in WildcardURIMatcher

1.0.1	11th August 2007	Checking for null description in GalleryTransformer
1.0.3	15th November 2007	Added RequestParameterSelector Cured bug in _BrowserSelector (wrong name for getBrowserName method)
1.0.4	18th December 2007	Added support for ImageMagick if no PHP5 support for jpeg, png, gif.
1.0.5	25th December 2007	Added better error reporting in GalleryTransformer when image file does not exist.
1.0.6	13th May 2008	Added better browser checking Added iPhone support
1.0.7	20th May 2008	Cured toString() in PipeElement producing malformed XML Cured problem in SelectorPipeElement with nested select constructions.
1.1.0	3rd June 2008	Cured discrepancy between Cocoon and Paloose in call statements.
1.1.1	14th June 2008	Cured bug in Aggregator stopping '`context://`' pseudo-protocols from working. Cured namespace problem in Galleries (must have namespace in gallery.xml)
1.1.1b1	25th August 2008	Minor cosmetics and included (undocumented) Gedcom Generator support. Cured bug in Aggregator stopping '`context://`' pseudo-protocols from working. Cured namespace problem in Galleries (must have namespace in gallery.xml)
1.1.1b2	5th October 2008	Cured bug where ImageMagick files had fixed bin path
1.1.1b3	6th October 2008	Tidied up I18nTransformer where I18n namespace was being used on attributes Improved I18nTransformer default time.date patterns
1.1.2	15th October 2008	Release GedCom support
1.1.2b1	26th November 2008	Removed Log4PHP from Paloose distribution
1.2.0	20th December 2008	Added FilterTransformer Added SQLTransformer Added ParameterModule Added GlobalModule Numerous minor updates Add documentation and correct many mistakes within documentation
1.2.0b1	21st December 2008	Cleaned up documentation for new Classes and removed spurious methods Changed error numbering
1.3.0	4th January 2009	Added caching system (TRAXTransformer, Aggregator, GedComGenerator and FileGenerator) Changed Exception system constants to PalooseException only Cured append bug in LogTransformer Improved error reporting in Match.php Cured RequestParameter bug that prevent store of query parameters Removed parameter detection into PipeElement base class Removed label detection into PipeElement base class Additional PHPDocumentation comments Added file locking in page hit code — should have been there on day 1 :-(General code cleaning

1.3.0b1	14th January 2009	Cured bug in GedComGenerator that prevented cache file being accessed correctly
1.3.0b2	22nd January 2009	Cured bug in PageHitTransformer that resulted from locking
1.3.0b3	5th February 2009	Cured bug in FileGenerator, PXTemplateGenerator and GedComGenerator with undefined variable cured bug in TraxTransformer with undefined variables
1.3.0b4	11th February 2009	Sorted out several uninitialised variables
1.3.1	18th February 2009	Added XIncludeTransformer Removed xinclude stuff from the generators (use a tranformer now)
1.3.1b1	24th February 2009	Cured problem of namespaces in XHTMLSerializer
1.3.2	5th March 2009	Added EntityTransformer (only very beta at this stage)
1.3.3	24th May 2009	Added removal of Paloose attributes from the HTML/XHTML serializers (__xxxxx) Corrected Reader error-number bug Added GALLERY_ERROR Added some more logging lines in Pipeline.php Created ErrorModule.php and extended HandleError.php to handle errors better
1.3.4	5th June 2009	Added entity translation in generators (File and PXTemplate) Added regular expression variables to global module
1.3.4b1	15th July 2009	Cleaned up XHTMLSerializer
1.3.4b2	22nd July 2009	Changed entity translation to ignore “lt”, “gt”, “amp” and “quote” which are the standard HTML supported entities.
1.3.4b3	5th October 2009	Added time zone setting in setup. Cures a bug found when porting to Mac OS X Snow Leopard Changed eregi() to preg_match etc in Browser.php
1.3.4b4	1st December 2009	Changed missed eregi() to preg_match etc in Browser.php Updated documentation which was wrong for flows (extended it as well)
1.3.5b5	12th January 2010	Changed GalleryTransformer etc to allow for richer text in image/folder descriptions Updated SQLTransformer to handle entities correctly Updated SQLTransformer to handle query return values better (existing users check the sql:query attributes change) Added is_utf8 function in Utilities Added utf8ToUnicodeEntities function in Utilities Rewrote the variable expander inStringResolver to handle expansion more efficiently (and correctly) Added makeStringHex function in Utilities (debugging aid) Updated logging to the latest log4php V2.0.0 Changed Request Parameter Handling to ensure internal UTF format is used Cured bug in RequestParameterModule that did not encode arrays properly as UTF-8 Cured bug in LogTransformer where too much logging was enabled (info should have been debug) Rewrote the string expander in StringResolver.php and to handle expansion more efficiently (and correctly)

1.3.6	2nd August 2010	Added HTML meta data to record version etc in XHTMLSerializer and HTMLSerializer
1.3.6b1	6th September 2010	Cured bug in Browser.php in regular expression handler for Opera browser detection
1.4.0b2	24th December 2010	Removed all globals where possible into Environment class Removed spurious commented out code (finally) Added PALOOSE_SESSIONS_REQUIRED and PALOOSE_CLIENT in config.inc.php Added sessionsRequired and client in config.inc.php Added necessary code to use above for Paloose to act as client to WordPress server Huge change to naming standard to ensure no clashes (as far as possible) with calling systems such as Wordpress
1.4.0b3	24th December 2010	Added client output buffer to Environment and associated code Cleaned up exit from Paloose
1.4.0b4	1st January 2011	Refined the root directory variables etc. to ensure multi-site working in WordPress
1.4.0b5	12th January 2011	Added clean-up code on exit (though needs more)
1.5.0b1	13th January 2011	Major change to way Paloose is invoked (now through class wrapper) Removed config.inc.php file and incorporated code into Paloose class Change the initiation files for each Paloose web site (server)
1.5.0b2	22nd January 2011	Cured bug that stopped request parameters being recognised
1.5.0b3	10th February 2011	Removed spurious debug output in SourceWritingTransformer
1.5.0b4	13th February 2011	Updated PasswordTransformer to handle null passwords Updated variable names in SourceWritingTransformer to make unambiguous meaning clear
1.5.1b25	12th April 2011	Updated Error routines Added VariableTransformer to convert embedded Paloose variables in the text
1.5.2	3rd August 2011	Added lib/environment/Cookie.php Added lib/environment/CookiesModule.php Added lib/acting/CookiesAction.php Added lib/selection/RegexpSelector.php Added lib/selection/VariableSelector.php Added lib/transforming/ModuleWriteTransformer.php Added cookies support in lib/environment/Environment.php Added cookies support in lib/serialization/HTMLSerializer.php Added cookies support in lib/serialization/XHTMLSerializer.php Added cookies support in lib/Paloose.php Added cookies support in resources/schemas/sitemap/paloose-sitemap-1.3.4.rng Added more logging in lib/error-handling/ErrorPage.php and lib/error-handling/HandleError.php Added exit exception in lib/pipelines/Pipeline.php to handle run errors properly Minor code cosmetics
1.5.2b1	31st October 2011	Added date/time zone stuff for Mac OS X Lion (10.7.x)
1.5.2b2	2nd December 2011	Removed spurious line for $documentRoot
1.5.2b3	3rd December 2011	Cured problems with cachable feature (null pointers on some components) Minor cosmetics
1.5.2b4	6th February 2012	Minor cosmetics and better logging in some classes

1.5.2b5	26th June 2012	Sorted undefined logging variable problem in compact version
1.5.3	7th April 2015	Bug fix to sort reader problem preventing direct sourcing of HTML files
1.5.4	25th May 2015	Updated to working with PHP 5.5 (Renamed Generator class to Generater not file)
1.5.5	25th May 2015	Created a separate defaults file
1.6.0	4th September 2015	Cosmetics in build system (invisible to users)
1.7.0	29th November 2015	Sorted several points which did not conform to PHP STRICT
1.7.2	22nd March 2016	Cured null include/exclude problem in DirectoryGenerator Added dir:empty tag for null returns from scan directory in DirectoryGenerator
1.8.0	6th February 2017	Added ResourceExistsSelector Cured null otherwise problem in pipe selectors
1.8.1	24th March 2017	Added expansion of module variables in sitemap globals
1.9.0	8th May 2017	Added String2XMLTransformer (to support PCMS) Cured bug which gave protected error (removed gTranslatedDOM variable in I18Transformer and GalleryTransformer)
1.9.1	13th May 2017	Cured bug that prevented certain cases of cachable behaviour
1.10.0	7th November 2020	. Changed sizeof() function calls to use empty() and count() Updated to use PHP V5.5 Minor cosmetics
1.10.1	23rd November 2020	Cured nasty fault in DirectoryTransformer that stopped recursion of directories
1.11.0	7th December 2020	Added SCSS/SSS compiler
1.11.1	10th December 2020	Minor cosmetics
1.11.2	18th December 2020	Bug in SCSSCompiler (not including Utilities class)
1.11.3	14th April 2022	Added src/dst in pipeline for SCSS transformer (component values now default values).
1.12.0	12th August 2022	Corrected static call to non-static method DOMImplementation in HTMLSerializer. Cured bug where GEDCOM tags starting with "_" were not being recognised. Added MobileSelector to support selection on mobile devices.
1.12.3	4th September 2022	Cured problem preventing null namespaces in enclosed serializer elements. Updated RNG Schema for sitemaps. General cosmetics.
1.13.0	28th December 2022	Changed enclosed serializer elements to parameters. Changed enclosed global-variables elements to variables.