Paloose Documentation

Hugh Field-Richards

13th December 2008

Abstract

This is a paper copy of the Paloose documentation that is on the Paloose Web site (www.paloose.org). It was generated automatically by a suitable XSL transformer from XML to LaTeX so the formatting is not always optimum. I may hand-craft this version later on when I am totally happy with documentation, an event that may be some way off if experience is my guide, If there are any differences between the two versions then the Web site must be considered definitive.

Contents

1 Welcome to Paloose **1**
1.1 What people say . . . 2

2 Installation Guide **3**

3 Configuring Paloose **4**
3.1 Configuration Variables . . . 4
3.1.1 PALOOSE_DIRECTORY . . . 4
3.1.2 LOGGER_CONFIG . . . 4
3.1.3 CACHE_DIR . . . 4
3.1.4 ROOT_SITEMAP . . . 4
3.1.5 USER_EXCEPTION_PAGE, USER_EXCEPTION_TRANSFORM, USER_EXCEPTION_STYLE 4
3.1.6 GALLERY_INDEX, IMAGEMAGICK_BIN . . . 5
3.1.7 INTERNAL_EXCEPTION_PAGE, INTERNAL_EXCEPTION_TRANSFORM, INTERNAL_EXCEPTION_STYLE . . . 5
3.2 Configuration Variables (you should not change) . . . 5
3.2.1 SITEMAP_NAMESPACE, SOURCE_NAMESPACE, DIR_NAMESPACE . . . 5
3.2.2 I18N_NAMESPACE . . . 5
3.2.3 PAGE_HIT_TRANSFORM . . . 5
3.2.4 PALOOSE_NAMESPACE . . . 6
3.2.5 Finally 6
3.3 Changing the Configuration File Name . . . 6
3.4 Adding file types to the .htaccess file . . . 6

4 First Site **7**

5 Views **12**

6 Sitemap Guide **13**
6.1 The Structure . . . 13
6.2 Sitemap Namespace . . . 13
6.3 Common Attributes of Components . . . 13
6.4 Component Parameters . . . 13
6.5 Subsitemap Component Definitions . . . 13
6.6 Generators . . . 14
6.7 Transformers . . . 14
6.8 Serializers . . . 14
6.9 Readers . . . 14
6.10 Selectors . . . 15
6.11 Aggregators . . . 15
6.12 Matchers . . . 15
6.13 Views . . . 15
6.14 Pipelines . . . 15
6.14.1 Defining a Pipeline . . . 15
6.14.2 Pipeline Elements . . . 16
6.14.3 Internal Pipelines . . . 16
6.15 Sitemap Variables and Matching Patterns . . . 16

6.15.1 Wildcard Pattern Matching 17
6.15.2 Regular Expression Pattern Matching 17
6.15.3 Variables 18
6.15.4 Global Variables 18
6.16 Protocols 18

7 Mounting sitemaps **20**

8 Aggregation **21**

9 Generators **23**
9.1 Component Declaration 23

10 FileGenerator **24**
10.1 Caching Support (avaliable after Version 1.3.0) 24

16 DirectoryGenerator **35**

12 GedComGenerator **28**
12.1 Simple example 28

13 PXTemplateGenerator **30**
13.1 Simple Example 30
13.2 Caching Support (avaliable after Version 1.3.0) 30

14 Transformers **32**
14.1 Component Declarations 32

15 TraxTransformer **33**
15.1 Caching Support (avaliable after Version 1.3.0) 33

16 DirectoryGenerator **35**

17 Source Writing Transformer **37**
17.1 Source Writing Namespace 37
17.2 Source Writing Output 37
17.3 Source Write 37
17.4 Source Insert 38
17.4.1 Case 1 (replace not specified) 39
17.4.2 Case 2 (replace specified, node exists, overwrite true) 39
17.4.3 Case 3 (replace specified, overwrite false) 40
17.4.4 Case 4 (replace specified, node does not exist, overwrite true or false) 40
17.5 Source Delete 41

18 SQLTransformer **42**
18.1 Errors 42

19 FilterTransformer **43**

20 LogTransformer **45**

21 PasswordTransformer **46**

22 Serializers **47**
22.1 Component Declaration 47
22.2 HTMLSerializer 47
22.3 XHTMLSerializer Usage 48
22.4 TextSerializer Usage 49
22.5 XMLSerializer Usage 50

23 Actions **51**
23.1 Component Declarations 51

24 SendMailAction **52**

25 Authorisation Actions **54**
25.1 Authorisation Example 54
25.2 Authorisation Handler 54
25.3 Protecting Individual Pages 55

26 Authorising the User **56**
26.1 The Admin User File 56

27 User Login **60**

28 User Logout **62**

29 Flows (and Forms) **63**
29.1 Requirements 63
29.2 Add User Form 63

30 The Flow Script **65**

31 Next Sequence and Checking for Errors **68**

32 Confirming the data **71**

33 Paloose Forms **72**
33.1 Basic Structure 72
33.1.1 Simple Input Field (Input). 72
33.1.2 Simple Password Field (secret). 72
33.1.3 Hidden Field (hidden). 73
33.1.4 Output Field (output). 73
33.1.5 Form button (submit). 73
33.1.6 Multiple Select Fields. 73
33.1.7 Single Select Fields. 74
33.1.8 Text Area 75
33.1.9 Label Field 76
33.1.10 Value Field 76
33.1.11 Hint Field 76
33.1.12 Violations Field 76

34 Optimising Paloose Performance **77**
34.1 Some Background 77

35 Optimising Paloose Performance **80**
35.1 Caching the Sitemap 80

36 Optimising Paloose Performance **83**
36.1 Optimising Paloose Code 83
36.2 Using a Data Cache 84
36.3 Conclusions for Paloose Web Site Design 84

37 Caching Discussion **85**
37.1 Introduction 85
37.2 Caching Code 85
37.3 Caching the Sitemap 85
37.4 Caching the Page Components 85
37.5 Checks and Balances 86

38 How to use the Gallery Transformer 87
38.1 Sitemap . . . 87
38.2 Defining the Gallery Index File . . . 88
38.3 Adding the Gallery to your XML Content . . . 89
38.4 Processing the Transformer Output . . . 90

39 How to use the SQL Transformer 97
39.1 Sitemap . . . 97
39.2 Using the SQL Transformer . . . 98
39.2.1 A simple query . . . 98
39.2.2 A simple query with substitution. . . . 99

40 How to use the Page-Hit Transformer 101
40.1 Sitemap . . . 101
40.2 Content . . . 101
40.3 Finally . . . 101

41 How to use the I18NTransformer for Internationalization 103
41.1 Sitemap . . . 103
41.2 Building the Catalogue Files . . . 103
41.3 Building the Text File . . . 104
41.3.1 ¡i18n:text¿ . . . 104
41.3.2 Attributes . . . 104
41.3.3 Date and Time Formatting . . . 104
41.3.4 Number Formatting . . . 105

42 How do I port Paloose Sites to Cocoon? 106
42.1 An example . . . 106
42.1.1 Root sitemap . . . 106
42.2 What about the Subsitemaps, XML Content and XSL Transforms? . . . 108

43 Frequently Asked Questions 109
43.1 About the author . . . 109
43.2 What's with the name? . . . 109
43.3 What future for Paloose? . . . 109
43.4 What facilities do you intend to add to Paloose? . . . 109
43.5 Do you intend to add PDF support via FOP? . . . 109
43.6 What systems will it run on? . . . 109
43.7 Why don't you use SAX like Cocoon, instead of DOM? . . . 109
43.8 How can I improve the performance of Paloose? . . . 110
43.9 What about caching in Paloose? . . . 110
43.10 Why did you not use CForms and JavaScript for flows? . . . 110
43.11 Who designs your own sites that use Paloose? . . . 110
43.12 Are there examples of sites that use Paloose? . . . 111
43.12.1 My sites . . . 111
43.12.2 Others . . . 111
43.13 How do I port Paloose Sites to Cocoon? . . . 111
43.14 How do I port Cocoon Sites to Paloose? . . . 111
43.15 How do I write Components for Paloose? . . . 111
43.16 Do you have any support for iPhone? . . . 111
43.17 Help! I have tried everything but I am still getting page not found. . . . 111
43.18 Help! Why am I getting "Class 'XsltProcessor' not found" errors . . . 112
43.19 Help! Why am I getting "Parse error: syntax error, unexpected '=', expecting '(' in /../../paloose/lib/Paloose.php on line xx" errors . . . 112
43.20 Help! Why am I getting "Input file not found" as the only browser output? . . . 112
43.21 Are there schemas for the XML used in this site? . . . 112
43.22 Why do you not use schemas for the sitemaps? . . . 112
43.23 Technical Trivia . . . 112

1 Welcome to Paloose

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

Paloose supports the following:

- Generators:
  - Aggregation
  - Simple XML file generator
  - Directory listing generator
  - XML Template file generator
  - Preliminary support for Tidy generator (eg Word documents)
  - GedCom generator
- Transformers:
  - XSL transforms
  - Logging Transformer
  - Support for multi-language through i18n
  - XML Write to external file (SourceWriting)
  - Querying an SQL database
  - Filter Transformer
  - Password Encoding Transformer
  - Page-hit Transformer
  - Picture Gallery Transformer
- Serializers:
  - HTML serializer
  - XHTML serializer
  - XML serializer
  - Text serializer
- Actions:
  - SendMail action
  - Authentication actions (supporting login and restricted pages)
- Selectors:
  - Browser Selector
  - Request Parameter Selector
- Forms:
  - Paloose Forms (based on JX forms)
  - Flowscripts and continuations
  - Entry checking
- Internal-only pipelines
- Wildcard and regular expression pattern matchers
- Redirection
- Simple Resource readers
- Subsitemaps
- Views
- Error handling
- Sitemap variables and pseudo-protocols
- Global sitemap variables
- Request parameters

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 will probably be unnoticable 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.

To get started read the documentation pages and download the latest version of Paloose.

1.1 What people say ...

> *I started using [Paloose] soon after your announcement on the Cocoon-users mailing list, and have been very impressed by its completeness. I'd been wanting a way to work in a Cocoon-like paradigm ... that could be used on "standard" web hosts, and Paloose has filled that need very well.*

Jason Johnston (327creative)

> *I just want to say thank you for developing Paloose and releasing it to the world. I have wonder at times about a PHP framework similar to Cocoon. ... Paloose does exactly what I need ... [it] is an excellent package. I was able to move a small web site from Cocoon to Paloose with very little changes. Cocoon and Paloose both demonstrate that content management can take place in a simple manner without the need for a database. ... Keep up the great work. You already have a great package that can only get better.*

Gary T. Schultz (Web Administrator, Wisconsin Department of Commerce)

2 Installation Guide

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

1. Unpack the download into the destination directory. This is currently in `/Library/WebServer/Documents` on my development box (Mac G5 running Mac OS X 10.5.4). This will be *ROOT_DIR* in the examples below.

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

2. That's it.

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

3 Configuring Paloose

Paloose configuration is done through a single file. This file can be called anything (in the current sites case it is `paloose.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, We will look at this in more detail below.

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 Palose will assign defaults: see the file "`config.inc.php`".

3.1 Configuration Variables

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

3.1.1 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 my changing this line.

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

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

3.1.3 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( 'CACHE_DIR', 'resources/cache' );
```

3.1.4 ROOT_SITEMAP

Normally should not have to change this. The directory is always the top level user directory.

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

3.1.5 USER_EXCEPTION_PAGE, USER_EXCEPTION_TRANSFORM, USER_EXCEPTION_STYLE

The error page to give back for User errors (errors in sitemap etc). They are not run time errors (such as missing page, 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. Must always have a leading path separator.

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

3.1.6 GALLERY_INDEX, IMAGEMAGICK_BIN

The file that contains the gallery information in each gallery directory. See the how-to page on the gallery for more information. The ImageMagick directory is there if the PATH on the server is not set to pick up correct ImageMagick binaries

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

3.1.7 INTERNAL_EXCEPTION_PAGE, INTERNAL_EXCEPTION_TRANSFORM, INTERNAL_EXCEPTION_STYLE

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

3.2 Configuration Variables (you should not change)

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

3.2.1 SITEMAP_NAMESPACE, SOURCE_NAMESPACE, DIR_NAMESPACE

This 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( 'SITEMAP_NAMESPACE', 'http://apache.org/cocoon/sitemap/1.0' );
define( 'SOURCE_NAMESPACE', 'http://apache.org/cocoon/source/1.0' );
define( 'DIR_NAMESPACE', 'http://apache.org/cocoon/directory/2.0' );
```

3.2.2 I18N_NAMESPACE

Change these with caution — should only be changed for different translation mechanism. Requires deep understanding of Paloose internals :-)

```
define( 'I18N_NAMESPACE', 'http://apache.org/cocoon/i18n/2.1' );
```

3.2.3 PAGE_HIT_TRANSFORM

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

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

3.2.4 PALOOSE_NAMESPACE

This is the Paloose namespace that will never need to change unless there is a very good reason — subtext for, it won't change.

Do not change anything else within the file — especially not the line:

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

3.2.5 Finally ...

Definitely do not change this line unless you know what you are doing :-)

```
require_once( PALOOSE_DIRECTORY . '/lib/Paloose.php' );
```

3.3 Changing the Configuration File Name

As was said above there is no need to change the name of the configuration file that sits in the user's site, and which starts Paloose running for each request. However if you do then the `.htaccess` file must be changed to reflect this. At present the default for this file is:

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

If you want the configuration file to be called, say, `mySite.php`, then the `.htaccess` file must be changed to:

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

3.4 Adding file types to the .htaccess file

If you want to process request other than `html` type, then these have to be added to the `.htaccess` file. If youu do not then the Apache server (or local equivalent) will process them instead. Say, for example, that we wanted Paloose to deal with RELAX NG files, then we would add the following line:

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

4 First Site

First, please read some of the Cocoon documentation (beginner's guide) to understand the pinciples of using the Cocoon approach. Note that the Paloose system is a subset of Cocoon so almost all of the features in Paloose are directly derived from Cocoon. In order to see how it works let us build the archetypal beginner's site: "Hello World".

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

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

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

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

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

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

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

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

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

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

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

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

```
   </xsl:template>

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

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

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

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

     </xsl:stylesheet>
```

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

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

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

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

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

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

6. We need to tie them all together so that a request for `http://hostname/HelloWorld/page.html` will produce the page that we require. The control of this is within the *sitemap.xmap* file. It follows standard Cocoon practice with only minor differences in the component definitions. First of all we define the components that we are going to use. Note that to those more used to Cocoon the '*src*' attribute normally defines a Java package, in Paloose it is the name of a PHP file. So `paloose://lib/generation/FileGenerator`' defines a file '`$`

{ROOT_DIR}/paloose/lib/generation/FileGenerator.php'. This means that you can define your own components fairly easily and place them where you want (note that I have not documented the process of writing new components yet — watch this space.

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

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

   <map:components>

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

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

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

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

   </map:components>
```

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

```
   <map:pipelines>

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

   </map:pipelines>

</map:sitemap>
```

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

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

This tells Apache that all requests for files in the HelloWorld directory (in which the .htaccess file sits) should obey the rule above. So a request for 'http://host-name/HelloWorld/page.html' world

be translated to ‘`http://host-name/HelloWorld/paloose.php?url=page.html`’.

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 on PHP4. You will need to speak to your ISP (or equivalent) to find out how to direct all Paloose code to use PHP5 (assuming that they have this as an option).

The remedy is usually some extra lines in the `.htaccess` file.

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

```
<?php
```

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' );
```

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' );
```

Normally should not have to change this. 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). They are not run time errors (missing page (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.

```
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 give back 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.x
```

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

Change this with caution — should not be changed in 99.9% of cases

```
define( 'SITEMAP_NAMESPACE', 'http://apache.org/cocoon/sitemap/1.0' );
```

Do not change anything below this line!

```
   require_once( PALOOSE_DIRECTORY . '/lib/Paloose.php' );
?>
```

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

5 Views

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

```
<map:views>
   <map:view name="raw" from-label="raw-content">
      <map:transform src="context://resources/transforms/xml2html.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" \textit{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 "`xml2html.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*.

6 Sitemap Guide

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

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

6.1 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>
```

6.2 Sitemap Namespace

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

6.3 Common Attributes of Components

Like Cocoon all components have some common attributes.

- *name* — Gives the component an identifying name by which it may be referenced in the pipeline section.
- *src* — Specifies the PHP file to be used implementing this component. ***This is a major change to Cocoon where the class is used not the file name.***

6.4 Component Parameters

Some component declaration use parameters to allow user-settable values to be incorporated in the definition, for example see the PageHitTransformer.

6.5 Subsitemap Component Definitions

Versions before 0.14.1: Component definitions in subsitemaps are handled differently to Cocoon. In the latter it is possible to redefine or declare a new component. In Paloose all declarations should be placed in the root sitemap, with just the default changed in the subsitemap.

Components can be defined with a subsitemap overriding a declaration made in a higher sitemap. The new definition is available to all sitemaps called by the one where the declaration is made.

6.6 Generators

A Generator generates XML content as DOM objects and initializes the pipeline processing. ***Using DOM instead of SAX is a change from Cocoon*** (see this FAQ). For example defining the *FileGenerator* generator:

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

The *default* attribute specifies the type of generator to use if none is specified in a pipeline. Further discussion on the types of generator are on the Generators Page.

6.7 Transformers

Transformers change a DOM into a another DOM using either an XSLT file or a custom translator. For example this defines *TRAXTransformer* which performs standard XSLT transformations, similar to Cocoon.

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

The *default* attribute specifies the type of transformer to use if none is specified in a pipeline. Further discussion on the types of generator are on the Transformers page.

6.8 Serializers

Serializers change a DOM into a stream of characters that the client can use. There are several serializers that work in a similar fashion to Cocoon, although not so extensive: only encoding and doctype in the XHTML/HTML serializers is currently supported. For example

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

See the Serializers page for more information.

6.9 Readers

Readers are used to deliver resources directly from the server to the client, usually without modification. As such they are generator, transformer and serializzer all in one. In Paloose there is currently only one type of matcher: *ReasourceReader*, which is equivalent to the Cocoon reader of the same name.

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

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

6.10 Selectors

Selectors allow conditional processing within the pipeline. Currently the only selector supported by Paloose if the *BrowserSelector* which allows different routes through a pipeline to be taken dependent on the client's browser.

See the Selectors page for more information.

6.11 Aggregators

Aggregators takes several XML documents and combine them into a single DOM, with appropriate enclosing tags for each part. They are equivalent to generators and thus must be placed first in a pipeline. See Aggregators page for more information.

6.12 Matchers

Matchers are used to match the request into a particular pipeline. In Paloose (version 0.7.0 or later) there are two types of matcher: *WildcardURIMatcher* and *RegexpURIMatcher*. *WildcardURIMatcher* is equivalent to the Cocoon matcher of the same name. However *RegexpURIMatcher* is slightly different to its Cocoon equivalent: it uses a PHP-modified Perl regular expression syntax. For details of this see the PHP5 manual pages)

```
<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 *default* attribute specifies the type of matcher to use if none is specified in a pipeline.

6.13 Views

Views are means of interrupting the normal pipeline flow under the control of the user via the quesry string. It is similar to that of Cocoon but not quite so extensive. It is a useful debugging aid. See the Views documentation page for more information.

6.14 Pipelines

Like Cocoon the heart of Paloose is the sitemap pipeline. Documents are entered one end of the pipeline (via a generator), progress through pipeline via zero or more transformations (transformers) and end up being processed to the client (via a serializer). Paloose also support aggregation to build up a document from a variety of other documents (or pipelines).

Paloose also supports a similar error handling procedure to Cocoon, although there are some slight differences withn the handlers.

6.14.1 Defining a Pipeline

Pipelines are defined within the `map:pipelines` element within individual `map:pipeline` elements.

```
<map:pipelines>
   <map:pipeline>
      <!-- pipeline elements -->
   </map:pipeline>

   <map:handle-errors>
```

```
      ...
      </map:handle-errors>

   </map:pipelines>
```

An individual pipeline can be marked as “ internal only” so that itis impossible to activate it from an external URL request. This is generally done on aggregation to hide the intermediate parts from direct access. It is declared

```
<map:pipelines>
   <map:pipeline internal-only="true">
      <!-- pipeline elements -->
   </map:pipeline>

</map:pipelines>
```

6.14.2 Pipeline Elements

Paloose supports a subset of the Cocoon sitemap pipeline elements:

- *map:match* — Selects which pipeline to run dependant on the incoming request.
- *map:mount* — Transfers control to a sub-site map for further processing.
- *map:parameter* — Defines a parameter to be carried into one of the pipeline elements.
- *map:generate* — Defines the generation step.
- *map:aggregate, map:part* — Combines several XML files into a single document.
- *map:transform* — Defines the transformation step.
- *map:serialize* — Defines the serialization step.
- *map:select* — Select which pipeline fragment to run.
- *map:act* — Perform some interaction with external resource.
- *map:redirect-to* — Redirect this URL request to another URL.
- *map:handle-errors* — Handles processing errors.

6.14.3 Internal Pipelines

Sometimes it is desirable to mark a pipeline to be only accessable from the sitemap itself. This is of particular use when aggregation is used (and authorisation). It prevents scraps of documents which are used in assembling a complete document from being accessed. See the documentation on aggregation for an example of how this is used.

The general structure is:

```
<map:pipelines>
   <map:pipeline>
      <!-- matchers appearing here are available externally and internally -->
   </map:pipeline>

   <map:pipeline internal-only="true">
      <!-- matchers appearing here are available internally -->
   </map:pipeline>
</map:pipelines>
```

6.15 Sitemap Variables and Matching Patterns

It is possible to define variables in the Cocoon sitemap. Paloose uses this principle, but, at present, uses a very restricted set. Like Cocoon variables are defined by enclosing them in brackets “{...}”. The primary area where this is used is in the pattern matching.

6.15.1 Wildcard Pattern Matching

For example take the wildcard matcher

```
<map:match pattern="*/*.html">
   ...
</map:match>
```

The wildcards in the match pattern are assigned to internal pattern variables “{1}”, “{2}” etc, in a similar fashion to Perl regular expressions. So in the above case if the pattern to be matched is “`documentation/index.html`” then there would be a match. The internal variables would be set {1} = “documentation” and {2} = “index”. These variables are used within the various pipeline components to resolve string expressions. For example

```
<map:match pattern="*/*.html">
   <map:generate src="context://{1}/{2}.xml"/>
   ...
</map:match>
```

In this case the generator would fetch the file `documentation/index.xml` from the current application context (see pseudo protocols below). Note that the constructions “**” and “*” are different and are identical to their Cocoon counterparts.

- “`*`” — Match any character except the path separator.
- “`**`” — Match any character including the path separator.

The values for the pattern variables are taken from the current matcher. If you want to access a higher matcher (wherever it is) then you would have to use the string {`../1`}, which means access the pattern variable {`1`} in the previous matcher. These can be extended as necessary, for example {`../../1`} to access a matcher two levels above the current one.

6.15.2 Regular Expression Pattern Matching

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

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

would match all requests that end in “html”, “tex” and “xml”. In the following all requests for documents between “nb1996” and “nb1999” would read the html file directly:

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

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

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

```
    <map:serialize/>
</map:match>
```

6.15.3 Variables

There are other variables that can be used within the sitemap. Currently the only one is derived on the *RequestParameterModule*. This 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" />
```

Note the format of the variable, it is formed of a pair "{*module name*:*variable name*}". As other predefined modules are added the *module name* will be used to access the variables defined by the module.

6.15.4 Global Variables

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

```
<map:pipelines>

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

   ...
```

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

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

6.16 Protocols

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

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

7 Mounting sitemaps

One of the most useful concepts of the 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 about of inheritance of components so that they do not have t be redefined. Although this does not preclude having extra components defined in the susitemaps. Subsitemaps allow a hierarchy or tree of sitemaps underneath the master sitemap. The advantage of this is that a site with several different areas can be developed separately using a sitemap for each area making larger sites easier to maintain. The master sitemap controls which sitemap is used dependent on the request.

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

The attributes are similar to those used in Cocoon

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

8 Aggregation

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

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

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

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

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

As and 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>
```

9 Generators

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

- 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.
- TidyGenerator — Takes an HTML file generates by Microsoft Word and translates it to an XML document suitable for processing in the pipeline.
- GedComGenerator — Takes an GEDCOM file and produces an XML equivalent.

9.1 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:generator name="tidy" src="resource://lib/generation/TidyGenerator">
         <map:parameter name="char-encoding" value="utf8"/>
         <map:parameter name="clean" value="yes"/>
         <map:parameter name="word-2000" value="yes"/>
   </map:generator>
</map:generators>
```

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

10 FileGenerator

File generators read an XML file and present it to the pipeline for processing. They are always the first item of a pipeline. 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>
```

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

10.1 Caching Support (avaliable after Version 1.3.0)

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

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

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

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

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

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

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

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

```
   </map:generator>
</map:generators>

...

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

11 DirectoryGenerator

Directory generators read a 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="test-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="xml2html"/>
       </map:match>
   </map:pipeline>

</map:pipelines>
```

where

- *src* — the root directory to start scanning.
- *depth* — (optional) the recurse depth for the directory scan (defaults to 1, the requested directory only).
- *dateFormat* — (optional) the format of the date string (follows the PHP format).
- *include* — (optional) a regular Perl expression defining what files/directories to include in the scan.
- *exclude* — (optional) a regular Perl expression 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="xml2html.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.

12 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 and change it to the GEDCOM 6 XML representation. If that is required then a suitable transformer must be written (currently not supplied with Paloose). So 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 type="gedcom" src="context://content/data/Field-Richards.ged" label="xml-c
            <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.

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

12.1 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
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 @S0@
   0 @S0@ SUBM
   1 NAME Hugh Field-Richards
   1 ADDR xxxxxxxxxxxxx
   2 CONT xxxxxxxxxxxxx
   2 CONT xxxxxxxxxxxxx
   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="xxxxxxxxxxxxx">
         <g:cont data="xxxxxxxxxxxxx"> </g:cont>
         <g:cont data="xxxxxxxxxxxxx"> </g:cont>
      </g:addr>
      <g:email data="hsrf@hsfr.org.uk"/>
   </g:subm>
   ...
```

13 PXTemplateGenerator

PXTemplate generators read an XML file and present it to the pipeline for processing. In essence they are 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 are always 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>
```

13.1 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>
```

13.2 Caching Support (avaliable 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.

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

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

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

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

```
<map:match pattern="*.xml">
   <map:generate type="px" 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="px" src="resource://lib/generation/PXTemplateGenerator" cachable="no">
      <map:use-request-parameters>true</map:use-request-parameters>
   </map:generator>
</map:generators>

...

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

14 Transformers

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

- TRAXTransformer — allows processing of the DOM according to a specified XSLT file.
- PageHitTransformer — inserts the number of page hits for the page that is being processed.
- GalleryTransformer — manages a photo gallery system.
- I18nTransformer — supports internationalisation.
- SourceWritingTransformer — allows the pipeline XML to be diverted to an external file.
- FilterTransformer — allows resticted numbers of named tags to be passed.
- SQLTransformer — allows queries to be made to a SQL database.
- PasswordTransformer — provides a simple password encoding (based on md5).
- LogTransformer — provides a simple method of logging XML fragments from within the pipeline.

14.1 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="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.

15 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`.

15.1 Caching Support (avaliable after Version 1.3.0)

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

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 name="xslt" src="resource://lib/transforming/TRAXTransformer" 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 “`?cachable=1`” or “`?cachable=0`“

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

16 DirectoryGenerator

Directory generators read a 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="test-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="xml2html"/>
       </map:match>
   </map:pipeline>

</map:pipelines>
```

where

- *src* — the root directory to start scanning.
- *depth* — (optional) the recurse depth for the directory scan (defaults to 1, the requested directory only).
- *dateFormat* — (optional) the format of the date string (follows the PHP format).
- *include* — (optional) a regular Perl expression defining what files/directories to include in the scan.
- *exclude* — (optional) a regular Perl expression 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="xml2html.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.

17 Source Writing Transformer

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*

17.1 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`".

17.2 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>
```

17.3 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>
```

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>
```

17.4 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 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:

17.4.1 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>
```

17.4.2 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>
```

17.4.3 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.

17.4.4 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>
```

17.5 Source Delete

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

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

where

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

For example:

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

18 SQLTransformer

Note that this does not match the Coocoon method 100%. There are important differences that are discussed in the How2 example.

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 use of SQLTransformer 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:transformer>
   <map:transformer name="xslt" src="resource://lib/transforming/TRAXTransformer">
      <map:use-request-parameters>true</map:use-request-parameters>
   </map:transformer>
</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:transform>
      ...
   </map:match>
```

For example of how to use the SQLTrnasformer and an explanation of the various attributes see the example How2 page.

18.1 Errors

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

```
<page:content xmlns:default="http://apache.org/cocoon/SQL/2.0" xmlns:t="http://www.hsfr.org.uk/Sc
   <t:heading level="1">SQL Transform Test</t:heading>
   <default:sql-error xmlns="http://apache.org/cocoon/SQL/2.0">
      <default:host>localhost:3306</default:host>
      <default:user>root</default:user>
      <default:password></default:password>
      <default:message>SQL query error: => query: select * fom composer </default:message>
   </default:sql-error>
</page:content>
```

19 FilterTransformer

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

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

 ...

 <map:pipeline>

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

where

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

Say the following data is stored in the database:

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

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

```
<page:content xmlns:default="http://apache.org/cocoon/SQL/2.0"
    xmlns:t="http://www.hsfr.org.uk/Schema/Text">
    <t:heading level="1">SQL Transform Test</t:heading>
    <default:row-set nrofrows="8" name="music">
       <default:block id="1"/>
       <default:block id="2"/>
       <default:block id="3">
          <default:row>
             <default:name>Haydn</default:name>
             <default:forenames>Franz Joseph</default:forenames>
             <default:birth>1732-03-31</default:birth>
             <default:death>1809-05-31</default:death>
          </default:row>
          <default:row>
             <default:name>Bernstein</default:name>
             <default:forenames>Leonard</default:forenames>
             <default:birth>1918-08-25</default:birth>
             <default:death>1990-10-14</default:death>
          </default:row>
       </default:block>
       <default:block id="4"/>
    </default:row-set>
 </page:content>
```

20 LogTransformer

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

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

<map:pipelines>
   <map:pipeline>

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

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

where

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

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

```
==============================================================================
   <t:p>Installation is very simple &#x2014; just follow the (brief) instructions
         <link:link type="uri" ref="install.html">here</link:link>.</t:p>
```

21 PasswordTransformer

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

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

<map:pipelines>
   <map:pipeline>

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

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

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

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

After processing with the *PasswordTransformer* the tag becomes:

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

22 Serializers

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

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

22.1 Component Declaration

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

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

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

22.2 HTMLSerializer

Serializers are the last part of a pipeline (unless there is a reader or mount pipeline element). So 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/HTMLSer
         <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: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 HTML serializer would produce the following typical output:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.
<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>
```

22.3 XHTMLSerializer Usage

XHTMLSerializers are similar to the HTMLSerializer except that they produce 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/XHTMLS
      <doctype-public>-//W3C//DTD XHTML 1.0 Strict//EN</doctype-public>
      <doctype-system>http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd</doctype-system>
      <encoding>iso-8859-1</encoding>
    </map:serializer>
  </map: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/x
<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>


  <map:components>
    ...
    <map:serializers default="xml">
      <map:serializer name="xhtml" src="resource://lib/serialization/XHTMLSerializer">
        <doctype-public>-//W3C//DTD XHTML 1.0 Strict//EN</doctype-public>
        <doctype-system>http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd</doctype-system>
        <encoding>iso-8859-1</encoding>
      </map:serializer>
    </map:serializers>

  </map:components>

  <map:pipelines>
    <map:pipeline>

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

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

22.4 TextSerializer Usage

There are occasions where it is useful, to output pure text, particularly on internal pipelines that return data to another pipeline (aggregation etc). 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.

22.5 XMLSerializer Usage

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="xhtml" src="resource://lib/serialization/XHTMLSerializer">
         <doctype-public>-//W3C//DTD XHTML 1.0 Strict//EN</doctype-public>
         <doctype-system>http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd</doctype-system>
         <encoding>iso-8859-1</encoding>
      </map:serializer>
      <map:serializer name="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>
```

23 Actions

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

- SendMailAction — allows the pipeline to send mail on the basis of the data passing through the pipeline.
- Authorisation Actions — allows the pipeline to restrict use until some authorisation takes place (login etc).

23.1 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="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>
```

Details on how to use these components are held on a separate page:

- SendMailAction
- Authorisation Actions

24 SendMailAction

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

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

where

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

It is best to show an example of *SendMailAction* to see how it is used in the pipeline. For example the Paloose site uses it to send EMAILs from the Contacts Page and has 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.

25 Authorisation Actions

This whole framework is still work in progress and so may change, although I hope not by much, therefore please treat with caution. Note also that it is not the strongest method of preventing unauthorised access. The Paloose authorisation framework is only very loosely based on Cocoon so it is important to read the following carefully if you want to use this facility. Please read the Cocoon documentation on authentication as it is a useful background (it is much better than mine anyway).

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

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

25.1 Authorisation Example

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

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

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

25.2 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.***

25.3 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*).

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

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

The next section deals with the actual authorising mechanism.

26 Authorising the User

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

```
<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.

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

```
<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 ) = $g
         <ID><xsl:value-of select="username"/></ID>
         <data>
            <ID><xsl:value-of select="username"/></ID>
            <username><xsl:value-of select="username"/></username>
            <password><xsl:value-of select="password"/></password>
         </data>
      </xsl:if>
   </xsl:template>

</xsl:stylesheet>
```

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

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

```
</authentication>
```

or a blank structure:

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

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

The next section deals with the login mechanism.

27 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. I have used the above for simplicity at this stage.

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

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

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

The next section deals with the logout mechanism.

28 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.

29 Flows (and Forms)

It is worth reading the FAQ entry on flows for some background information on why Paloose Forms and Flows differ so much from Cocoon. Like the authorisation actions it is best to describe a simple example.

29.1 Requirements

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:

- Username
- Password
- data:
  - 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>
```

29.2 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 the just the form not the surfounding 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.

30 The Flow Script

This part of the system is the one that diverges most from Cocoon — in fact it is completely different 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.
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 *PXTemplateTransformer*:

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

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 of the flow process and returns the page `addUserTest-1.html` since the continuation id is “0”.

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

```
   <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:

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

Running this will produce a form similar to:

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

31 Next Sequence and Checking for Errors

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

```
http://<hostname>/pp/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 his time it is directed to the 2 case of the switch statement which now has to output the next form in the sequence. If data is to be checked, this is where it is done. How you do this is up to you but the following seems to work quite well for this case. First declare a violations array:

```
<?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" => "",
   );

   \textit{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 :
            \textit{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 condi
            // 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
                  \$ errors = true;
               $\}$
               if ( \$ this->gAddFormTestModel[ ‘password’ ] != \$ this->gAddFormTestModel[ ‘pass
                  \$ 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 viol
               \$ this->gContinuation--;
               \$ finished = false;
            $\}$ else $\{$
               // No errors so send next page
               \$ this->sendPage(
                  ‘‘addUserTest-2.html’’,
                  \$ this->gAddFormTestModel,
                  ++\$ this->gContinuation,
                  \$ this->gViolations );
               \$ finished = true;
             $\}$
            break;}

         ...
         }
      }
   }
```

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

The next page describes the confirmation of data.

32 Confirming the data

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

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

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

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

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

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

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

which will display:

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

I have set up the example above so that you can see how it works.

33 Paloose Forms

The PForms framework is still work in progress and, while mostly stable, is not complete in some areas. I hope that this situation will not last long.

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

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

33.1 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:__f
   <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.

33.1.1 Simple Input Field (Input).

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

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

where the attributes are:

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

33.1.2 Simple Password Field (secret).

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

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

```
    <pf:violations/>
</pf:input>
```

where the attributes are:

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

33.1.3 Hidden Field (hidden).

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

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

where the attributes are:

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

33.1.4 Output Field (output).

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

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

where the attributes are:

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

33.1.5 Form button (submit).

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

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

where the attributes are:

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

33.1.6 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 mutiple 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 checke at page load time. If *checked* is omitted then *false* is assumed.

For example:

```
<pf:select appearance="full" ref="roles">
   <pf:value>{flow:roles}</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 manage users: change password, delete/add users, change roles</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 (assuming items 1 and 3 are selected):

33.1.7 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>{flow:roles}</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 manage users: change password, delete/add users, change roles</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:

33.1.8 Text Area

text area fields allow multi-line

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

where the attributes are:

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

For example:

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

33.1.9 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

33.1.10 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>
```

33.1.11 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>
```

33.1.12 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>
```

34 Optimising Paloose Performance

34.1 Some Background

First of all it is useful to look at the performance of a completely "vanilla" system. Currently the test server for Paloose that I use is 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 spec'ed machine but it serves — 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 *voila*!

First of all to produce a bench mark I ran some tests (using siege) running on my local network. The server is connected to this network wirelessly via a Netgear DG834G router, while the siege machine, which is 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 sige test on Apache (using the same HTML version of the XML Paloose test file) and also the same with Tomcat runnning 2.1.10 Cocoon (caching on). These were run against an identical set of XML files and transforms in Paloose. The sitemap was:

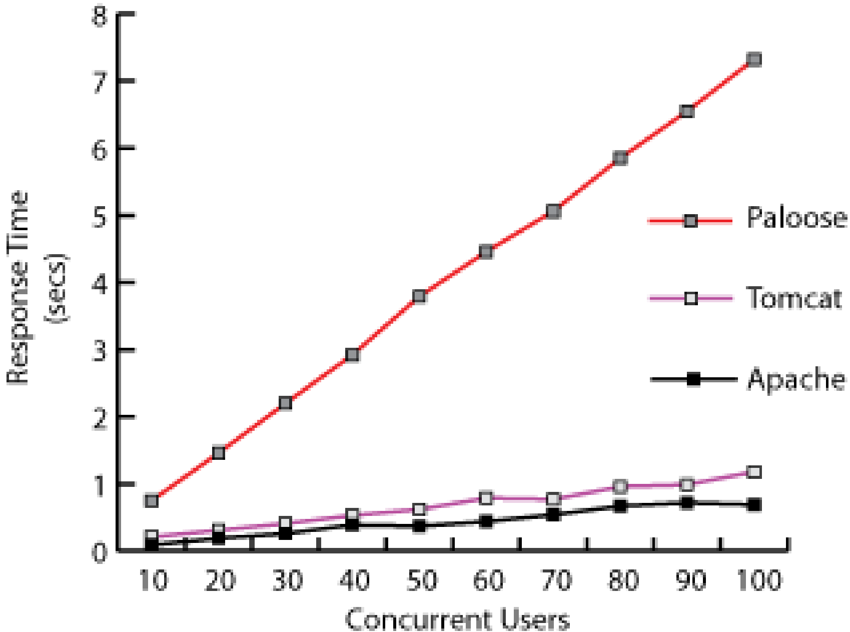
```
<map:sitemap xmlns:map="http://apache.org/cocoon/sitemap/1.0">

   <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/HTMLSer
            <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/XHTMLS
            <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/TextSe
         <map:serializer name="xml" mime-type="text/xml" src="resource://lib/serialization/XMLSerial
```

```
      </map:serializers>

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

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

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

   </map:components>

   <map:pipelines>

      <map:pipeline>

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

      </map:pipeline>

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

   </map:pipelines>

</map:sitemap>
```

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

As can be seen from the graph below Paloose does not score highly against Apache or a cacheTomcat/Cocoon. This is quite expected and a result of many factors: PHP, parsing the sitemaps each time, using DOM instead of SAX in the sitemap pipeline. The best we can say is that there is room for improvement and that all the sites that I have run have not needed the raw speed that Apache or Tomcat could

deliver. It is also another reason why Apache should be used to serve the files hat Paloose does not deal with (CSS, graphics etc).



The next page looks at how we can increase the perfomance

35 Optimising Paloose Performance

35.1 Caching the Sitemap

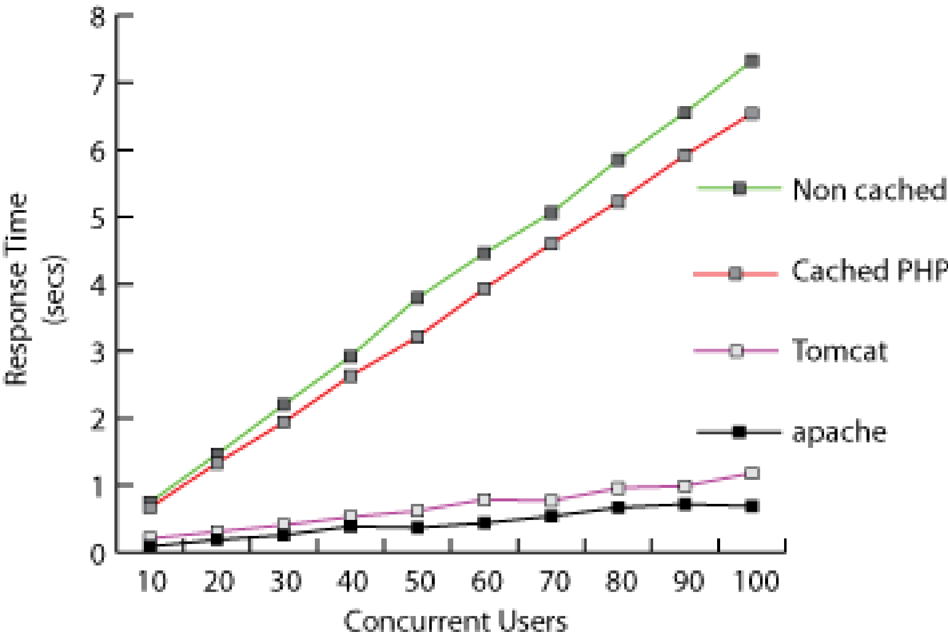
So how do we improve this situtation? After much testing an interesting conclusion surfaced: that caching the sitemap in Paloose (while undoubtedly providing some speed increase) was not the most effective way of improving things.

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

```
<?php

class CachedSitemap {

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

   function __construct()
   {
   }

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

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

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

   public function run( $inURL, $inQueryString, $inInternalRequest ) {
    global $gVariableStack;
 try { // Pipelines parse
{ // Pipeline parse
```

```
if ( ( $matchArray = Match::match( 'WildcardURIMatcher', $inURL, '**.html' ) ) != NULL ) {
   $gVariableStack->push( $matchArray );
   $dom = GeneratorPipeElement::generate( 'FileGenerator', 'context://content/{1}.xml',
            'xml-content', $matchArray, $this->gRequestParameters );
$this->gParameters = new Parameter();
$this->gParameters->setParameterList( $this->gRequestParameters );
$this->gParameters->setParameter( 'page', '{1}' );
$dom = TransformerPipeElement::transform( 'TRAXTransformer',
'context://resources/transforms/page2html.xsl',
$dom, $label, $matchArray, $this->gParameters, $gVariableStack );
$this->gParameters = new Parameter();
$this->gParameters->setParameterList( $this->gRequestParameters );
$dom = TransformerPipeElement::transform( 'TRAXTransformer',
'context://resources/transforms/stripNamespaces.xsl',
$dom, $label, $matchArray, $this->gParameters, $gVariableStack );
$this->gParameters = new Parameter();
$dom =    $dom = SerializerPipeElement::serialize( 'HTMLSerializer',
$dom, $label, $matchArray, $this->gParameters, $this->gOutputStream );
$gVariableStack->pop(); }
} // Pipeline parse
} catch ( ExitException $e ) { // Pipelines parse
            throw new ExitException();
         } catch( UserException $e ) {
            // handle error pipeline
         } catch( RunException $e ) {
            // handle error pipeline
            $dom = GeneratorPipeElement::generate( 'FileGenerator', 'context://content/error.xml',
            '', $matchArray, $this->gRequestParameters );
$this->gParameters = new Parameter();
$this->gParameters->setParameterList( $this->gRequestParameters );

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

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

?>
```

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



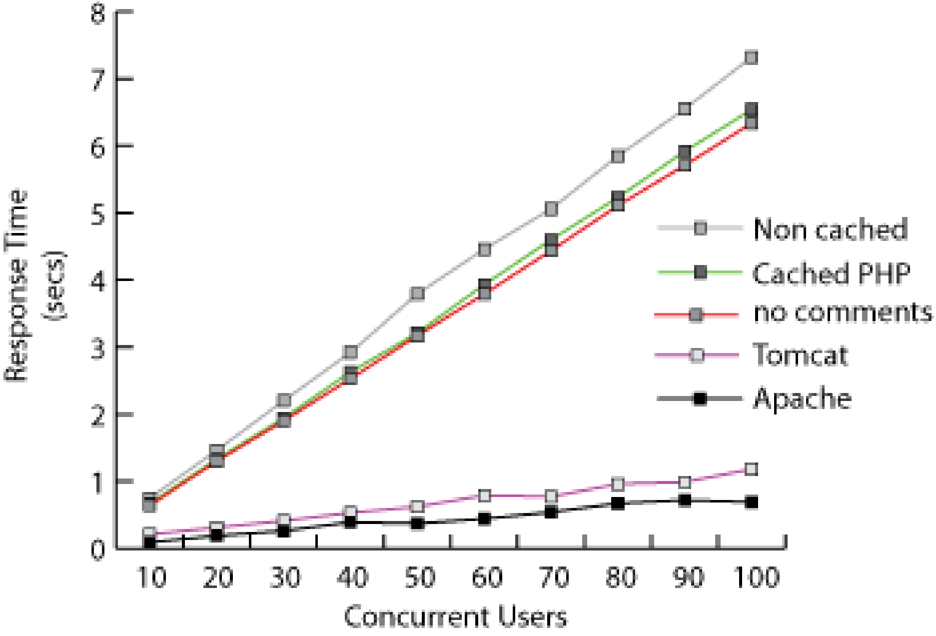
It was clear that exploring other avenues would be more fruitful. After a couple of weeks of trials I ended up with the conclusion that looking at the design of the Paloose and its PHP was the best way to continue. Changing the XML, XSLT and sitemap of a site really did not have as much effect as I wanted.

The next page describes these changes and their implications.

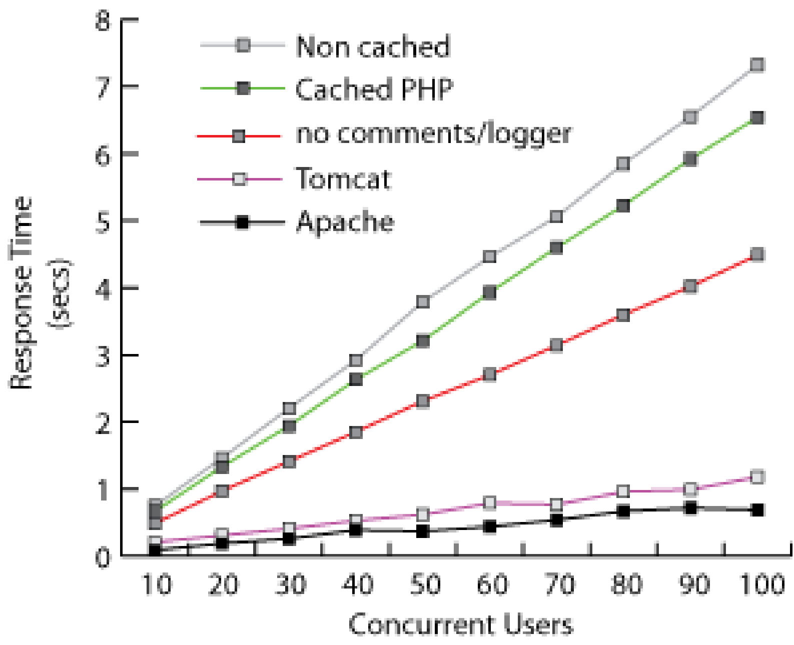
36 Optimising Paloose Performance

36.1 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.



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

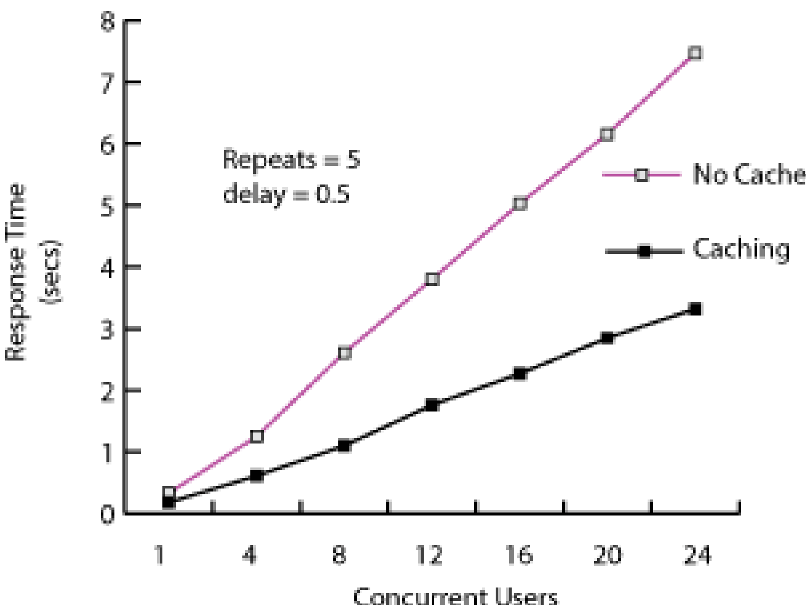


The main reason for this is the amount of code needed to be included from the `log4php` system. In particular every time a *require_once* statement is invoked it degrades the performance, not unexpected.

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

36.2 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 but only two of the stages use it: *FileGenerator* and *TRAXTransformer*. I will add more when I am finally convinced that it is a good idea. The theory is that each stage is responsible for maintaining a cache of the data that it outputs. In the case of a transformer the output is taken from the cache dependent on several things: the input DOM being valid, the XSL file being up to date and the parameters being valid. Valid implies that there has been no change since the last access. I am not wholly convinced that the system is useful except in cases where there are a large amount of transformations to be done — either as a number of sequential transforms of a single large one. For example I tried loading my Hop Vine site catalogue page on a local server and which uses up to 6 transformers to run. The results are relatively encouraging:



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

36.3 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.

37 Caching Discussion

37.1 Introduction

One subject that continues to be brought up is caching and the favorite question: when is it going to be added to Paloose ? It has exercised by mind for several months and I thought it would be worth noting down some of my thoughts about it.

First of all it is worth asking exactly why do we want to use caching. The basic answer is to achieve a speed increase and thus a performance (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 use input or database queries and thus can be safely served by the Apache server.

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

37.2 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.

37.3 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.

37.4 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 check this as a change her will cause all the other conditions to fail. If they do not then the cacheed data can be safely used.
- **The XML fragments (have they changed?)** — The most efficient way of achieving this is to note the latest modifictaion 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 quesry (how may the results have changed?)** — things that are dependant on these types of exteral queries obviously cannot be cached suitably as they are outside control.

37.5 Checks and Balances

Adding a caching sytem 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.

38 How to use the Gallery Transformer

The Gallery transformer provides a simple Picture Gallery addition for Paloose. To see this in action try the gallery at Guinness Park Farm or Chandos Symphony Orchestra.

38.1 Sitemap

The root sitemap needs to have the Gallery Transformer 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 overriden 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"/>
      \textit{<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`. The reason for this will be explained later.

A slightly more expanded use from one of development web site is:

```
<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>
```

38.2 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 the Guinness Park Farm 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>
```

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

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 immedtiate 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.

38.3 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.

38.4 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">resources/images/AlbumIcon.png</xsl:attribut
```

```
                         </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]/@n
                 </xsl:with-param>
                 <xsl:with-param name="inPrevImageName">
                    <xsl:value-of
                       select="//paloose:image[ @name = $thisImageName ]/preceding-sibling::*[1]/@n
                 </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 canbe 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 Guinness Park Farm site. Go there to see how it all turned out :-):

```
.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.

39 How to use the SQL Transformer

Note that this does not match the Coocoon 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> \textbf{use music;}
Database changed
mysql> \textbf{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> \textbf{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 t will suffice.

39.1 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).

39.2 Using the SQL Transformer

39.2.1 A simple query

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

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

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

```
<page:content>

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

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

</page:content>
```

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="music" 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.

39.2.2 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="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/xml2html.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="music">
           select * from composer
           where name = "{param:composer}"
      </query>
     </execute-query>

</page:content>
```

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.

40 How to use the Page-Hit Transformer

The Page-hit transformer provides a very simple mechanism to see how many times a particular page has been accessed. Individual pages can be targetted rather than a complete site. It is also possible to ignore certain IP numbers.

I have found that this transformer fails under some high load conditions. This only happens with continuous loads of 100 concurrent users, but until I find the cause, please treat with caution.

40.1 Sitemap

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

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

      <map:transformer name="pageHit" src="resource://lib/transforming/PageHitTransformer">
         <map:parameter name="file" value="context://logs/PageHit.cnt"/>
         <map:parameter name="ignore" value="127.0.0.1"/>
      </map:transformer>
```

where

- **file** — the actual counter file where the record is kept.
- **ignore** — a comma separated list of IP numbers to ignore.

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

```
<map:match pattern="**.xml">
   <map:generate src="context://content/{1}.xml"/>
   \textit{<map:transform type=‘‘pageHit’’ >
      <map:parameter name=‘‘file’’ value=‘‘context://logs/pp-$\{$1$\}$.cnt‘‘/>
   </map:transform>}
   ...
   <map:serialize/>
</map:match>
```

Note that the parameter *file* overrides the component declaration of *file*, so that we have a counter for each page.

40.2 Content

In order to use the transformer a single element is placed in your content file that you wish to record the page hits.

```
<paloose:page-hit xmlns:paloose="http://www.paloose.org/schemas/Paloose/1.0"/>
```

After running through the transformer this element is replaced by another containing the actual count

```
<paloose:page-hit>6</paloose:page-hit>
```

40.3 Finally

The only thing now is to format it as you want. For example you might have the following in your XSLT transfrmer file:

```
<xsl:if test="//paloose:page-hit">
   <xsl:text>Hits: </xsl:text>
   <xsl:value-of select="//paloose:page-hit"/>
</xsl:if>
```

41 How to use the I18NTransformer for Internationalization

The I18N Internationalization Transformer is similar to tha of the Cocoon transformer but it does not contin the same functionality as the latter, for example parametered text does not exist in Paloose. The following describes the Paloose version.

41.1 Sitemap

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

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

where

- **name** — the name of this selector (in this case *i18n*).
- **src** — the location of the PHP package for this component.

There is a sub-element, `<map:catalogues>`, that defines the the catalogue (translation) files. In this case a single catalogue file is shown, where

- **id** — the id of the catalogue file and its various language files.
- **name** — the name of the catalogue series (they take the form `index_xx.xml`, where *xx* is the language code).
- **location** — the location of the translation files for this catalogue.

The other sub-element, `<map:untranslated>`, defines the text to be output when the appropriate translation cannot be found. We can use the transformer anywhere in the pipeline that any other transformer can be and can have label views associated with it. For example (from this site):

```
<map:match pattern="**.xml">
   <map:generate src="cocoon:/{1}.xml" label="xml-content"/>
   <map:transform type="i18n" label="i18n-transform">
      <map:parameter name="default-catalogue-id" value="i18n-id" />
   </map:transform>
   <map:serialize/>
</map:match>
```

41.2 Building the Catalogue Files

For each language that you require a catalogue file must be written. The format for this file is very simple, for example:

```
<?xml version="1.0" encoding="UTF-8"?>
<catalogue xmlns:link="http://www.hsfr.org.uk/Schema/Link" xml:lang="de">
   <message key="one">ein</message>
   <message key="two">zwei</message>
```

```
    <message key="three">drei</message>
</catalogue>
```

It is effectively a list of messages accessed by key, where

- **key** — the key that is used in the content text to refer to the message.
- **tag content** — the translated text to be used to substitue in the content text.

41.3 Building the Text File

In order to use the transformer there are several tags that indicate what text should be translated.

41.3.1 ¡i18n:text¿

The primary one is <i18n:text> which can be used in two ways. Note that Paloose support multiple dictionaries declared in the <i18n:text> or parameter substitution yet.

1. Using the content text as the key. Using the above catalogue file example we could say

   ```
   <t:p><i18n:text>one</i18n:text></t:p>
   ```

   would translate to

   ```
   <t:p>ein</t:p>
   ```

2. Using an explicit key. Again uUsing the above catalogue file example we could say

   ```
   <t:p><i18n:text key="one"/></t:p>
   ```

   translate to the same XML. It is possible to put in some content which would be output if the locale is not defined. For example

   ```
   <t:p><i18n:text key="one">one (1)</i18n:text></t:p>
   ```

   which would translate to

   ```
   <t:p>one (1)</t:p>
   ```

   locale given.

41.3.2 Attributes

As well as tag content it is possible to translate attributes within tags. For example if have

```
<link:link type="uri" ref="?locale=de_DE">
   <graphic:graphic ref="/pp/resources/images/flag-de.gif"
      name="Germany" i18n:attr="name">De</graphic:graphic>
</link:link>
```

The *name* attribute's value would be translated.

41.3.3 Date and Time Formatting

The date and time formatting is one area that is different to Cocoon in how it deals with the pattern format. Take the two date and time tags which will output the current date and time:

```
<t:p>
   <i18n:date pattern="%A, %e %B %G"/>
</t:p>
<t:p>
   <i18n:time pattern="%X (%Z)"/>
</t:p>
```

The *pattern* attribute uses PHP pattern format character (see PHP manual for a full list).

If a *value* attribute is given then the date/time is taken from that. Note that there is no *src-pattern* attribute to define the format of the value, Paloose uses the PHP parser as best it can to discover the date/time.

41.3.4 Number Formatting

Currently there is no facility for number formatting — I will add this when I find I need it.

42 How do I port Paloose Sites to Cocoon?

It is inevitable that some sites will grow so much in both traffic and functionality that moving the whole thing to Cocoon is sensible. It is obviously important that the minimal amount of work is done in doing this. The whole process is relatively simple with very little in Paloose that cannot be moved to Cocoon. The following is a summary of the important points to remember:

- The component *src* attribute is a Java class in Cocoon (uses package separators), not a file name in Paloose (uses path separators) — (although since there is one class per file it amounts to the same thing).
- Paloose is more relaxed in places about having the *uri-prefix* attribute present or not in the *mount* pipeline element.
- Some transformers in Paloose do not work in Cocoon. For example the Gallery and Page-hit transformers. There is no reason, however, why it should not be possible to write a version in Java of both of these.
- Paloose errors work slightly differently to Cocoon ones, although the differences are not great.

42.1 An example

The example below is a very simple private site that I run to serve my work at home. I have it both in Paloose and Cocoon forms as a test of porting.

42.1.1 Root sitemap

The sitemap component declaration is where the most change must occur. In the Paloose sitemap there is:

```
<map:components>

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

   <map:transformers default="xslt">
      <map:transformer name="xslt" src="\textit{resource://lib/transforming/TRAXTransformer}">
         <map:use-request-parameters>true</map:use-request-parameters>
      </map:transformer>
      <map:transformer name="i18n" src="\textit{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="html">
      <map:serializer name="html" mime-type="text/html" src="\textit{resource://lib/serialization
      <map:serializer name="text" mime-type="text/plain" src="\textit{resource://lib/serializatio
      <map:serializer name="xhtml" mime-type="text/html" src="\textit{resource://lib/serializatio
         <map:doctype-public>-//W3C//DTD XHTML 1.0 Transitional//EN</map:doctype-public>
         <map:doctype-system>http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd</map:doctype
         <map:omit-xml-declaration>yes</map:omit-xml-declaration>
      </map:serializer>
   </map:serializers>

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

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

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

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

</map:components>
```

The Cocoon version uses the same with the *src* attribute changes:

```
<map:components>

   \textit{<map:pipes default=‘‘noncaching‘‘>
      <map:pipe name=‘‘caching’’ src=‘‘org.apache.cocoon.components.pipeline.impl.CachingProcessi
      <map:pipe name=‘‘noncaching’’ src=‘‘org.apache.cocoon.components.pipeline.impl.NonCachingPr
   </map:pipes>}

   <map:generators default="file">
      <map:generator name="file" src="\textit{org.apache.cocoon.generation.FileGenerator}" />
   </map:generators>

   <map:transformers default="xslt">
      <map:transformer name="xslt" src="\textit{org.apache.cocoon.transformation.TraxTransformer}
      <map:transformer name="i18n" src="\textit{org.apache.cocoon.transformation.I18nTransformer}
         <catalogues default="index">
            <catalogue id="index" name="index" location="context://content/translations"/>
         </catalogues>
         <map:untranslated-text>untranslated text</map:untranslated-text>
      </map:transformer>
   </map:transformers>

   <map:serializers default="xhtml">
      <map:serializer name="html" mime-type="text/html" src="\textit{org.apache.cocoon.serializat
      <map:serializer name="text" src="\textit{org.apache.cocoon.serialization.TextSerializer’’ m
      <map:serializer name="xhtml" mime-type="text/html" src="\textit{org.apache.cocoon.serializa
         <map:doctype-public>-//W3C//DTD XHTML 1.0 Transitional//EN</map:doctype-public>
```

```
        <map:doctype-system>http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd</map:doctype
        <map:omit-xml-declaration>yes</map:omit-xml-declaration>
      </map:serializer>
    </map:serializers>

    <map:matchers default="wildcard">
      <map:matcher name="wildcard" src="\textit{org.apache.cocoon.matching.WildcardURIMatcher}"/>
      <map:matcher name="regexp" src="\textit{org.apache.cocoon.matching.RegexpURIMatcher}"/>
    </map:matchers>

    <map:readers default="resource">
      <map:reader name="resource" src="\textit{org.apache.cocoon.reading.ResourceReader}"/>
    </map:readers>

    <map:selectors default="browser">
      <map:selector name="browser" src="\textit{org.apache.cocoon.selection.BrowserSelector}">
        <browser name="explorer" useragent="MSIE"/>
        <browser name="pocketexplorer" useragent="MSPIE"/>
        <browser name="handweb" useragent="HandHTTP"/>
        <browser name="avantgo" useragent="AvantGo"/>
        <browser name="imode" useragent="DoCoMo"/>
        <browser name="opera" useragent="Opera"/>
        <browser name="lynx" useragent="Lynx"/>
        <browser name="java" useragent="Java"/>
        <browser name="wap" useragent="Nokia"/>
        <browser name="wap" useragent="UP"/>
        <browser name="wap" useragent="Wapalizer"/>
        <browser name="mozilla5" useragent="Mozilla/5"/>
        <browser name="mozilla5" useragent="Netscape6/"/>
        <browser name="netscape" useragent="Mozilla"/>
        <browser name="safari" useragent="Safari"/>
      </map:selector>
    </map:selectors>

  </map:components>
```

Obviously in the Cocoon sitemap it is possible to manage the cache, logging and component pools which are not present in Paloose.

42.2 What about the Subsitemaps, XML Content and XSL Transforms?

Absolutely nothing to do here at all. Good isn't it?

Ok, that is not strictly true, on my home web site and this site I had to change the subsitemap "src" attributes. But other than that nothing. However in future the authorisation and continuations that I am currently adding will require a slightly different use of session variables in the XML. When this is added I will detail the differences then. The authorisation (login etc.) is working on the next version and I am finalising the form continuations. Watch this space.

43 Frequently Asked Questions

43.1 About the author

The Paloose poject is currently run by me, Hugh Field-Richards. I recently retired after a (too) long career designing (amongst other things) computer hardware and software. I started Paloose to keep myself amused and to support my other interest, Hop Vine Music.

43.2 What's with the name?

The original name I chose was "Papoose"; in the UK this is defined as "*... a type of bag used to carry a child on the back*". Unfortunately it does not seem have this meaning in the US, indeed, with some people, it is a derogatory term. However, it occurred to me that a simple change of letter would give "Paloose", which is another term for the Appaloosa horse, something dear to my heart as I am privileged to have and ride one of my own.

43.3 What future for Paloose?

Now that I am happy with the initial system I am releasing it under an open-source licence. If anyone wishes to get involved then please EMAIL me. I intend to use it for all my small sites and will continue to do so until Cocoon is supported by the smaller ISPs that I use. It is also interesting (and slightly depressing) that the latest Cocoon does not seem to be as easy to use as Cocoon Version 2.1.x. I use this latter version on my local servers and have no plans (yet) to upgrade to Cocoon Version 2.2 or later. While I am prepared to admit that this is my problem, I know others who have had the same experience. As a result perhaps Paloose will fulfil the requirements of those who want a quick Cocoon-like engine for their small Web sites.

43.4 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 all of what I intended to add is complete. I will use the coming weeks to refine the code and remove extraneous stuff and improve the documentation and comments. I am happy to look at anything that users suggest — just EMAIL me at .

43.5 Do you intend to add PDF support via FOP?

The short answer to this is: no. It would mean writing a complete XSL-fo system in PHP5 which is something I really do not have the time (or inclination) to do. If you really want to produce PDF output then I suggest that you write a suitable XML to LaTeX transformer and then run that to get PDF.

I have written a transformer to produce a paper PDF copy of this Paloose documentation Web site which runs within an Ant build file. It is not particularly difficult to generate XML to LaTeX transformers, although obviously it requires a knowledge of LaTeX. I would be happy to help anyone who has a problem.

43.6 What systems will it run on?

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

43.7 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 current 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 () and my horses have more call on my time.

43.8 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 on a separate page of the documentation in more detail.

43.9 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 be minimal and not really worth the effort except in some key cases, see my caching discussion for more on this. It is not the end of the story but more performance increases will probably come from elsewhere.

43.10 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 PHP). 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 else where. Obviously 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 PHP (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 PHP 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.

43.11 Who designs your own sites that use Paloose?

These have been done by my son Ian Field-Richards who is the GUI art director for DeviantArt.

43.12 Are there examples of sites that use Paloose?

43.12.1 My sites

- The Paloose Site (this one)
- Guinness Park Farm
- Hop Vine Music
- Chandos Symphony Orchestra

43.12.2 Others

- 327 Creative
- Wisconsin Department of Commerce

43.13 How do I port Paloose Sites to Cocoon?

This is very easy with the only major change being the sitemap components. See this How-to for more information. But beware, the Tidy Generator, Password, Gallery and PageHit transformers will not work unless you write your own in Java. I have not needed to yet, but when I do I will post them on this site.

43.14 How do I port Cocoon Sites to Paloose?

This 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. I hope to address this in the near future with some form of caching, but this is in the future just now.

43.15 How do I write Components for Paloose?

I have written an example component here (tgz) that is a "template" for writing generator, transformer and serializer components; also included is an example exceptions class used by the template. Future releases of Paloose will included it in the folder `resources/templates/`. Also have a look at the various existing components that make up Paloose.

43.16 Do you have any support for iPhone?

Yes. The browser selector can detect the iPhone to allow separate pipelines to be run. I have used this on one of my sites, Chandos Symphony Orchestra so try it. The data (XML) is identical in the normal browser and the iPhone browser. It makes having browser sensitive sites relatively trivial.

43.17 Help! I have tried everything but I am still getting page not found.

The usual problem here that you have not set up the `.htaccess` file up correctly. Remember that Apache will serve all your requests, including ones that are destined for Paloose unless you tell it otherwise. For example if you want requests for, say, `http://<hostname>/<site-path>/file.asm`, to be processed by Paloose then you would have to add the following line into your `.htaccess` file:

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

Remember though that the one type of file extension that this method does not handle is PHP, so putting

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

in the `.htaccess` file will cause an infinite loop.

43.18 Help! Why am I getting "Class 'XsltProcessor' not found" errors

99% of the time this shows that you are running PHP5 without the XML/XSL support that there should be. Try recompiling PHP5 with the following configuration parameters included

```
--with-xml --with-libxml-dir=<dir path> --with-xsl --with-tidy
```

`tidy` should be there for the *TidyGenerator*, although the latter is in an early stage of development and is not stable, so treat with caution. There is no problem with PHP5 having the tidy build but just use *TidyGenerator* with care.

43.19 Help! Why am I getting "Parse error: syntax error, unexpected '=', expecting '(' in /../../paloose/lib/Paloose.php on line xx" errors

This means you are still running PHP4. You need to run PHP5. Speak to your ISP to provide PHP5. The other reason may be that your .htaccess file has not been set properly to use PHP5.

43.20 Help! Why am I getting "Input file not found" as the only browser output?

See FAQ entry about the .htaccess file, this is the usual problem. Also make sure that you do not overwrite a system .htaccess that is required. That is why it is best to have your site in a separate folder.

43.21 Are there schemas for the XML used in this site?

Yes the basic schema is written in RELAX NG. They are not an essential to running Paloose or understanding, they are merely given out of interest; although I would urge the use of schemas to reduce errors in your XML files. The primary page schema is page.rng. They are pretty versions of the raw RELAX NG which are easier to read. If anyone would like a copy of the transforms necessary to produce the pretty-printed version please EMAIL me and I will consider putting them up as a download.

43.22 Why do you not use schemas for the sitemaps?

This is a fairly contentious subject. Sitemaps are built in such expandable form that producing a schema in something like RELAX NG or XML-Schema is very difficult — they simply do not have a rich enough structure to do what I think is necessary without having to use Schematron additions. The closest that I got to producing one used DSD-2. I wrote an extensible set of schemas which worked fine, but are now sadly out of date. I also produced schemas for Dublin Core, Jakarta Ant and RDF (although RDF is a major problem because it is impossible to write a universal validating XML schema for it — it ain't the right sort of grammar), all of which can be found on the DSD-2 home page. DSD-2 is the best schema language we never had, sad . . . .

43.23 Technical Trivia

The code that builds Paloose is a little rough and ready in places. This is my first PHP program (other than the odd embedded line within HTML) and it shows. I am more used to program in strongly typed languages having had a lifetime using Algol, Coral, Algol68, Pascal, Java at al. I have programmed in Perl for many years and so PHP was not a wholly new experience. I am sure that I have used 10 lines in places where PHP5 would let me use one; never mind — it suffices, the future will refine the code.