Serving XML with Apache Cocoon

BRING ON THE DANCING GIRLS...

MARKUS KRUMPCK

Everyone has heard of XML, but how can you use it and how do you transfer data to clients that do not speak XML? That is the topic of our current article, where we will be looking into the Apache Groups Web Publishing Framework.

Normally, the first question to be asked is: "Why XML? - Just because it happens to be hyped?"

"And what do all those abbreviations mean anyway: XML, XSL, DTD, and XSLT etc?". Everyone is talking about XML - at least about its look and feel, but nobody quite seems to know what it is used for. XML is short for "eXtensible Markup Language" with the emphasis firmly placed on 'extensible' in contrast to HTML, which (although it has grown with each consecutive version) still comprises a fixed set of permitted tags. So you can define your own tags. Additionally, XML files comprise only textual content and logical characteristics or, more simply, meaning. XSL (eXtensible Stylesheet Language) or XSLT (eXtensible Stylesheet Language Transformations) are required for output. Often this will be performed by using HTML tags within XSL

documents. Again the approach differs from HTML documents where content and formatting instructions are combined in a single file.

Developing Web pages in XML is normally more time consuming than using HTML as you need a document each for content and for output via XSL. You will often see a separate document with tag definitions known as DTD (or Document Type Definition). One advantage of applying logical tags to data is that it allows applications to browse and parse the documents. At the same time the data and its display characteristics are stored separately, allowing different people to work on a document's content or appearance.

Unfortunately, there are very few XML/XSL editors available at present, and that leaves you with very little choice but to fire up your favourite text editor. At the time of writing XML looks like it might just become THE standard data exchange format.

Now you might ask what tools you can use to display XML or XSL documents considering the fact that current browsers provide only limited support (Mozilla) or refuse to comply with specifications (Internet Explorer).

This is the point at which Cocoon enters the scene:

Why Cocoon?

Cocoon is a publishing framework for Web content that is currently under development by Stefano Mazzocchini and others as part of an Apache XML project. Simply put, Cocoon brings XML functionality to the server. In other words Cocoon processes XML/XSL documents, allowing them to be displayed by any client. The client does not even need to be a browser - it could be a mobile phone, for example. By defining different stylesheets for the same content you can change the way a document is displayed by the client. You might need to do this to provide better support for browsers or simply to display content in a different format (XHMTL, HTML, XML, WML, or PDF for instance).

As browsers begin to provide native support for XML/XSL, in future there will be no need to perform conversions of this kind. The data can be displayed natively without the need to perform the intermediate step of converting to HTML. One further advantage is the fact that the same data can be displayed 'on the fly' in multiple formats, and this would be extremely difficult to achieve and support without XML/XSL.

Separating content (which is stored in XML documents), business logic (stored in eXtensible Server Pages) and layout (client-based presentation) - and Cocoon does provide support for this functionality - makes it easier to develop and support complex Web projects. These three areas can be managed independently by different people - no more stepping on each other's feet, provided you keep to the pre-defined interfaces, that is.

By the way, Science Fiction fan and author of Cocoon, Stefano Mazzocchi, was inspired by the movie Cocoon, which was shown on TV while he was working on an idea for an XSL rendering servlet. You may recall that in this movie senior citizens were wrapped in a kind of cocoon before emerging to a new life, just like butterflies.

Functionality

XML documents are processed using an XML parser (Cocoon uses Apache Xalan, which was named after a rare musical instrument and developed as part of the

XML Apache Project) and placed in an internal tree structure known as the Document Object Model (DOM). The data structure can be accessed by one of a whole bunch of processors (XSP processor, SQL processor, LDAP processor and DCP processor amongst others) that processes and manipulates the data following the guidelines of the host applications logic. Following this step an XSLT processor (Apache Xerces) is used to generate output for the clients.

XML 101

XML documents always begin with *<?xml version="1.0"?>* followed by any number of procession instructions (PI) that use the syntax *<?target instruction?>*. Instructions tell the application how to parse i.e. process the document content: For example, *<?cocoon-process type="xsp"?>* or *<?xml-stylesheet href="sample.xsl" type="text/xsl"?>* are typical PIs.

The XML tag and PIs are the only tags that do not need to be terminated by an end tag. This is followed by the root element that comprises any other elements. XML documents must be well-formed, that is, you need to pay attention to proper nesting (the first tag to be opened is the last to be closed). EVERY tag must be terminated by an end tag (in contrast to HTML where tags such as *<hr>*, *
* or *<p>* are often used without an end tag). However, you can define empty tags, that is, tags that do not have any text content but consist entirely of attributes (similar to the *<img src="pict.gif">* tag in HTML).

To avoid having to type tags of this type twice, a tag can be self-terminating, for example:*<author name="Markus Krumpck"/>*. You should also be aware that XML documents are case-sensitive.

A well-formed XML document can be valid, although this is not mandatory. A document is said to be valid if it complies to a Document Type Definition (DTD). DTDs contain structural definitions for a document – for example, they might define what kind of elements can be used in a document. In addition, a DTD provides guidelines for nesting

elements, stipulates the attributes associated with certain elements and specifies the valid attribute values. You can specify a DTD within the XML document itself, although it is more common to refer to an external file. DTDs may be replaced by XML schemas.

Installation

Before you install Cocoon, you must ensure that you have a functional servlet engine. You could use any servlet engine, such as Tomcat. Both the Cocoon manual and Java and XML, Chapter 9 (which can be read online) contain further details. For the purpose of this test we used SuSE 6.4 with JDK 1.1.7 and Apache JServ 1.1, although similar results can be obtained with other distributions.

First and foremost, download the 2.5 MB Cocoon Package (1.7.4) from the Apache XML Project Web site. This package contains classes for Xalan, Xerces and FOP to avoid inconsistencies caused by newer versions of these programs being developed separately from the Cocoon project.

Note: The Cocoon installation guide contains an error at this point, referring to

.../cocoon/BIN/cocoon.properties instead of *.../conf/cocoon.properties*):

```
/etc/httpd/jserv/jserv.conf:
Action cocoon /servlet/org.apache.cocoon.Cocoon
AddHandler cocoon xml
```

After running the commands in the Installation script text box and modifying the files shown in the Configuration files text box, you need to create a directory for Cocoon to store its Java classes in - that is, the user ID under which the web server is running must have write access to the directory - and point to the directory in your configuration files. When I tried to use the directory specified in default configuration, *./respository* (relative to the web server root), I kept on getting *"Can't create store repository: //./repository. [...]"*, although I had assigned global read and write privileges. I finally used an absolute path name for the directory and re-named it to temp to resolve this problem:

```
mkdir -p /usr/local/httpd/htdocs/temp
chmod a+rwx /usr/local/httpd/htdocs/temp

/usr/local/java/cocoon/conf/cocoon.properties:
processor.xsp.repository = /usr/local/httpd/➥
htdocs/temp
```

In /etc/httpd/httdp.conf the line should not contain:

```
LoadModule action module      /usr/lib/apa➥
che/mod_actions.so
```

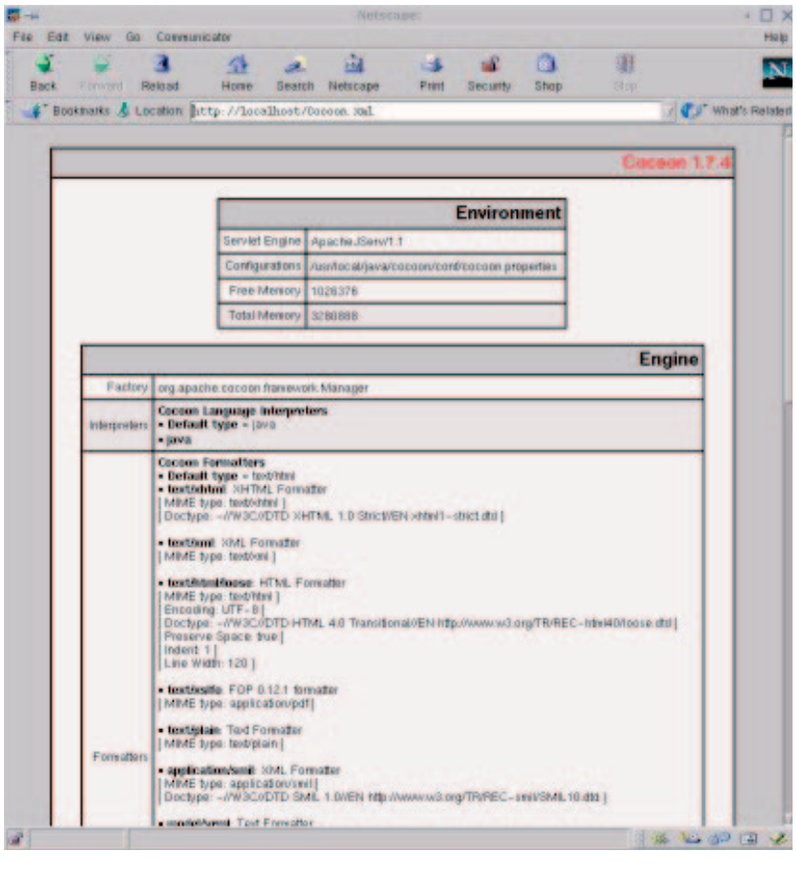
any comment characters - this is normal for SuSE 6.4.

After restarting Apache (*/sbin/init.d/apache restart*) you should be able to type in the following URL *http://localhost/Cocoon.xml* in any browser to display the output shown in the picture.

After using the following command:

```
cp -R /usr/local/java/cocoon/samples/ /usr/➥
local/httpd/htdocs/
```

to copy the sample files that accompany the Cocoon package to the htdocs directory on your Web server, you can also load the following page *http://localhost/samples/index.xml* for an overview of the features that Cocoon has to offer.



Pic.: Cocoon following successful installation

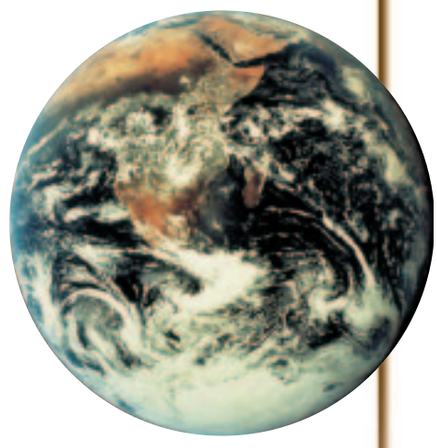
HelloWorld

Let's look at an example - the ubiquitous HelloWorld - to explain things. The example is taken from the sample files that accompany the distribution. (*hello-page.xml* and *hello-page-html.xsl* are just two plain, old XML and XSL documents, except for the special Processing Instruction (*<?cocoon-process type="xslt"?>*) in the XML document that tells Cocoon to pass the document to the XSLT processor, that is, to transform the document. The output format is specified in the stylesheet using:

```
<xsl:processing-instruction name="cocoon-fo➥
rmat">type="text/html"
```

The XML document also contains a sample DTD that was embedded in the document (refer to the lines starting with *<!DOCTYPE page [* through *]>*).

You can generate client-specific output using different stylesheets, the advantage being that you



Installation script

```
tar -xvzf Cocoon-1_7_4_tar.gz
cd cocoon-1.7.4
mkdir -p /usr/local/java/cocoon
cp -R * /usr/local/java/cocoon
cd /usr/local/java/cocoon/lib
ln -s xerces_1_0_3.jar xerces.jar
ln -s xalan_1_0_1.jar xalan.jar
ln -s fop_0_12_1.jar fop.jar
```

Configuration files

```
/etc/httpd/jserv/jserv.properties:
wrapper.classpath=/usr/local/java/cocoon/bin/cocoon.jar
wrapper.classpath=/usr/local/java/cocoon/lib/xerces.jar
wrapper.classpath=/usr/local/java/cocoon/lib/xalan.jar
wrapper.classpath=/usr/local/java/cocoon/lib/fop.jar
/etc/httpd/jserv/zone.properties:
servlet.org.apache.cocoon.Cocoon.initArgs=properties=/usr/local/java/cocoon/conf/cocoo
n.properties
```

hello-page.xml

```
<?xml version="1.0"?>
<?xml-stylesheet href="hello-page-html.xsl" type="text/xsl"?>
<?cocoon-process type="xslt"?>
<!DOCTYPE page [
 <!ELEMENT page (title?, content)>
 <!ELEMENT title (#PCDATA)>
 <!ELEMENT content (paragraph+)>
 <!ELEMENT paragraph (#PCDATA)>
]>
<!– Written by Stefano Mazzocchi "stefano@apache.org" –>
<page>
 <title>Hello>/title>
 <content>
  <paragraph>This is my first Cocoon page!>/paragraph>
 </content>
</page>
```

hello-page-html.xsl

```
<?xml version="1.0"?>
<!– Written by Stefano Mazzocchi "stefano@apache.org" –>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:template match="page">
   <xsl:processing-instruction name="cocoon-format">type="text/html"
   <html>
    <head>
     <title>
      <xsl:value-of select="title"/>
     </title>
    </head>
    <body bgcolor="#ffffff">
     <xsl:apply-templates/>
    </body>
   </html>
  </xsl:template>
<xsl:template match="title">
   <h1 align="center">
    <xsl:apply-templates/>
   </h1>
  </xsl:template>
<xsl:template match="paragraph">
   <p align="center">
    <i>
     <xsl:apply-templates/>
    </i>
   </p>
  </xsl:template>
</xsl:stylesheet>
```

.../conf/coocon.properties:

```
########################################
# User Agents (Browsers)               #
########################################
browser.0 = explorer=MSIE
browser.1 = pocketexplorer=MSPIE
browser.2 = handweb=HandHTTP
browser.3 = avantgo=AvantGo
browser.4 = imode=DoCoMo
browser.5 = opera=Opera
browser.6 = lynx=Lynx
browser.7 = java=Java
browser.8 = wap=Nokia
browser.9 = wap=UP
browser.10 = wap=Wapalizer
browser.11 = mozilla5=Mozilla/5
browser.12 = mozilla5=Netscape6/
browser.13 = netscape=Mozilla
```

clean-page.xml

```
<?xml version="1.0"?>
<?cocoon-process type="xslt"?>
<?xml-stylesheet href="page-xsp.xsl" type="text/xsl"?>
<page>
 <title>First XSP Page</title>
 <p>Hi, I'm your first XSP page ever.</p>
 <p>I've been requested <count/> times.</p>
</page>
```

page-xsp.xls

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:xsp="http://www.apache.org/1999/XSP/Core"
>
<xsl:template match="page">
<xsl:processing-instruction name="cocoon-process">type="xsp"</xsl:processing-instruction>
<xsl:processing-instruction name="cocoon-process">type="xslt"</xsl:processing-instruction>
    <xsl:processing-instruction name="xml-stylesheet">href="page-html.xsl" type="text/xs
l"</xsl:processing-instruction>
<xsp:page language="java" xmlns:xsp="http://www.apache.org/1999/XSP/Core">
<xsp:logic>
       static private int counter = 0;
private synchronized int count() {
         return counter++;
       }
private String normalize(String string) {
         if (string == null) return "";
         else return string;
       }
     </xsp:logic>
<xsl:copy>
       <xsl:apply-templates/>
     </xsl:copy>
</xsp:page>
</xsl:template>
<xsl:template match="title">
   <xsl:copy-of select="."/>
  </xsl:template>
<xsl:template match="p">
   <xsl:copy>
    <xsl:apply-templates/>
   </xsl:copy>
  </xsl:template>
<xsl:template match="count">
   <xsp:expr>count()</xsp:expr>
  </xsl:template>
</xsl:stylesheet>
```

can concentrate your efforts on maintaining a single content file while simultaneously generating HTML pages for multiple browsers.

This allows you to support various vendor-specific HTML, DHTML, or JavaScript versions, or supply a WML page if the client happens to be a WAP compatible mobile phone, and even a PDF document, as we will see in the following section.

The default stylesheet in our example is specified as follows:

```
<?xml-stylesheet href="hello-page-html.xs➥
l" type="text/xsl"?>
```

in our sample document.

You could follow up this line with a sequence like the following:

```
<?xml-stylesheet href="hello-page-ie-html.➥
xsl" type="text/xsl" media="explorer"?>
```

to assign your own template for the Internet Explorer, or use the following line:

```
<?xml-stylesheet href="hello-page-wml.xsl" t➥
ype="text/xsl" media="wap"?>
```

to define a stylesheet for a WAP mobile phone.

You will find a range of values for the attribute *media* in *.../conf/cocoon.properties*

The list is extensible, however, you must pay attention to the order of the list entries, as the MS Internet Explorer (for example) uses the following banner

"Mozilla/4.0 (Compatible; MSIE 4.01; ...)" and would be recognised as Mozilla, if you had omitted an entry to check for the MSIE string.

The client media type is read from the HTTP request header. You can therefore use a simple CGI script that outputs environment variables to discover the exact syntax (this is pre-configured for SuSE 6.4).

Stylesheets are fairly powerful tools, when you consider that they can be used not only for simple formatting tasks but to embed logical constructs that allow you to output entries complying to a given search pattern. A brief explanation is all we have space for:

A stylesheet comprises multiple templates that begin with *<xls:template match="....">* and end with *</xls:template>*. The value of the atttribute *match* defines the tag in the attached XML document that the template is applied to. Starting at the root element (<page> in our case), the XLST processor replaces the content of the XML document with the HTML tag defined in the appropriate template. If the processor finds an occurrence of the *<xsl:apply-templates/>* tag, the children of the root element are also parsed, and then their children, and so on until all the XML tags have been replaced. *<xsl:value-of select="title"/>* is used to output the value of the XML tag *<title>*. XSL allows for a variety of more complex operations, such as *for* loops and *if* constructs. You can define the output order, or output only selected elements, and perform many other useful tasks that are unfortunately beyond the scope of this article.

However, I would like to make the following observation on stylesheets before we move on: Because XSL documents need to be well-formed, just like XML documents, the HTML tags in the stylesheet also have to be well-formed. In fact, you will not actually be generating an HTML page, but an XHTML page. Tags such as *<hr>* for a horizontal line or *
* for a carriage return must appear in their syntactically correct form *<hr />* or *
* in stylesheets. Our example contains an occurrence of this, as you can see by referring to the paragraph tag, *<p>*, whereas HTML will often omit the end tag. You also need to pay special attention to metacharacters such as the ampersand (&). This is normally used to reference so-called entities, that is, text constants that are repeatedly used. XML contains pre-defined entities to resolve this problem:

& for the ampersand, *>* and *<* for greater than and lesser than, *"* for double quotes and *'* for single quotes.

In the case of longer text passages, whose syntactical validity does not need to be checked, we recommend that you place these lines between *<![CDATA[* and *]]>* - CDATA is short for 'character data'.

The W3C web site contains the full specifications of XML and a draft for XSL, although these documents are by no means easy reading and thus not recommended for beginners. If you are looking for a tutorial that is also applicable to Cocoon, take a look at 'Java & XML'.

FOP

XML can (at least in theory) be converted to any format and not only to text-oriented formats, such as HTML or WML. The Formatting Objects Processor (FOP) is required for this task. The processor currently supports conversion to PDF format. Just like in the previous example, you simply need a different stylesheet to convert the content of an XML document into a PDF document. However, we will not be discussing the exact syntax of this special stylesheet at this time. For further information, please review the examples in the Cocoon sample files or take a look at the W3C Consortium web site, where you will find the complete XSL-FO draft.

XSP

Now that we have seen how to define static pages - with the exception of one or two logical constructs that can be defined in XSL stylesheets - it might be a good idea to find out how to generate dynamic output. In the case of Cocoon the

XSP processor (eXtensible Server Pages) is

responsible for this task. Code segments are stored in so-called logic sheets, in contrast to Java Server Pages where code is embedded in normal HTML pages. This means creating at a third file type in addition to XML and XSL documents, but it also means genuine separation of content and business logic. This in turn provides for ease of maintenance, since the files can be managed independently.

To demonstrate this point I have abbreviated one of the files in the Cocoon sample file compendium. The listings *clean-page.xml* (content file), *page-xsp.xml* (business logic) and *page-html.xsl* (layout) are no more than a simple counter that outputs the number of hits for a page, *clean-page.xml*. Using a technique commonly seen with servlets, the number of hits is stored in memory.

On initial access to *clean-page.xml* Cocoon generates and compiles the Java source, which is stored in the repository defined in the configuration file in the same (relative) subdirectory as the corresponding XML document. If you store the XML/XSL documents in the directory, */usr/local/httpd/htdocs/samples/xsp/* then the corresponding Java source or classes will be stored in directory, */usr/local/httpd/temp/_usr/_local/_htdocs/_samples/_xsp/*. At first sight this may seem somewhat complicated, however, if you are working on a large-scale project, you will begin to appreciate the strict separation with its beneficial effects on maintenance tasks. If you are only interested in creating a quick and dirty prototype or demonstrating navigational or layout features, you may prefer to use CGI or PHP.

Summary

This article introduced you to Cocoons XSP processor, however, Cocoon comes with a variety of other processors, such as the SQL processor, or the LDAP processor.

If you require more information on this topic, please refer to the comprehensive documentation included in the Cocoon package.

At time of writing Cocoon2 is under development, although an official release is not yet available. The web site does not provide any clues as to when Cocoon2 might be released, however, I would like to introduce you to some of Cocoon2s features at this point:

Cocoon 1.x is based on DOM1, that is, documents are placed in a hierarchical tree and stored completely in memory after parsing. This can be a problem, especially in the case of large-scale documents. This problem has been resolved by using SAX (Simple API for XML) in Coocon2. SAX parses a document sequentially and triggers the appropriate event when a tag is encountered. In other words only partial or no storage of the document in memory is sufficient.

DCPs (Dynamic Content Pages) have also been dropped in favour of Extensible Server Pages. Additionally, a lot of effort has been put into optimising performance, and you have to admit that Cocoon 1.x is not exactly quick if you need to process a number of requests simultaneously.

At this point I would also like to draw your attention to another interesting project that also deals with XML content generation issues and similarly envisages separating content, logic and style: MxXML. This project was launched by and is under the supervision of Clemens Kerer, Engin Kirda and Roman Kurmanowytsch of the Institute for Distributed Systems at the Vienna Technical University.

A template engine is used to generate Java classes or HTML files from XML/XSL files, allowing them to be integrated in their own servlets. This technique has one major advantage over Cocoon - it allows you to implement business logic within traditional servlets and does not require logic sheets where Java and XML mingle in a single file. This also allows you to continue using existing servlets, only exchanging those parts where HTML code is output for classes generated by MyXML. Classes and HTML files are generated offline, that is, after modifying an XML or XSL file, you must manually launch a Java program that automatically creates the required classes or HTML files. Of course you do not need to re-compile the classes each time you modify the XML content and you can also define dynamic classes (for database applications, for example).

The comprehensive online documentation contains further details. MyXML has already been used to implement a major commercial Web site and has stood up well to field testing so far. ■

Information

Brett McLaughlin: Java and XML - Web Publishing Frameworks
http://www.oreilly.com/catalog/javaxml/chapter/ch09.html
Cocoon:
http://xml.apache.org/cocoon/
W3C: XML-Specifikation
http://www.w3.org/TR/REC-xml
W3C: XSL Working Draft
http://www.w3.org/Style/XSL/
MyXML:
http://www.infosys.tuwien.ac.at/myxml/

■

The Author

Markus Krumpck is a student of Information Technology. Web site programming is his major field of activity. He spends most of his free time playing clarinet. You can contact Markus at markus.krumpoeck@gmx.at

page-html.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Tra?
nsform">
<xsl:template match="page">
<xsl:processing-instruction name="cocoon-format">type="text/html"</xsl:?
processing-instruction>
   <html>
    <head>
     <title><xsl:value-of select="title"/></title>
    </head>
    <body>
    <p><br/></p>
    <center>
      <big><big><xsl:value-of select="title"/></big></big>
      <xsl:apply-templates/>
    </center>
</body>
   </html>
  </xsl:template>
<xsl:template match="title">
   <!— ignore —>
  </xsl:template>
<xsl:template match="p">
   <xsl:copy>
    <xsl:apply-templates/>
   </xsl:copy>
  </xsl:template>
</xsl:stylesheet>
```