AxKit - An XML Publishing Toolkit for Apache

Matt Sergeant - AxKit.com

September 24, 2000

Abstract

AxKit is a set of modules and tools that turn Apache into a complete XML publishing and delivery environment. By integrating XML transformation into the Apache server we get a high performance framework for dynamically generated HTML and other output formats that fits in perfectly with the current Apache configuration methodology. This enables you to install AxKit into your Apache setup and maintain your current webmaster skill sets while introducing the power of semantically rich data using XML.

Contents

1 AxKit Introduction

1.1 What is AxKit?

AxKit was born out of my desire to work with XML for my source data in the form of articles (where I might choose to use DocBook) or FAQ's, or some other data format. I've now been working with XML for nearly three years, and the lack of tools for working with it outside of the Java sphere was disappointing for me. So I decided to at least try and correct that to some extent. Other people's projects have helped immensely too, and AxKit really is the plumbing between many other XML projects and Apache.

AxKit is implemented as an Apache module, written initially in Perl using mod_perl[1], with plans to port the core functionality to C if needed at a later date. It provides content developers with an XML pipeline (which I will describe in more detail later) that progressively renders XML to different formats before finally sending to the user agent. AxKit also allows you to provide different views of the content based on stylesheet selection, or based on user agent media type (the official W3C media types are screen, tty, tv, projection, handheld, print, braille and aural). The engines within AxKit that allow you to transform your XML data sources include XSLT (based on Sablotron from http://www.gingerall.com, or using Perl's XML::XSLT module), XSP (XML Server Pages, a technology originally developed for the Cocoon project), and some custom modules written in Perl that make it easier to work with XML-News and make more sense for Perl developers than XSLT!

In theory AxKit is very much like the Cocoon project in functionality. Since AxKit was developed after Cocoon, it is important to point out the reasoning behind developing such a similar project. The first and foremost reason is that my company doesn't run Java on our development machines. This is simply a "three strikes and you're out" reasoning - we have tried Java development on Linux 3 times in the past, and while we now hear of improvements in the stability of Java on Linux, we have since moved on and are very happy with our current path. The second reason was that we continually heard from the Perl community two chants fairly regularly. The first is "Why don't we have something like Zope?", and the second is "Why don't we have something like Cocoon?". Well now you have the latter and we are working on the former!

It is also important to note that AxKit is built from plug-in modules and components. The main reasoning behind this is an architectural decision that we will eventually be implementing AxKit as a complete configuration management

[1]Many people mistakenly assume mod_perl is a way to speed up CGI scripts for Apache, but in fact it simply provides a means to access the Apache module API from Perl, and allows you to write Apache modules in Perl.

system, very similar in some ways to Zope. Rather than implement this as a CMS with AxKit transformations built in, we built AxKit so that anyone else can drop in their own CMS back end.

1.2 AxKit Implementation

As we already noted, AxKit is developed in mod_perl as an Apache module. This gives us additional Apache configuration directives much in the same way that mod_mime gives us the AddEncoding directive, AxKit provides you with around 16 new directives designed to help configure how Apache then transforms your XML data source to the target user agent.

When a request comes in for a particular URI that AxKit is enabled for, it first checks to see if it should handle the resource. It does so by using some simple checks to determine if the resource is actually XML or not. First it checks for a ".xml" extension. Then it checks the outgoing Content-Type of the resource (since mod_mime will have already been called into action) for an XML content type (see the proposed IETF XML Mime types[2] document). Finally if none of those came true, it checks the actual contents of the file for a leading "<?xml" marker. It does so in a way independent of the document encoding, as described in the XML 1.0 specification.

Once AxKit has determined to process a particular URI, it then goes on to determine how it should be processed. It does so depending on the modules in use for this request, but the default method is to check the <?xml-stylesheet?> processing instructions at the start of the XML document (these instructions appear before the first element, see http://www.w3.org/TR/xml-stylesheet). If no instructions are found there it uses methods defined in the Apache configuration files. The simplest of which defines the appropriate transformation based on the name of the document element[3].

Next AxKit transforms the document using the stylesheet and instructions obtained from the previous step, and finally it delivers the transformed content to the browser or user agent.

1.2.1 Associating Stylesheets/Processors with XML Files

As described above, AxKit can use the <?xml-stylesheet?> processing instruction to associate stylesheets with the XML resource, however this is generally best used as an overriding mechanism, because it doesn't scale particularly well - imagine changing this value in every file on a large web site! So AxKit also

[2]http://www.ietf.org/internet-drafts/draft-murata-xml-07.txt

[3]The document element is the very first element in the XML document after the preamble and the DOCTYPE declaration.

allows you to set the stylesheets based on Apache configuration directives. For example, the following assigns the stylesheet “/stylesheets/docbook_html.xsl” to all files that have the DOCTYPE public identifier “-//OASIS//DTD DocBook XML V4.1.2//EN”:

```
AxAddDocTypeProcessor text/xsl /stylesheets/docbook_html.xsl \
                ``-//OASIS//DTD DocBook XML V4.1.2//EN''
```

AxKit also allows you to do many more complex mappings dependent on the root element name, the DTD file referenced, and also to group the mappings dependent on media type and stylesheet preference.

One thing worth noting is that we have a very simple way to distinguish between a stylesheet transformation (such as XSLT) and a processor (such as XSP, see below). We simply use the stylesheet href of “.”. This initially looks like a completely wrong use of the technology, but it is a very simple way to implement things, and as we know from experience, the way to make things successful and easy to use is to make them simple.

1.3 AxKit Performance

XML transformation can be slow. Very slow. The reasons for this are simple: It requires fairly complex tree traversal and node matching to do complex things with XML documents. This is where you trade off power for efficiency. In order to make sure that your users aren’t held up by deficiencies in the model of XML, AxKit does everything in its power to make things as fast as possible for you.

The basic premise here is to cache everything. In local memory (the memory of each child httpd process) we store the information pertaining to what transformations an XML file has to go through to be processed. We only re-determine those details when something changes. On disk we store the generated output (using a plug-in cache module, should you want some central cache management that doesn’t use files, such as an RDBMS for example). This means that for static files the path through AxKit is as follows:

- Determine if we process the URI
- Determine if anything has changed
- Deliver cached results

As you can see, this is a very short process, and we’re very quickly through to a point where we can say:

```
Apache->request->filename( <cache filename> );
return DECLINED;
```

This provides users of AxKit who have high system demands to deliver a web site that runs at approximately 60% of the speed of a regular Apache server. Since we know that Apache is quite capable of saturating a T1 on a 486, this should be sufficient for almost all sites on the Internet, especially when combined with technology like mod_backhand to provide load balancing over a number of servers.

2 Developing a Web Site Using XML

So now that XML will let you go and sun yourself in Ibiza for half of the year you probably want to know exactly how to go about building that XML based web site. Unfortunately the bad news is that designing and developing an XML based web site will initially increase the time it takes to get your web site online. This is because you now have many extra things to consider. On the other hand, if you are already considering the issues I am about to outline, then you may find this decreases your delivery time, which probably sounds like great news.

2.1 Considerations

When building your new next generation web site, there are several things to consider which prior to now you may have bypassed, saying that it can wait for the future to arrive. Well the future is here, and these are the things you now need to be thinking about for a modern web site:

2.1.1 Accessibility

Up until just a couple of years ago, accessibility meant that you had to ensure your web site looked OK in browsers for the blind, or perhaps in text only browsers such as Lynx. However now it means much more than that. Suddenly in 1998 we had WAP[4] spring onto the scene from a consortium of mobile phone companies. Now while WAP has its faults (and they are many), it is very quickly becoming a de-facto standard for the handheld Internet, with millions of pounds being pumped into WAP development in this country alone.

But it doesn't stop at WAP. Modern web site content needs to be able to be rendered on *Set Top Boxes*. Current practice for this is to either provide totally separate set-top web sites, or to provide work arounds in your HTML

[4]WAP is the Wireless Application Protocol. The term is generally used to describe the the services for delivering content to wireless devices, and include WML, the Wireless Markup Language, and WMLScript a scripting language.

that enable content to stretch to different sized user agents. While this is a good thing, web designers aren't always happy with this lack of control over layout. And the alternate web site issue is a problem too. What happens when you email a URL to your Mother who reads her email on her Sky Digibox? Under AxKit she could just click on the URL and Apache would give her the appropriately styled content.

2.1.2 Changing Content

In order to drive visitors to your site you need content that gets updated on a regular basis. If you want to build an XML based web site without AxKit or any other XML publishing framework, you can do so using command line tools to generate content off-line. But if you have a news oriented site and only FTP access this would start to drive you crazy. AxKit's cache model ensures that content is always up to date from your XML source files, even if your XML source consists of multiple source files using external parsed entities[5].

2.1.3 Internationalization

Internationalization means the support of multiple character sets, or wide character sets. For example, Japanese cannot reasonably be encoded in an ordinary 8 bit character set due to the 256 character limit. XML allows you to utilize the numerous character sets that XML parsers support, which in AxKit's case are generally the most well known character sets, such as the ISO sets, shift-jis (a Japanese encoding) and Big-5 (a Chinese encoding). AxKit by default outputs in UTF-8, which is a character set that supports most of the world's languages using Unicode (and some off-world languages, like Klingon!).

2.1.4 Localization

While this is not an AxKit specific feature, it is very much a feature of Apache. Using content negotiation Apache can deliver content in different languages according to user preference. This process is called localization.

2.1.5 Low Latency and Speedy Delivery

It has been proven that many people are driven away from web sites simply because they are slow to load. This is especially true in Europe where we have relatively little broadband Internet access compared with our Stateside cousins.

[5]External parsed entities are like the SSI includes of the XML world. They allow you to construct a document from various pieces, for example a book would be constructed from various chapter components, each stored in separate files.

In order to make life easier on people's modems AxKit supports dynamic compression of content using gzip. It has been shown that approximately 95% of browsers now support gzip compression, and HTML and other tag based markup languages compress extremely well (up to 20% of the original size for some files). This is very important for those with a slow link. The cache mechanism also means that people get their content very quickly, because the pipeline from request to delivery is very short.

2.1.6 Syndication

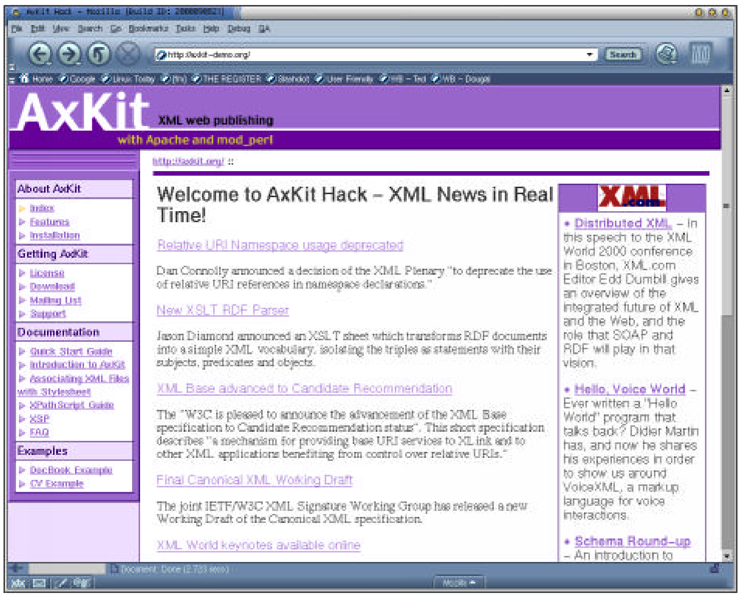
So called *Rich Site Summary*[6] documents are becoming more and more popular on the Internet due to the fact that they are very easy to implement both on the parser side and on the generator side. They are also gaining in popularity due to the work of aggregators such as O'Reilly's Meerkat and sites such as XMLTree.com. Its also a great way to provide your users with more dynamic content, both in bringing content in from off-site and delivering your content to other web sites. While AxKit does nothing special with RSS besides providing some tools to manipulate it, the very use of XML for your content makes generating more XML *about* that content a lot easier.

2.2 Your New XML Web Site

Let us postulate that your web site is a news oriented site, possibly a slashdot-like site with a main headlines page and links off to the actual stories. We need to think about how we would go about such a web site using XML for all the content.

Let us imagine the layout of such a site to be very similar in design to the current Slashdot. This is a very common layout these days, and I'm sure you can come up with something better, but I'm a tools developer, not a web site designer. The design is something like this:

[6]There is a battle underway at the moment over a new RSS 1.0 proposal. This battle may result in a split of the RSS name into two factions. While I am a supporter of the 1.0 proposal I cannot unfortunately predict the outcome at this time.


AxKit Hack - Mozilla {Build ID: 2000090821}
File Edit View Search Go Bookmarks Tasks Help Debug QA
http://axkit-demo.org/
Search
Home Google Linux Today (tm) THE REGISTER Slashdot User Friendly WB - Ted WB - Dougal
Document: Done (2.723 secs)

AxKit XML web publishing with Apache and mod_perl

http://axkit.org/ ::

About AxKit
- Index
- Features
- Installation

Getting AxKit
- License
- Download
- Mailing List
- Support

Documentation
- Quick Start Guide
- Introduction to AxKit
- Associating XML Files with Stylesheet
- XPathScript Guide
- XSP
- FAQ

Examples
- DocBook Example
- CV Example

Welcome to AxKit Hack – XML News in Real Time!

Relative URI Namespace usage deprecated

Dan Connolly announced a decision of the XML Plenary "to deprecate the use of relative URI references in namespace declarations."

New XSLT RDF Parser

Jason Diamond announced an XSLT sheet which transforms RDF documents into a simple XML vocabulary, isolating the triples as statements with their subjects, predicates and objects.

XML Base advanced to Candidate Recommendation

The "W3C is pleased to announce the advancement of the XML Base specification to Candidate Recommendation status". This short specification describes "a mechanism for providing base URI services to XLink and to other XML applications benefiting from control over relative URIs."

Final Canonical XML Working Draft

The joint IETF/W3C XML Signature Working Group has released a new Working Draft of the Canonical XML specification.

XML World keynotes available online

XML.com

- Distributed XML – In this speech to the XML World 2000 conference in Boston, XML.com Editor Edd Dumbill gives an overview of the integrated future of XML and the Web, and the role that SOAP and RDF will play in that vision.
- Hello, Voice World – Ever written a 'Hello World' program that talks back? Didier Martin has, and now he shares his experiences in order to show us around VoiceXML, a markup language for voice interactions.
- Schema Round-up – An introduction to

Many apologies to the XML Hack crew, who I stole the data from for this mock-up.

This of course is a design limited to HTML clients, we also need to think about WAP and probably Web TV. If we are really thinking outside of the box we could extend that to braille or aural (VoXML).

If you have seen a WAP device you know how limited they are. A Nokia 7110, one of the more popular WAP devices in Britain (mostly due to a certain movie...) has only about 4 lines of text display on a 95 x 65 pixel screen. This means you have to totally re-think your content. That tends to mean no banner ads (I'd love to know what the business model is for WAP enabled web sites), and next to no layout. Our site on WAP might look like:

AxKit Hack

Relative URI Namespace usage deprecated
New XSLT RDF Parser
XML Base advanced to Candidate Recommendation
Final Canonical XML Working Draft
More...

And there are your 4 lines before your user has to start scrolling (the line marked "More..." will not appear on your Nokia's screen, it is included here for clarity, something I often wonder if the WAP forum thinks a lot about).

The most important point to see here is that the source of this remains the same. You can even allow the user to scroll down on this headlines page to see your external links (in this case to XML.com), or to your site navigation links.

In short, you can recode your entire site to work transparently on a WAP phone. And the same applies to TV Internet viewers.[7]

2.3 Development

Now we have an initial design in place, we can start to think of this as components, and break it down into separate parts. The main part, is the central headlines. And the two sub-parts are the menu item to the left, and the off-site links to the right. What is interesting about the main headlines and the off-site links is that their content is pretty much the same. In fact we can use the same data model for both. This data model fits neatly into RSS[8]. The RSS file used for the main headlines is also used directly for syndication to other web sites. And it could be generated automatically very simply using a tool such as Jonathan Eisenzopf's XML::RSS Perl module, or just using something like XML::XPath to query the main content pages.

The side menu bar can be a very simple XML vocabulary of your own invention. I'll detail the syntax I have used below, along with a simple bit of code to transform it.

The main content pages could be encoded in a format such as DocBook articles or as a simple form of XHTML. This would allow for complete transformation to viewing on a lower level device, and because you are transforming the content to a more complex form, adding in tables and images to give your preferred layout, you don't lose any control over the formatting.

Transformation happens using either XSLT, or perhaps your developers would choose XPathScript, a language that I invented that is specific to Perl and AxKit, which provides a combination of the power of Perl and XPath along with declarative stylesheet based transformations. To bring all the content together you use the XPath document function[9], and render the different components using named stylesheet templates.

[7]There is some debate in the WAP and alternative device web site development community about the validity of using the same content on such different devices. My take on this is that while development is harder, and some refactoring may be required, the payoffs are large when compared with having two or more separate development teams. One problem here is that manipulating content down to the text level in a language like XSLT is non trivial (XSLT is much better suited to manipulating elements), and we suggest XPathScript as a very good alternative in this case because of the wide variety of tools available in Perl for text manipulation.

[8]Rich Site Summary `http://my.netscape.com/publish/help/`

[9]The document() function is defined in the XSLT specification, but it is a "plugin" to the XPath syntax. It enables you to reference files other than the source XML file using relative or absolute URIs.

2.3.1 XPathScript

XPathScript is a fairly simple language designed for Perl users who want the power of XSLT from a procedural language like Perl. It provides features for declarative template based processing and node resolution using XPath, along with more powerful features for executing Perl code within the template. I do not recommend XPathScript for people who wish to only use standards on their server, however it is a very useful language for web shops with Perl skills, or people who have just found themselves turned off by trying to do something complex in XSLT and found that they end up emulating a procedural language using XSLT constructs.

XPathScript uses the same syntax as Active Server Pages to separate code from output. While this doesn't really fit in with the "XML Way", it is a pragmatic decision based on the availability of editors and other applications that can recognize the <% %> syntax. This makes life a little bit easier for a template editor. We also combined in some function names taken from XSLT, in order for learning both to be that much simpler.

XPathScript can be thought of in two pieces of a puzzle. The first piece is the procedural template, which simply looks like the output you are trying to obtain. This can be intermingled with Perl code using the <% %> syntax. This is useful for transforming XML sources that look more like data than documents, for example a very simple person record:

```
<person>
    <name>
       <firstname>Matt</firstname>
       <lastname>Sergeant</lastname>
    </name>
</person>
```

We can transform that into something simple using the following XPathScript template:

```
<html>
  <head><title>Person</title></head>
  <body>
    Name: <b><%= findvalue('/person/name/lastname') %></b>,
          <%= findvalue('/person/name/firstname') %>
  </body>
</html>
```

The output this produces is:

```
<html>
  <head><title>Person</title></head>
  <body>
    Name: <b>Sergeant</b>,
          Matt
  </body>
</html>
```

The part between the <%= ... %> delimiters is Perl code - a call to the findvalue() function, which locates the nodes in the parameter and returns the string-value of those nodes ("string-value" is a term defined by the XPath specification[10], please see that for more details).

Now imagine for a second that you have a long DocBook document that you would like to transform. It would be extremely difficult to come up with something that can transform XML containing *mixed content* using the procedural template model presented here. What we need is the equivalent to XSLT's declarative templates - a way to specify how specific elements are transformed.

To execute a declarative template in XPathScript you simply call the Perl function apply_templates(). This function can take an XPath expression that specifies a starting point in the source document that you would like to use. From there, the tree structure of the source document is walked in document order[11], and matching templates are looked for each node in the $t hash structure.

The $t hash structure contains keys which are element names. This is slightly different to XSLT, which has templates which are indicated by XPath match expressions. This is a performance vs functionality trade-off. We believe that XSLT will be slightly slower because it has to do an XPath match on every node of the tree as it walks it to try and find a matching template. In XPathScript this is reduced to a simple Perl hash lookup on the element name.

The $t hash also has further depth, indicating what to do with a particular node. Lets assume for a minute that your source document uses a subset of XHTML, and you are using <a> tags for links. If you wish to make all links appear in italics when rendered to HTML, you can use the following declaration:

```
<%
  $t->{'a'}{pre} = '<i>';
  $t->{'a'}{post} = '</i>';
%>
```

[10] XPath Specification at the W3C `http://www.w3.org/TR/xpath`

[11] The order that the nodes appear in the document, also can be seen as a depth-first tree walk.

All we are saying here is that before an <a> tag we add the string '<i>', and after an <a> tag we add the string '</i>'. We can build this up with many more complex expressions using the following possible sub-key's:

- pre
- post
- prechildren
- postchildren
- prechild
- postchild
- showtag
- testcode

The *children sub-keys specify what comes on the very inside of the tags, and the *child sub-keys specify what comes before and after child elements. Showtag is simply a on/off flag to determine if the tag in question gets reproduced on the output and testcode specifies a subroutine reference that will be executed at runtime when that element is encountered. The above is by no means a full example, so please see The XPathScript Guide `http://axkit.org/docs/xpathscript/guide.dkb` for more details.

2.3.2 XSLT

XSLT is the semi-standard[12] option for transforming XML into other formats. Should you be looking at portability to other toolkits, or even skills migration, XSLT would be a good choice for your XML transformation. In AxKit, XSLT is implemented by two modules. The first is the best choice for XSLT transformation in AxKit, Sablotron. This module provides a C based XSLT engine for AxKit, and is extremely quick, capable of real-time XSLT transformations. The other choice is XML::XSLT, a pure perl module implementing only a small subset of XSLT. This is useful for those who cannot get Sablotron to compile on their systems.

Like XPathScript, XSLT can be an extremely in-depth subject, and it is best covered elsewhere than in this document. In contrast, XPathScript is specific

[12]We refer to XSLT as a "semi-standard", rather than standard because the W3C issue recommendations, not standards, and XSLT is not yet widespread enough to be called a *de-facto* standard.

to AxKit, and there will be little other documentation available about it other than that referenced above.

A list of useful XSLT resources on the Internet are:

- Dave Pawson's XSLT FAQ `http://www.dpawson.freeserve.co.uk/xsl/xslfaq.html`
- Zvon's Tutorial Web Site `http://www.zvon.org`
- James Tauber's XSL Info site `http://xslinfo.com`

2.3.3 RSS

RSS stands for "Rich Site Summary". It is a very simple format for syndicating headlines to other web sites; however, you can also use it internally for your own web site's summary page. This will allow you to tag up your headlines and provide a short summary of the main story you are linking to. If you look at Slashdot and the copycat[13] web sites that are cropping up all over the Internet, that is basically what they have on their front pages. And now with the RSS 1.0 proposal, you can extend that specification using XML namespaces to include extra information that might be relevant to your particular setup. Here is the content for our sidebar, taken direct from XML.com's RSS feed `http://xml.com/xml/news.rss`:

```
<?xml version="1.0"?>
<!DOCTYPE rss PUBLIC "-//Netscape Communications//DTD RSS 0.91//EN"
           "http://my.netscape.com/publish/formats/rss-0.91.dtd">
<rss version="0.91">
 <channel>
  <title>XML.com</title>
  <description>XML.com features a rich mix of
information and services for the XML
community.</description>
  <language>en-us</language>
  <link>http://xml.com/pub</link>
  <copyright>Copyright 1999, O'Reilly and Associates
and Seybold Publications</copyright>
  <managingEditor>dale@xml.com (Dale
Dougherty)</managingEditor>
  <webMaster>peter@xml.com (Peter Wiggin)</webMaster>
```

[13]While Slashdot may not have come up with the design, because they appeal to geeks and geeks build web sites, I attribute the popularity of this design to Malda *et-al.*

```
  <image>
   <title>XML.com</title>
   <url>http://xml.com/universal/images/xml_tiny.gif</url>
   <link>http://xml.com/pub</link>
   <width>88</width>
   <height>31</height>
  </image>
  <item>
   <title>Going to Extremes</title>
   <link>http://xml.com/pub/2000/09/13/extremes.html?wwwrrr_rss</link>
   <description>Geeks in tweed and metadata maniacs,
shapers of the future of structured information
representation. The recent Extreme Markup Languages
conference had it all. Liora Alschuler was there and
reports back on the Topic Maps and RDF
head-to-head.</description>
  </item>
  <item>
   <title>XSLT, Comments and Processing
Instructions</title>
   <link>http://xml.com/pub/2000/09/13/xslt/index.html?wwwrrr_rss</link>
   <description>XSLT isn't just for transforming
elements and attributes. In this month's Transforming
XML column we show how to create and transform
processing instructions and comments
too.</description>
  </item>
  <item>
   <title>Gentrifying the Web</title>
   <link>http://xml.com/pub/2000/09/13/xhtml/index.html?wwwrrr_rss</link>
   <description>XHTML promises to civilize the unruly
mass of HTML on the Web. But is anybody listening?
Leigh Dodds examines whether web developers know or
care about XHTML.</description>
  </item>
 </channel>
</rss>
```

When transformed we get a nice list of headlines. The advantage of using this format internally for our main headlines page is that we can immediately offer this content up for syndication - just turn on AxKit's "passthru" plugin, and users can request this with a simple request to the server of http://server/headlines.xml?passthru=1, where headlines.xml is your front headlines page. Sites such as O'Reilly's Meerkat, and XMLTree.com can then load this syndicated content and provide users with different views on the data.

2.4 Dynamic Content

You may be thinking that all of this is terrific for a "mostly" static web site[14], and is not necessarily that great for all the complex things that you like to do. Well we provide support for you there too.

AxKit implements a taglib based XML language called XSP[15]. This allows your developers to design custom tags that your content editors can insert into their code to provide dynamic content. The tags generate *data* not HTML, and so it is unlike many dynamic server side languages that could be named. That data combined with the source XML then goes on to the next stage in the AxKit pipeline, which most likely formats the content to HTML or WML.

XSP allows web site developers to develop a library of tags that implement functionality for a web site. This can be done to the extent that Cold Fusion provides tags for web content delivery, or something very simple, that perhaps says "Good Morning" or "Good Afternoon" depending on the time of day.

XSP works by transforming the taglibs and the parameters "passed" to the taglibs into the "raw" XSP tags, which simply output elements, attributes, processing instructions and comments. It is possible to use these raw tags directly in your XSP page, and this allows you to directly include Perl code in your page, but this is not recommended, as you lose your separation of content from presentation then (unless your Perl code is simply creating tags). The internal implementation currently works on a DOM tree, however the aim in the next few weeks is to convert this to a series of SAX events which will be processed in a parallel-like manner.

The following is a very simple XSP example, that pulls information from a database using the DBI:

```
<?xml version="1.0"?>
<?xml-stylesheet href="." type="application/x-xsp"?>
<?xml-stylesheet href="sql.xsl" type="text/xsl"?>
<xsp:page language="Perl"
    xmlns:sql="http://www.apache.org/1999/SQL"
```

[14]By "mostly" static I mean one where the content is not generated from a database on every hit. This includes something like a news site where content may be updated every 30 minutes.

[15]XSP is a trademark of DataChannel corporation.

```
    xmlns:xsp="http://www.apache.org/1999/XSP/Core"
>
<page title="SQL Search Results">
 <sql:execute-query>
  <sql:driver>Sybase</sql:driver>
  <sql:username>webboard</sql:username>
  <sql:password>password</sql:password>
  <sql:dburl></sql:dburl>
  <sql:doc-element>options</sql:doc-element>
  <sql:row-element>option</sql:row-element>
  <sql:tag-case>lower</sql:tag-case>
  <sql:null-indicator>yes</sql:null-indicator>
  <sql:id-attribute>ID</sql:id-attribute>
  <sql:id-attribute-column>msgid</sql:id-attribute-column>
  <sql:query>select * from Messages</sql:query>
  <sql:count-attribute>count</sql:count-attribute>
 </sql:execute-query>
</page>
</xsp:page>
```

When executed, AxKit picks up the fact that this is to initially be processed by the XSP processor, and the results passed on to the XSLT processor. The XSP processor has an SQL module, modeled after the Cocoon SQL processor, that connects to a DBI database and runs the SQL. The results before transformation using XSLT look like this:

```
<page title="SQL Search Results">
 <options count="84">
  <option ID="1">
   <msgid>1</msgid>
   <forumid>2</forumid>
   <userid>1</userid>
   <parentmsgid>0</parentmsgid>
   <subject>Test post</subject>
   <postdate>May 16 2000 12:55PM</postdate>
   <verified>1</verified>
   <timesread>1</timesread>
   <anonymous>0</anonymous>
   <viaemail>0</viaemail>
   <threadid>1</threadid>
  </option>
  ... <!-- more rows here -->
 </options>
</page>
```

And following that, the XSLT processor transforms the results into HTML for display using an HTML table.

The value in using a technique like this is that your query results still retain rich semantic information - the output format you choose after returning the results from the database is still undecided and flexible. A new taglib for SQL is also in development that allows you much greater control over the generated XML format, and better error handling. Other taglibs exist for forms processing, and form value processing, and more taglibs are in development all the time. But this doesn't stop you building your own...

2.5 Building Your Own Taglibs

Taglibs in XSP can be separated into two separate entities: User taglibs and Builtin taglibs. The builtin taglibs have to be shipped with AxKit, and they have some performance benefits but are much harder to write, because you have to work at the low level of the XSP compiler. User taglibs are merely stylesheets. These stylesheets translate the XML into the raw XSP tags, which then get compiled by the XSP compiler.

Writing taglibs can be a quite complex process, certainly it is not as simple as perhaps writing a bit of Perl code that generates the same output, but the point here is to generate something that your designers can work with, not something for programmers. The balance is the payoff in being able to generate dynamic content while maintaining semantic richness.

As taglibs can grow up to be quite complex, we represent here a very simple taglib for outputting the current date/time in any choice of format (here based on strftime format strings). The taglib is written in XPathScript, simply because we have represented XPathScript in this documentation. It would have been equally simple to do this in XSLT:

```
<%
  $t->{'example:time-of-day'}{testcode} = sub {
    my ($node, $temp_t) = @_;
    # get the value in the format attribute:
    my $format = $node->findvalue('@format');
    # output raw XSP code:
    $temp_t->{pre} = ``<xsp:expr>use POSIX ();
      POSIX::strftime('$format', localtime)</xsp:expr>'';
    return 1;
  };
%>
```

This is our first introduction to the “testcode” aspect of XPathScript, so we will skip over that for now and leave it to the core XPathScript documentation should you wish to know more. Basically, this code translates XML containing the following:

```
<example:time-of-day format=''%H:%M:%S''/>
```

Into the following raw XSP code:

```
<xsp:expr>use POSIX ();
POSIX::strftime('%H:%M:%S', localtime)</xsp:expr>
```

At execution time, this simply generates the string containing the current time according to the format specified and the strftime rules. In this case it would output something like:

```
18:15:36
```

This only gives you a brief introduction to the true power of XSP taglibs. In effect you can build large libraries giving you complete control over custom functionality for your web site, allowing developers and designers to totally separate their concerns - no code will ever output HTML again, and no designer will ever have to chew a developers head off because his design changes can't work in code. Everyone is just working with pure semantic content, and given a fixed schema for that content, content and presentation can finally be separated for good, with both static and dynamic content.

3 XML Editing

Anyone who works on a large content based web site knows that authors don't really work in tags. They work in visual concepts such as bold or italics or section headings. This makes it extremely difficult to convince anyone who is used to producing content in MS Word to move over to editing some new XML format in Emacs. The solution comes in the form of XML editors. While these editors don't tend to provide all of the features of MS Word (is that possible?), they provide an environment that allows your content editors to see what their documents look like, while constraining them to your semantically rich data format.

The current crop of XML editors that work this way include XMetaL from Softquad, Adept from Arbortext, some extensions to popular word processors, such as Corel's built in XML editor in WP9, and other plugins for MS Word.

By using these editors along with mod_dav[16], and Microsoft's Web Folders technology, your editors feel like they are editing in a perfectly natural environment and saving to a network disk. When in fact they are creating semantically rich data which you are harvesting into Rich Site Summaries and they are writing direct to the web.

Obviously in a complex environment you have a lot more to think about, such as content management, but these editors are really the next step towards the semantic web.

4 Future Directions

AxKit is a very new technology. I first started working on the project in May. Already we have a growing mailing list and some live web sites producing content using the AxKit engine. And we have big plans for the future of AxKit. The next step is the content management system which we are building around the framework. This will provide several important features. The first is a very powerful metadata feature, which allows you to add metadata to any resource on your web site. This allows, for example, you to store the number of times a particular URL has been "hit" right in the resource itself. We will also provide a web based editing environment, so that setting up things like new templates and new resource types is much easier, and you will be able to directly assign transformations to those resource types.

We also have big plans for Apache 2.0, which I hope you get to hear lots about at the conference (maybe they will even release it here). Apache 2.0 implements a new filtering mechanism, which allows handlers to pass their content on to the next handler in a chain. This will allow us to throw out some of our current handler chaining mechanisms (though not all of them, see below) and utilize the Apache core directly. This will reduce memory overhead and improve speed.

Finally we are now funding development of a Perl SAX2 implementation so that we can replace the current XSP implementation with a SAX2 based implementation. This work will hopefully be complete by the time you read this paper!

[16]mod_dav is an Apache module implementing WebDAV, the Distributed Authoring and Versioning protocol. It ships with Apache.

A List of Links

- AxKit Web Site `http://axkit.org/`
- Cocoon Web Site `http://xml.apache.org/cocoon/`
- Apache Web Server `http://www.apache.org/httpd.html`
- The Apache and Perl integration project `http://perl.apache.org/`
- Sablotron - C based XSLT processor `http://www.gingerall.com`