AxKit Essentials

Matt Sergeant

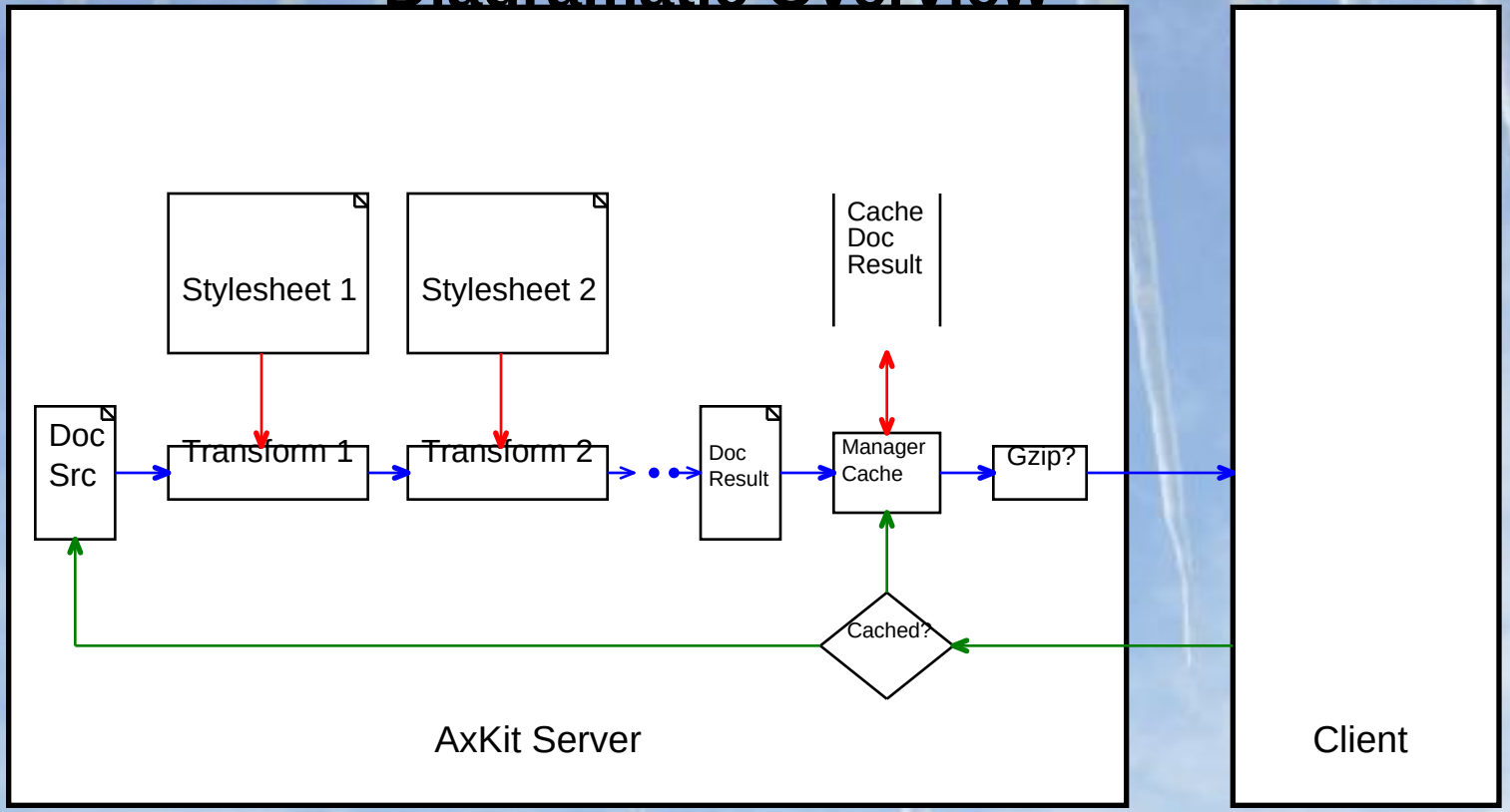
AxKit 101

Everything you always wanted to know about AxKit but were afraid to ask

30,000ft Overview

- AxKit turns Apache into an **XML Application Server**
- Meaning: Server does XML transformations
- Pipeline based - multiple transformations per request
- Transformations can be specialised based on client
- Cache management for performance

Diagramatic Overview



What's in a transformation?

- * XSLT - XML Stylesheet Language - Transformation
- * XPathScript - Combines Perl, ASP, XPath, and some whizzy magic
- XSP - eXtensible Server Pages
- AxPoint
- XSL-FO - XML Stylesheet Language - Formatting Objects
- HTMLDoc - HTML to PDF converter
- Anything else you like

`* means "requires stylesheet"`

Caching

- Smart Caching module
- Caching *must* depend on:
  - XML files involved in the request (source, external entities, included files)
  - Stylesheets involved in the request
  - The client type - web browser, handheld device, etc
  - The *style* applied to this resource
  - Custom parameters
- Caching also works smartly with gzipping of results

The AxKit Dichotomy

- AxKit offers two key things:

- **Static Publishing Engine**
- XML files stored on the server
- Transformation happens at request time, and cached
- Delivery possible in different formats - HTML, PDF, XML
- Useful as part of a content management system...
- ... or just to make building your site easier

- **Dynamic XSP Applications**
- XSP file executed at runtime
- XSP generates XML
- Transformation engine then turns that XML into something usable by the client
- XSP truly allows separation of content from presentation and logic
  - Logic: Perl code
  - Content: XSP files, external XML files, XSP results
  - Presentation: XSLT or XPathScript

XSP

Dynamic Applications

```
# Load AxKit into the server
PerlModule AxKit


##################################################################
## Debugging Options

## Maximum debugging
AxLogDeclines On
AxDebugLevel 10

## Specify the stylesheet to be executed on errors
AxAddStyleMap text/xsl Apache::AxKit::Language::LibXSLT
AxErrorStylesheet text/xsl /axkit/stylesheets/error.xsl
##################################################################
## XSP Options

# Enable AxKit for .xsp files
AddHandler axkit .xsp

## Load the module mapping:
AxAddStyleMap application/x-xsp Apache::AxKit::Language::XSP

## Enable XSP processing for files starting with
## <xsp:page xmlns:xsp="http://apache.org/xsp/core/v1">
AxAddRootProcessor application/x-xsp NULL \
            {http://apache.org/xsp/core/v1}page
```

My First XSP Page

```
The time is now:
        <xsp:expr>
          localtime->strftime("%I:%M%P on %A %e %B, %Y")
        </xsp:expr>
    </page>
  </xsp:page>
```

Results in:

```
  <page>
    <title>My First XSP Page</title>
    The time is now:
      11:15am on Sunday 24 March, 2002
  </page>
```

My First XSP Deconstruction

```
<xsp:page xmlns:xsp="http://apache.org/xsp/core/v1">
  <xsp:structure>
    <xsp:import>Time::Piece</xsp:import>
  </xsp:structure>
  <page>
    <title>My First XSP Page</title>
    The time is now:
      <xsp:expr>
        localtime->strftime("%I:%M%P on %A %e %B, %Y")
      </xsp:expr>
  </page>
</xsp:page>
```

- Code executed when the page is compiled
  - Module importing (equivalent to "use Module")
  - Function definition
  - Page-global variables
- Code executed at request time
- First non-`xsp:foo` tag indicates start of output stream

Things to note

- The source code is XML
  - Means we sometimes have to be careful with < and & signs
- This guarantees that the output will be XML
- The output is not HTML
- So how do we output HTML?

XSP -> XML -> HTML

```
#################################################################
## XSP Options

# Enable AxKit for .xsp files
AddHandler axkit .xsp

## Load the module mapping:
AxAddStyleMap application/x-xsp Apache::AxKit::Language::XSP

## Enable XSP processing for files starting with
## <xsp:page xmlns:xsp="http://apache.org/xsp/core/v1">
AxAddRootProcessor application/x-xsp NULL \
            {http://apache.org/xsp/core/v1}page

## Turn XSP output into HTML using XSLT
AxAddRootProcessor text/xsl /stylesheets/xsp_output_2html.xsl \
            {http://apache.org/xsp/core/v1}page
```

Mix in the XSLT...

```
<xsl:stylesheet
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <head>
        <title><xsl:value-of select="/page/title"/></title>
      </head>
      <body>
        <h1><xsl:value-of select="/page/title"/></h1>
        <xsl:apply-templates match="/page/*[name() != 'title']"/>
      </body>
    </html>
  </xsl:template>
</xsl:stylesheet>
```

But...

- I promised you separation of *logic* from *content*
  - (we can see we separate *content* from *presentation*
- But we're mixing code in on our XSP page!

```
The time is now:
  <xsp:expr>
    localtime->strftime("%I:%M%P on %A %e %B, %Y")
```

- The solution is taglibs

```
The time is now: <datetime:now/>
```

Writing Taglibs - SimpleTaglib

```
package AxKit::XSP::DateTime;
use strict;

use Apache::AxKit::Language::XSP::SimpleTaglib;
use vars qw($VERSION $NS);
$VERSION = 0.90;

# The namespace associated with this taglib.
$NS = 'http://axkit.org/xsp/demo/datetime';

sub now {
    return localtime->strftime("%I:%M%P on %A %e %B, %Y");
}

package AxKit::XSP::DateTime::Handlers;

sub now : expr {
    return 'AxKit::XSP::DateTime::now();';
}

1;
```

- Note: requires Perl 5.6.0

```
package AxKit::XSP::DateTime;
use strict;

use Apache::AxKit::Language::XSP::TaglibHelper;
use Time::Piece;
use vars qw($VERSION $NS @ISA @EXPORT_TAGLIB);
$VERSION = 0.90;

# The namespace associated with this taglib.
$NS = 'http://axkit.org/xsp/demo/datetime';

# Using TaglibHelper:
@ISA = qw(Apache::AxKit::Language::XSP::TaglibHelper);

@EXPORT_TAGLIB = (
    'now()',
);

sub now () {
    return localtime->strftime("%I:%M%P on %A %e %B, %Y");
}

1;
```

```
package AxKit::XSP::DateTime;
use strict;

use Time::Piece;
use vars qw($VERSION $NS @ISA);
$VERSION = 0.90;

# The namespace associated with this taglib.
$NS = 'http://axkit.org/xsp/demo/datetime';
@ISA = qw(Apache::AxKit::Language::XSP);

sub parse_start {
  my ($e, $tag, %attribs) = @_;

  if ($tag eq 'now') {
    $e->start_expr($tag);
    $e->append_to_script(
      'Time::Piece->new->strftime("%I:%M%P on %A %e %B, %Y")'
    );
    $e->end_expr();
  }

  return '';
}

1;
```

```
    xmlns:datetime="http://axkit.org/xsp/demo/datetime"
    >
    ...
```

- Note: `xmlns` declaration **must** match the one in your module!

- Use the taglib

```
<xsp:page xmlns:xsp="http://apache.org/xsp/core/v1"
  xmlns:datetime="http://axkit.org/xsp/demo/datetime">
  <xsp:structure>
    <xsp:import>Time::Piece</xsp:import>
  </xsp:structure>
  <page>
    <title>My First XSP Page</title>
    The time is now: <datetime:now/>
  </page>
</xsp:page>
```

CPAN Taglibs

- Param
- Cookie
- Sendmail
- Session
- Exception
- ESQL
- CharsetConv
- And many more...
- These are generally for "one-off" scripts

```
<xsp:page xmlns:xsp="http://apache.org/xsp/core/v1"
  xmlns:param="http://axkit.org/NS/xsp/param/v1"
>
  <html>
    <head>
      <title>Param Taglib Example</title>
    </head>
    <body>
      <div>
        <form method="POST">
          <input type="text" name="myparam"/>
          <input type="submit" value="Submit"/>
        </form>
      </div>

      <xsp:logic>
        if (<param:myparam/>) {
          <div>
            You entered: <param:myparam/>
          </div>
        }
      </xsp:logic>
    </body>
  </html>
</xsp:page>
```

```
<xsp:page xmlns:xsp="http://apache.org/xsp/core/v1"
  xmlns:param="http://axkit.org/NS/xsp/param/v1"
  xmlns:mail="http://axkit.org/NS/xsp/sendmail/v1"
>
  <html>
    <head>
      <title>Subscribed to axkit-users@axkit.org</title>
    </head>
    <body>
      <xsp:logic>
        if (<param:email/>) {
          eval {
            <sendmail:send-mail>
              <sendmail:from><param:email/></sendmail:from>
              <sendmail:to>axkit-users-subscribe@axkit.org</sendmail:to>
              <sendmail:body>Subscribe to axkit-users</sendmail:body>
            </sendmail:send-mail>
            <div>
              Subscription sent.
            </div>
          };
          if ($@) {
            <div>
              Subscription failed: <xsp:expr>$@</xsp:expr>
            </div>
          }
        }
        else {
          <div>
            Need to enter an email address.
          </div>
        }
      </xsp:logic>
    </body>
  </html>
</xsp:page>
```

Large-scale example

- No flow control. No logic. Simply pointers to what we need to do here.

```
<?xml version="1.0"?>
<xsp:page
    xmlns:xsp="http://www.apache.org/1999/XSP/Core"
    xmlns:f="res:perl/VR/Kernel/FirstTime"
    xmlns:p="res:perl/VR/Kernel/Package"
    language="perl"
>
<response>
    <f:bootstrap-database/>
    <!--
        We install the kernel's package.xml file now because it has
        to run during the request's cleanup process. So we need to get
        everything prepared in time for the actual parameter reading.
    -->
    <p:save-temp-package-from-temp-file install-from-kernel="1"/>
</response>
</xsp:page>
```

- *"This file generates not one, but two databases on two different servers, then bootstraps a few core tables into those databases, then loads up an XML-based package description file into the database. How does it do all this? Who cares! There could be 200 or more lines of code hidden in those calls, with complicated logic and whatnot. This is the beauty of XSP and taglibs: The taglibs do all the freaky complex stuff, with no knowledge that they're in a web environment, and the XSP files just describe what an individual page should do."*

XSP Internals

- What happens when we compile an XSP page?
- What happens when we run an XSP page?
- How does XSP work?
- *We have to save this for another time...*
- *... or hopefully you'll be able to come to San Diego!*

XSP Conclusions

- XSP Taglibs allow true separation of content from logic and presentation
- Taglibs are **easy!**
- XSP can also be for throwaway scripts
- XSP -> XSLT (or XPathScript) means we can redirect our output to different formats

Static Publishing

Server side transformations

```
# Load AxKit into the server
PerlModule AxKit

## Load the module mapping:
AxAddStyleMap text/xsl Apache::AxKit::Language::LibXSLT

##################################################################
## Debugging Options

## Maximum debugging
AxLogDeclines On
AxDebugLevel 10

## Specify the stylesheet to be executed on errors
AxErrorStylesheet text/xsl /axkit/stylesheets/error.xsl
##################################################################
## XSLT Options

# Enable AxKit for .xml files
AddHandler axkit .xml

## Enable XSLT processing for files starting with
## <article> with stylesheet /stylesheets/article_2html.xsl
AxAddRootProcessor text/xsl /stylesheets/article_2html.xsl article

##################################################################
## Caching Options

AxCacheDir /tmp/axkit-cache
```

XSLT Introduction

- XSLT provides XML -> XML transformation based on a stylesheet
- This is **different** to Perl template tools
  - And by that we mean Template Toolkit, Text::Template, HTML::Template, etc
- - those modules provide *data structure* to *text* transformations
- XSLT is a key to the pipeline approach
- Like XSP, XSLT uses XML Namespaces to define "functions"
- XSLT is a recursive declarative functional language!!!

XSLT Example

- First, our source XML:

```
<?xml version="1.0"?>
<book>
<title>Camels: An Historical Perspective</title>
<chapter>
  <title>Chapter One</title>
  <para>
     It was a dark and <emphasis>stormy</emphasis> night...
  </para>
</chapter>
</book>
```

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
    <xsl:template match="/">
      <html>
        <head><xsl:copy-of select="/book/title"/></head>
        <body>
          <xsl:apply-templates/>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="chapter">
      <div class="chapter">
        <xsl:attribute name="id"><xsl:value-of
        select="title"/></xsl:attribute>
        <xsl:apply-templates/>
      </div>
    </xsl:template>
    <xsl:template match="para">
      <p><xsl:apply-templates/></p>
    </xsl:template>
    <xsl:template match="emphasis">
      <em><xsl:value-of select="."/></em>
    </xsl:template>
    <xsl:template match="chapter/title">
      <h2><xsl:value-of select="."/></h2>
    </xsl:template>
    <xsl:template match="book/title">
      <h1><xsl:value-of select="."/></h1>
    </xsl:template>
</xsl:stylesheet>
```

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
    <xsl:template match="/">
      <html>
        <head><xsl:copy-of select="/book/title"/></head>
        <body>
          <xsl:apply-templates/>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="chapter">
      <div class="chapter">
        <xsl:attribute name="id"><xsl:value-of
        select="title"/></xsl:attribute>
        <xsl:apply-templates/>
      </div>
    </xsl:template>
    <xsl:template match="para">
      <p><xsl:apply-templates/></p>
    </xsl:template>
    <xsl:template match="emphasis">
      <em><xsl:value-of select="."/></em>
    </xsl:template>
    <xsl:template match="chapter/title">
      <h2><xsl:value-of select="."/></h2>
    </xsl:template>
    <xsl:template match="book/title">
      <h1><xsl:value-of select="."/></h1>
    </xsl:template>
</xsl:stylesheet>
```

```
<book>
<title>Camels: An Historical
Perspective</title>
<chapter>
  <title>Chapter One</title>
  <para>
  It was a dark and
  <emphasis>stormy</emphasis>
  night...
  </para>
</chapter>
</book>
```

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
    <xsl:template match="/">
      <html>
        <head><xsl:copy-of select="/book/title"/></head>
        <body>
          <xsl:apply-templates/>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="chapter">
      <div class="chapter">
        <xsl:attribute name="id"><xsl:value-of
        select="title"/></xsl:attribute>
        <xsl:apply-templates/>
      </div>
    </xsl:template>
    <xsl:template match="para">
      <p><xsl:apply-templates/></p>
    </xsl:template>
    <xsl:template match="emphasis">
      <em><xsl:value-of select="."/></em>
    </xsl:template>
    <xsl:template match="chapter/title">
      <h2><xsl:value-of select="."/></h2>
    </xsl:template>
    <xsl:template match="book/title">
      <h1><xsl:value-of select="."/></h1>
    </xsl:template>
</xsl:stylesheet>
```

```
<book>
<title>Camels: An Historical
Perspective</title>
<chapter>
  <title>Chapter One</title>
  <para>
  It was a dark and
  <emphasis>stormy</emphasis>
  night...
  </para>
</chapter>
</book>
```

```
<?xml version="1.0"?>
<xsl:stylesheet
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  version="1.0">
    <xsl:template match="/">
      <html>
        <head><xsl:copy-of select="/book/title"/></head>
        <body>
          <xsl:apply-templates/>
        </body>
      </html>
    </xsl:template>
    <xsl:template match="chapter">
      <div class="chapter">
        <xsl:attribute name="id"><xsl:value-of
        select="title"/></xsl:attribute>
        <xsl:apply-templates/>
      </div>
    </xsl:template>
    <xsl:template match="para">
      <p><xsl:apply-templates/></p>
    </xsl:template>
    <xsl:template match="emphasis">
      <em><xsl:value-of select="."/></em>
    </xsl:template>
    <xsl:template match="chapter/title">
      <h2><xsl:value-of select="."/></h2>
    </xsl:template>
    <xsl:template match="book/title">
      <h1><xsl:value-of select="."/></h1>
    </xsl:template>
</xsl:stylesheet>
```

```
<book>
<title>Camels: An Historical
Perspective</title>
<chapter>
  <title>Chapter One</title>
  <para>
  It was a dark and
  <emphasis>stormy</emphasis>
  night...
  </para>
</chapter>
</book>
```

Results

```
<?xml version="1.0"?>
<html>
  <head>
    <title>Camels: An Historical Perspective</title>
  </head>
  <body>
    <h1>Camels: An Historical Perspective</h1>
    <div class="chapter" id="Chapter One">
      <h2>Chapter One</h2>
      <p>
         It was a dark and <em>stormy</em> night...
      </p>
    </div>
  </body>
</html>
```

Things to note

- Result is cached
- Cache invalidated if XSLT or XML changes
- There may be further XSLT transformations made after this one
  - For example: add a table of contents
- Could use XPathScript instead of XSLT
  - (but that's another talk :-)

Further Resources

- The AxKit Web site: http://axkit.org/
- W3C XSLT Spec: http://www.w3.org/TR/xslt
- Zvon - XML Technologies Tutorials: http://www.zvon.org/
- The XSLT FAQ: http://www.jenitennison.com/xslt/index.html
- XML Projects @ Apache.org: http://xml.apache.org/
- libxml2 and libxslt: http://www.xmlsoft.org/
- Sablotron XSLT: http://www.gingerall.com/
- AxKit XSP Taglibs: http://search.cpan.org/search?mode=module;query=AxKit%3A%3AXSP