METRIX
INTERNET DESIGN GMBH

XML transformations with PHP

PHPCon 2003 East

April 25th 2003, New York City, USA

Stephan Schmidt

Agenda

- About the speaker
- XML transformations
- Drawbacks of XSLT
- Transforming XML with PHP
- Installation patXMLRenderer
- Adding fun by overloading XML namespaces
- Design and usage of extensions
- PEAR::XML_Transformer
- XSLT vs PHP Transformers

Stephan Schmidt

- Web Application Developer at Metrix Internet Design GmbH in Karlsruhe/Germany
- Programming since 1988, PHP since 1998
- Publishing OS on http://www.php-tools.net
- Contributor to the German PHP Magazine
- Regular speaker at conferences
- Maintainer of patXMLRenderer, patTemplate, patUser and others

XML transformations

- Data is stored in an XML document
- Needed in different formats and environments
  - Other XML formats (DocBook, SVG, ...)
  - HTML
  - Plain text
  - LaTeX
  - Anything else you can imagine
- Content remains the same

Transform XML to XML

Source document:

```
<example title=„My Example“>
  <greeting>
        Hello <imp>world</imp>!
  <greeting>
</example>
```

Result of transformation into a different XML format:

```
<document>
    <title>My Example</title>
    <section>
        <para>
            Hello <mark type=„important“>world</mark>!
        </para>
    </section>
</document>
```

Transform XML to HTML

Result of transformation to HTML:

```
<html>
    <head>
        <title>My Example</title>
    </head>
    <body>
        <h1>Hello <b>world</b></h1>
    </body>
</html>
```

Transform XML to plain text

Result of transformation to plain text:

My Example

Hello W O R L D !

Transform XML to LaTeX

Result of transformation to LaTeX

```
\documentclass{article}
\title{My Example}

\begin{document}
Hello {\em World}!
\end{document}
```

Introduction to XSLT

- XSLT has been developed for the task of transforming XML documents
- XSLT stylesheets are XML documents
- Transforms XML trees that are stored in memory
- Uses XPath to access parts of a document
- Based on pattern matching
  ("When see you something that looks like this, do that…")
- "awk of the XML universe"
- May only append to its output
- Functional syntax

„Hello World“ Example (XML)

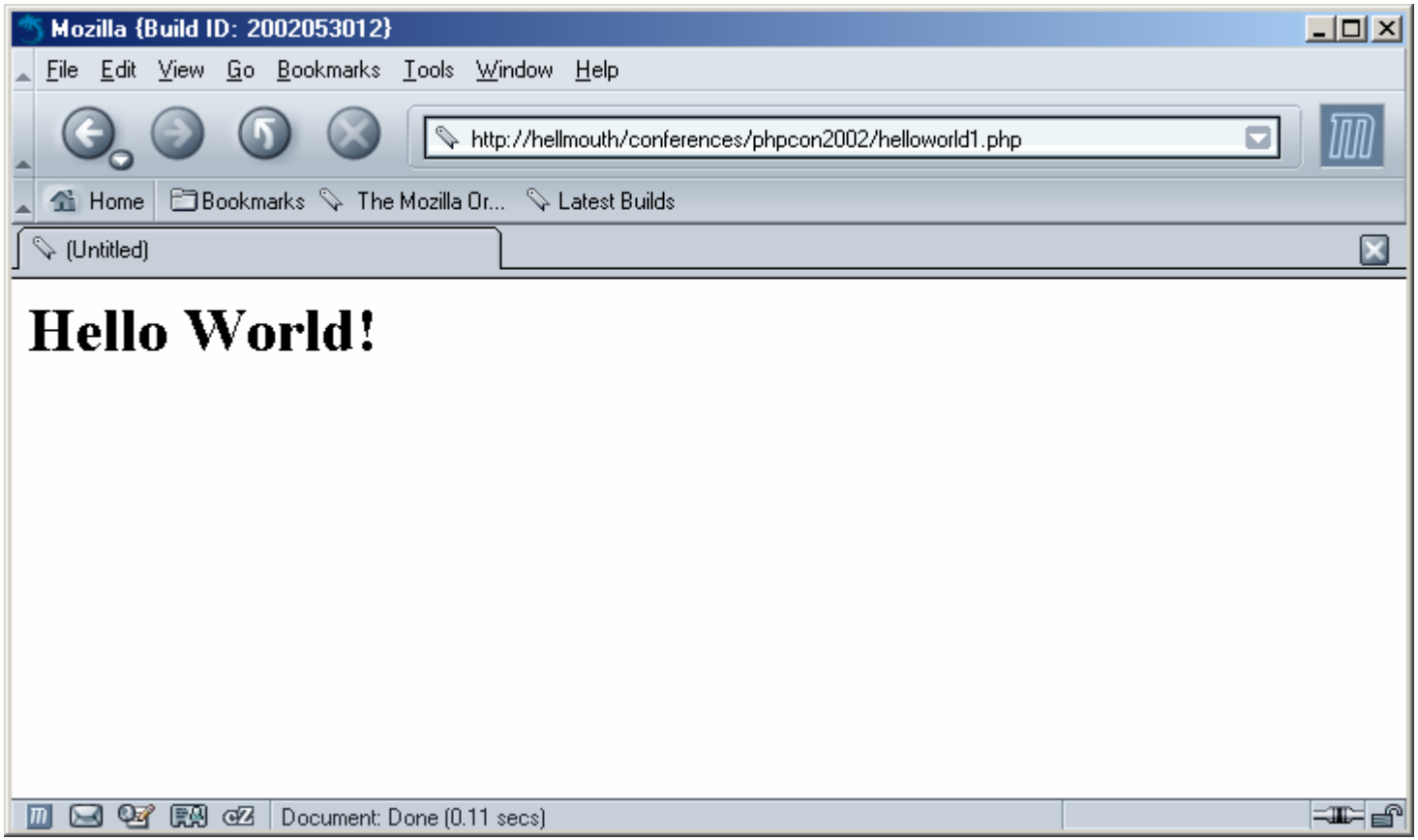
Sample Document:

```
<?xml version="1.0"?>
<greetings>
    <greeting>Hello World!</greeting>
</greetings>
```

„Hello World“ Example (XSL)

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="html"/>
  <xsl:template match="/">
    <html>
      <body>
        <xsl:apply-templates select="greetings/greeting"/>
      </body>
    </html>
  </xsl:template>
  <xsl:template match="greeting">
    <h1>
      <xsl:value-of select="."/>
    </h1>
  </xsl:template>
</xsl:stylesheet>
```

„Hello World“ Example (Output)



Drawbacks of XSLT

XSLT is domain specific:

- Developed to work with XML
- Creating plain text/LaTeX is quite hard, as whitespace is important (<xslt:text>)
- Transforming “world” to “W O R L D” is next to impossible

Drawbacks of XSLT

XSLT is verbose and circumstantial:

```
<xsl:choose>
  <xsl:when test="@author">
    <xsl:value-of select="@author"/>
    <xsl:text> says: </xsl:text>
    <xsl:value-of select="."/>
  </xsl:when>
  <xsl:otherwise>
    <xsl:text>Somebody says: </xsl:text>
    <xsl:value-of select="."/>
  </xsl:otherwise>
</xsl:choose>
```

Drawbacks of XSLT

XSLT is hard to learn:

- Functional programming language
- Complex structure (see “if/else” example”)
- XPath is needed
- Designer needs to learn it

Drawbacks of XSLT

XSLT is not up to the task of creating up-to-date websites:

- Not able to re-transform its own output
- XSL stylesheet is not able to manipulate itself
- Not able to query databases , include files or even the current date (even LaTeX is able to do this with “\today”)

So, why should I use XSLT?

Why use XSLT?

You should use XSLT when…

- …you are transforming XML to XML or XHTML.
- …you need a portable application.
- …want to build a W3C compliant website.
- …you have too much free time and like hurting yourself

Otherwise, you should do the transformation on your own.

Transforming XML using PHP

Transforming an XML document is easy:

1. Define a transformation rule for each tag
2. Start at the root element
3. Traverse the document recursively
4. Insert the transformation result to the parent tag
5. Go home early as you have completed the task faster than with XSLT.

Creating transformation rules

Rules are simple:

- “When you see this, replace it with that.”
- Implemented in PHP using templates
- Attributes of the tag are used as template variables
- PCData of the tag is used as template variable “CONTENT”

Example

XML

```
<section title="XML Transformation">
    <para>XML may be transformed using PHP.</para>
</section>
```

Template for <section>

```
<table border="0" cellpadding="0" cellspacing="2" width="500">
        <tr><td><b>{TITLE}</b></td><tr>
        <tr><td>{CONTENT}</td></tr>
</table>
```

Template for <para>

```
<font face="Arial" size="2">{CONTENT}<br></font>
```

Example (Result)

```
<table border="0" cellpadding="0" cellspacing="2"
   width="500">
   <tr><td><b>XML Transformation</b></td><tr>
   <tr><td>
        <font face="Arial" size="2">
        XML may be transformed using PHP<br>
        </font>
   </td></tr>
</table>
```

Implementing the transformer

Use a template class to “implement the rules”:

- Create a template for each tag
- Include placeholders for attributes and content of the tag

Make use of the internal SAX parser:

- Simple callbacks for events (start tag, end tag, cdata)
- Traverses the document recursively
- Template is parsed when closing tag is found
- Result is appended to the content of the parent tag

Don’t reinvent the wheel

There already are XML transformers available for PHP:

- patXMLRenderer
  http://www.php-tools.net
- PEAR::XML_Transformer
  http://pear.php.net
- phpTagLib
  http://chocobot.d2g.com

Installation of patXMLRenderer

- Download archive at http://www.php-tools.de
- Unzip the archive
- Adjust all path names and options in the config file (cache, log, etc.)
- Create the templates (transformation rules)
- Create your XML files
- Let patXMLRenderer transform the files

Finished: » It's mere child's play «

Using patXMLRenderer

```
$randy = new patXMLRenderer;          // instantiate renderer
$randy->setExtensionDir( $extDir );   // set directory for extensions
$randy->setKnownExtensions( $knownExtensions ); // set list of extensions
$randy->setSkins( $skins );           // set list of available skins
$randy->selectSkin( "blue" );         // select one skin
$randy->setOption( "autoadd", "on" ); // enable auto add for extensions
$randy->setOption( "cache", "auto" ); // enable caching
$randy->setXMLDir( $xmlDir );         // set directory for all xml files
$randy->setXMLFile( $file );          // file to parse
$template = new patTemplate();        // template object
$randy->setTemplate( $template );

foreach( $baseTemplates as $tmplFile ) // load templates
    $randy->addTemplateFile( $tmplFile );

$randy->initRenderer();               // do some init routine
$randy->displayRenderedContent();     // transform and display (echo) result
```

Introduction to patTemplate

- PHP templating class published under LGPL
- Placeholder for variables have to be UPPERCASE and enclosed in { and }
- Uses <patTemplate:tmpl name="..."> tags to split files into template blocks that may be addressed seperately
- Other Properties of the templates are written down as attributes, e.g: type="condition" or whitespace="trim"
- Emulation of simple switch/case and if/else statement by using <patTemplate:sub condition="..."> tags

patTemplate Example

simple Template with two variables

(Corresponds to the XML tag <box>)

```
<patTemplate:tmpl name="box">
<table border="1" cellpadding="5" cellspacing="0" width="{WIDTH}">
    <tr>
        <td>{CONTENT}</td>
    </tr>
</table>
</patTemplate:tmpl>
```

patTemplate Example 2

Task:

Box should be available in three sizes: “small”, “large” and “medium” (default)

Solution:

Condition Template to emulate a switch/case statment:

- Template type is "condition"
- Variable that should be checked is called "size"
- Three possible values for "size": "small", "large" and "medium" (or any other unknown value)

  » three Subtemplates.

patTemplate Example 2

```
<patTemplate:tmpl name="box" type="condition" conditionvar="size">
    <patTemplate:sub condition="small">
        <table border="1" cellpadding="5" cellspacing="0" width="200">
            <tr><td>{CONTENT}</td></tr>
        </table>
    </patTemplate:sub>
    <patTemplate:sub condition="large">
        <table border="1" cellpadding="5" cellspacing="0" width="800">
            <tr><td>{CONTENT}</td></tr>
        </table>
    </patTemplate:sub>
    <patTemplate:sub condition="default">
        <table border="1" cellpadding="5" cellspacing="0" width="500">
            <tr><td>{CONTENT}</td></tr>
        </table>
    </patTemplate:sub>
</patTemplate:tmpl>
```

Adding fun

Simple transformation is easy, but patXMLRenderer can do more:

- Callbacks for events (start tag, end tag, data) can be dispatched to classes (dubbed “extensions”) that are responsible for the transformation of the tag.
- Extensions may access any PHP function
- Extensions may return XML that is still transformed

Overloading namespaces

Extensions are added by overloading namespaces:

- On each event patXMLRenderer checks whether the current tag has a namespace
- patXMLRenderer checks whether an extension for the namespace has been defined
- patXMLRenderer instantiates the extension class (if not done already)
- patXMLRenderer calls the appropriate method of the class (startElement, endElement, characterData)
- Extension returns transformed tag and patXMLRenderer appends it to parent tag (or transforms it again)

Simple Example

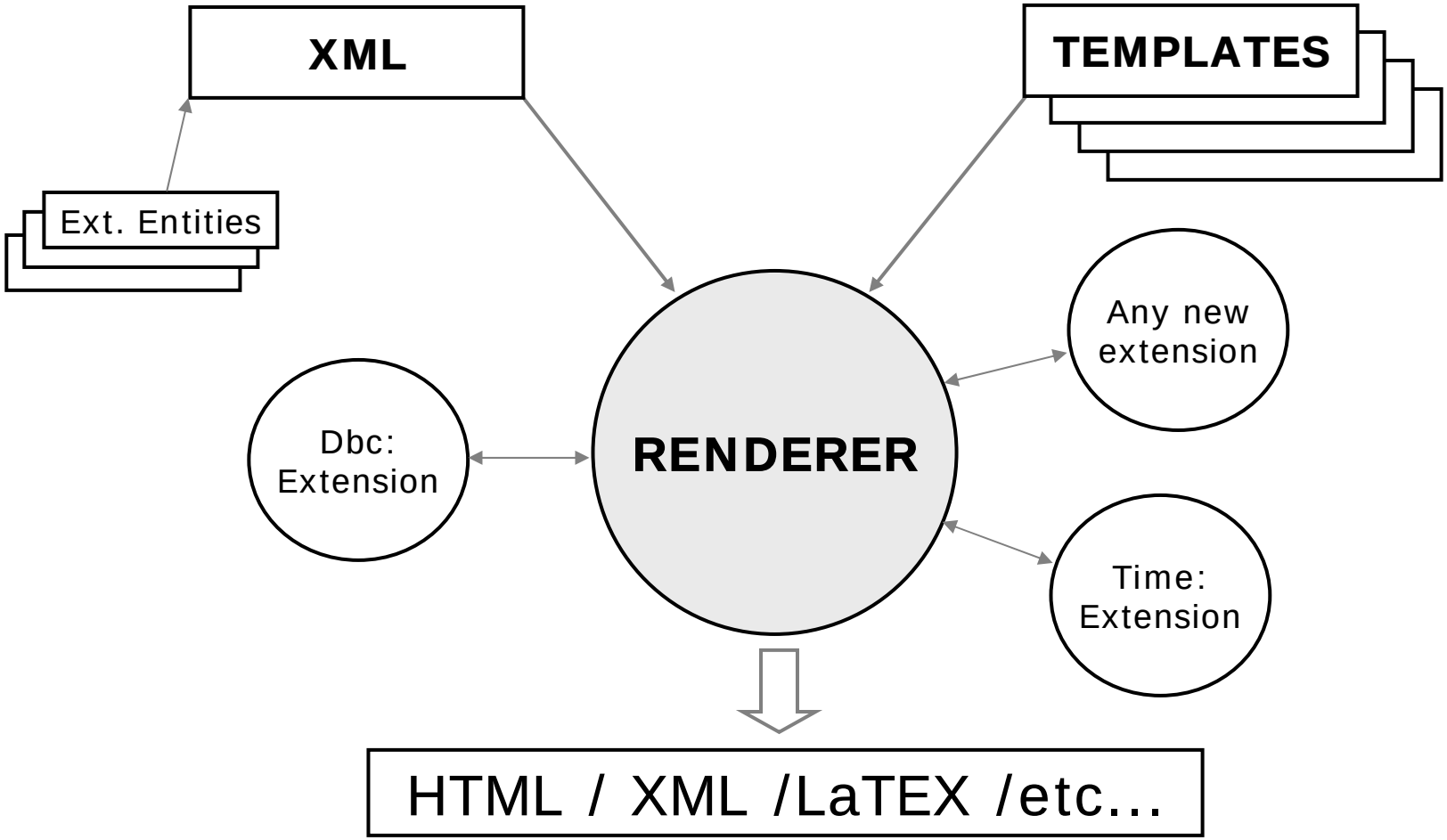
```
<example>
    Today is <time:current format="m-d-Y"/>.
</example>
```

Will be transformed to…

```
<example>
    Today is 04-25-2004.
</example>
```

…Which will then be transformed using the rules defined in the templates.

Architecture of patXMLRenderer



patXMLRenderer Example

```
<page>
  <dbc:connection name="foo">
    <dbc:type>mysql</dbc:type>
    <dbc:host>localhost</dbc:host>
    <dbc:db>myDb</dbc:db>
    <dbc:user>me</dbc:user>
    <dbc:pass>secret</dbc:pass>
  </dbc:connection>
        ...place any XML code here...
  <dbc:query connection="foo" returntype="assoc">
    SELECT id,name,email FROM authors
    WHERE id=<var:get scope="_GET" var="uid"/>
  </dbc:query>
<page>
```

patXMLRenderer Example (Result)

```
<page>
        ...any static XML...
  <result>
    <row>
      <id>1</id>
      <name>Stephan</name>
      <email>schst@php-tools.de</email>
    </row>
  </result>
</page>
```

patXMLRenderer example

```
<control:switch>
  <control:expression><var:get scope="HTTP_GET_VARS" var="agree" /></control:expression>
  <control:case>
    <control:expression>yes</control:expression>
    <control:body>
      <para>Oh, It's nice that you agree!</para>
    </control:body>
  </control:case>
  <control:case>
    <control:expression>no</control:expression>
    <control:body>
      <para>Mmh, maybe we should work harder to convince you!</para>
    </control:body>
  </control:case>
  <control:default>
    <control:body>
      <para>You should set the value to 'yes' or 'no'. I don't think that's too hard!</para>
    </control:body>
  </control:default>
</control:switch>
```

Existing Extensions

- Repository on http://www.php-tools.net
- Documentation is generated automatically
- Examples:
  - » **<xml:...>** for XML highlighting
  - » **<dbc:...>** database interface
  - » **<var:...>** access to variables
  - » **<control:...>** control structures
  - » **<randy:...>** XML-API for patXMLRenderer
  - » **<formgen:...>** for creating (HTML) forms
  - » **<rss:...>** to include content from RSS feeds
  - » **<file:...>** file operations
  - » and many more...

Getting bored?

Let's take a look at some real-life examples...

Creating new extensions

Creating new extensions is easy:

- Create a new folder in the extensions directory
- Create a new PHP file containing the extension class
- Create a new extension class by deriving it from the “patXMLRendererExtension” class
- Implement at least a handler for the endElement
- Create a new XML file that contains documentation about your extension
- Open the administration interface of patXMLRenderer and add the extension

The extension class

```
class patXMLRendererPHPConExtension extends patXMLRendererExtension
{
    var   $name       = "patXMLPHPConExtension";
    var   $version    =  "0.1";
    var   $cacheAble  =  array(
                              "RATE"        =>        true
                              );
}
```

The endElement handler

```
function  endElement( $parser, $ns, $tag )
{
  $data       = $this->getData();
  $tag        = array_pop( $this->tagStack );
  $attributes = array_pop( $this->attStack );

  switch( $tag )
 {
    case  "RATE":
      return array(
                   "data"              => "I think PHPCon 2003 is " . $data,
                   "containsMarkup" => false
                   );
      break;
  }
}
```

The XML file

```
<configuration>

  <!-- information about the extension -->
  <path name="extension">
          <configValue type="string" name="namespace">phpcon</configValue>
          <configValue type="string" name="name">PHPCon Extension</configValue>
          <configValue type="string" name="class">patXMLRendererPHPConExtension</configValue>
          <configValue type="float" name="version">0.1</configValue>
          <configValue type="string" name="requiredRandyVersion">0.6</configValue>
          <configValue type="string" name="description">
                  Just an example for the PHPCon 2003 East in NYC
          </configValue>
    </path>

    <!-- information about the author -->
    <path name="author">
          <configValue type="string" name="name">Stephan Schmidt</configValue>
          <configValue type="string" name="email">schst@php-tools.net</configValue>
    </path>

    <!-- documentation of the tags would follow here -->

</configuration>
```

patXMLRenderer features

More features of patXMLRenderer include:

- Automated, intelligent caching
- Use of external entities (or xinc) to include files
- Administration interface
- <?PHP to include PHP code in XML files
- Parameters in XML attributes
- Offline Generator, generates static HTML pages

PEAR::XML_Transformer

- Developed by Kristian Köhntopp and Sebastian Bergmann
- Only used to overload namespaces or tags with Objects or functions
- No usage of a templating class, transformation from XML to HTML has to be done in PHP classes
- Existing Namespace handlers
  - Simple image generation
  - DocBook to HTML transformation
  - Widget, to create HTML Widgets
  - PHP, to define your own tags or eval PHP code

XML_Transformer example

```
<html>
  <body>
    <img:gtextdefault bgcolor="#888888" fgcolor="#000000"
    font="cour.ttf" fontsize="32" border="1" spacing="2"
    cachable="yes"/>
    <img:gtext>My Example</img:gtext>
  </body>
</html>
```

- **Same syntax and structure as patXMLRenderer**

Comparison

	patXMLRenderer	PEAR	phpTagLib
Templates	yes, patTemplate	-	yes, plain HTML
„Intelligent“ Templates	✔	-	Only with PHP code
Dynamic Content (Callbacks)	Yes, only object	Yes, objects and functions	-
Namespaces	✔	✔	-
Recursion	Yes, infinite	Yes, infinite	-
External Entities	✔	-	-
<?PHP ?>	✔	-	✔
Parameters	✔	-	-
Administration	✔	-	-
Ext. Repository	✔	All in one package	-

XSLT vs PHP Transformers

Pro XSLT:

- W3C Standard
- XPath allows restructuring of the document, e.g. creation of a table of contents
- Portable (not limited to servers with PHP)

Pro PHP Transformers:

- Easier to learn (designers are familiar with templates)
- Easy to extend
- Complete control during the whole transformation
- Result may be transformed again (recursive transformation)

Drawbacks of PHP Transformers

Biggest disadvantages of PHP Transformers:

- Only the current tag is accessible
- As a SAX parser is used, no restructuring or reordering is possible.
- Table of contents can not be generated
- Cross references are not possible
- Sorting is not possible

Possible solutions:

- Two pass transformations (like LaTeX)
- Combination of XSLT and PHP Transformers

Combining forces

Combining XSLT and patXMLRenderer is possible by using the XSLT extension of patXMLRenderer:

- Uses XSLT to transform
  - The whole document
  - Parts of the document
- May be combined with all of patXMLRenderers features: inclusion of any dynamic content, control structures, caching, etc.
- Result of XSLT transformation may be transformed using templates.

patXMLRenderer and XSLT example

```
<xslt:transform returntype="xml">
   <xslt:stylesheet type="file">
         extensions/xslt/xsl/example.xsl
   </xslt:stylesheet>
   <xslt:param name="displayToc">yes</xslt:param>
      <sections>
         <section title="Dummy Section 1">
            <para>This is just a dummy....</para>
         </section>
         <section title="Dummy Section 2">
            <para>Another Section...</para>
         </section>
      </sections>
</xslt:transform>
```

XSLT Stylesheet

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">
  <xsl:output method="xml" />
  <xsl:param name="displayToc" />
  <xsl:template match="/">

    <xsl:if test="$displayToc = 'yes'">
      <section title="Table of contents">
        <ol>
          <!-- Create TOC -->
          <xsl:for-each select="//section">
            <oli><xsl:value-of select="@title" /></oli>
          </xsl:for-each>
        </ol>
      </section>
    </xsl:if>

    <!-- Display contents -->
    <xsl:for-each select="*|text()">
      <xsl:apply-templates select="*|text()" />
    </xsl:for-each>
  </xsl:template>

  <!-- xsl template to insert original content has been left out -->
</xsl:stylesheet>
```

The End

Thank you!

More information:

- http://www.php-tools.net
- schst@php-tools.net

Thanks to:

Sebastian Mordziol, Gerd Schaufelberger,
Metrix Internet Design GmbH