Introduction to XSLT with PHP4

PHPCon 2002

10/25/2002, Millbrae California USA

Stephan Schmidt

Table of Contents

Stephan Schmidt

- Web application developer since 1999
- Working for Metrix Internet Design GmbH in Germany
- Maintainer of www.php-tools.net
- Author of patTemplate, patXMLRenderer, patUser and others
- Focussing on the combination of XML and PHP and the separation of content and layout

Separation of content and layout

Why?

- Modify layout without accessing content
- Modify content without accessing layout
- Different views for different devices (Browser, mobile phone,...)
- Different views for different user types (visitor, client, administrator)

How?

- Different approaches...

Structuring content

- Classic websites and the use of HTML:
  - » no structure at all
  - » not readable by machines
- Classic software-development uses RDBMs:
  - » hardly readable by humans
    - – Use of relations to implement structure
    - – Use of several table (normalization)
    - – Use of cryptic IDs

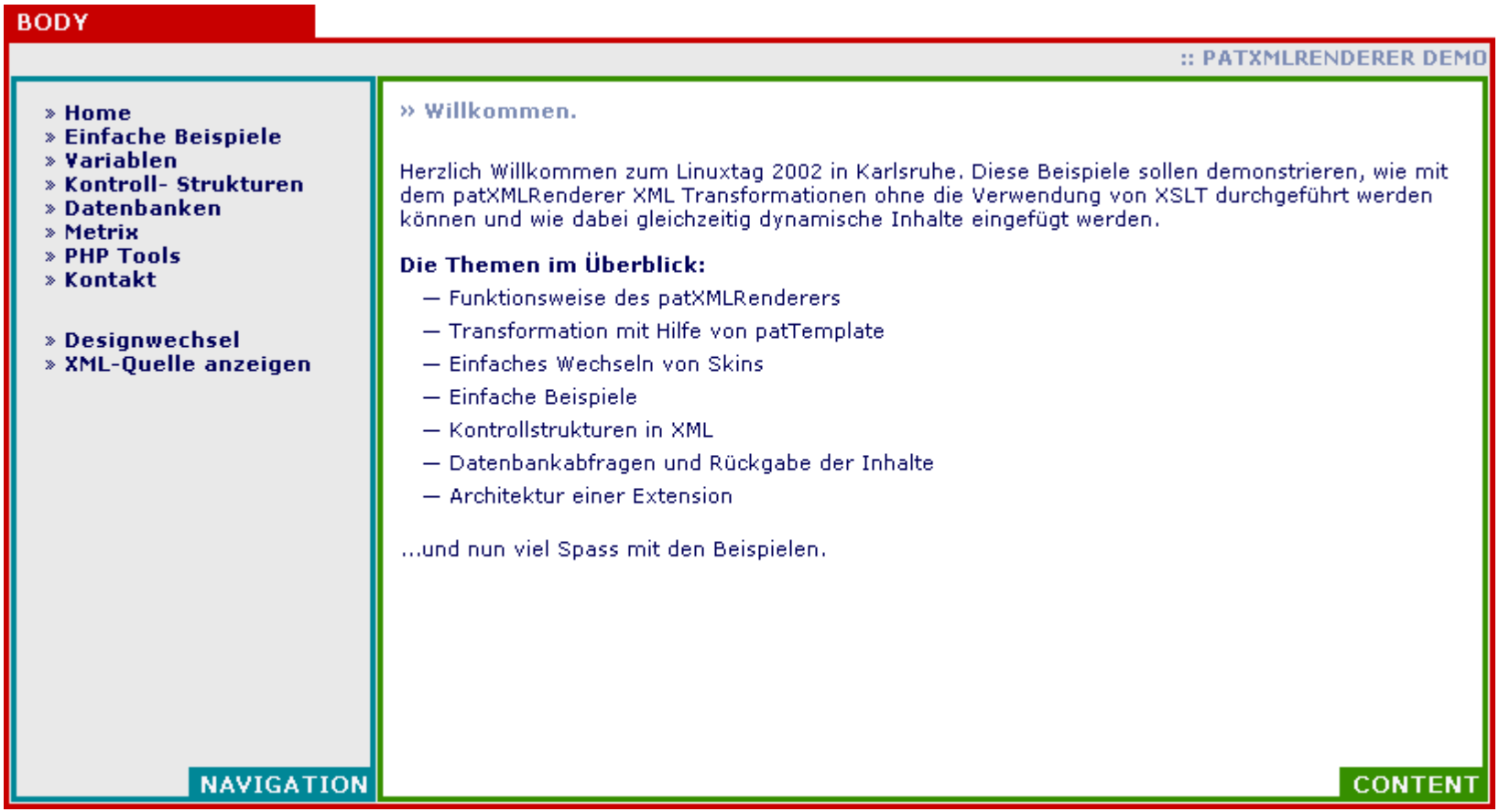
Using XML for structured data

- Readable by humans:
  - » complete data in one file
  - » self-explaining tag names
  - » self-explaining attribute names
  - » structured by indentation
- Readable by machines:
  - » Well-formed document
  - » only ASCII data
  - » Validation with DTD or schema
- Separation of content and layout

Building a content structure 1

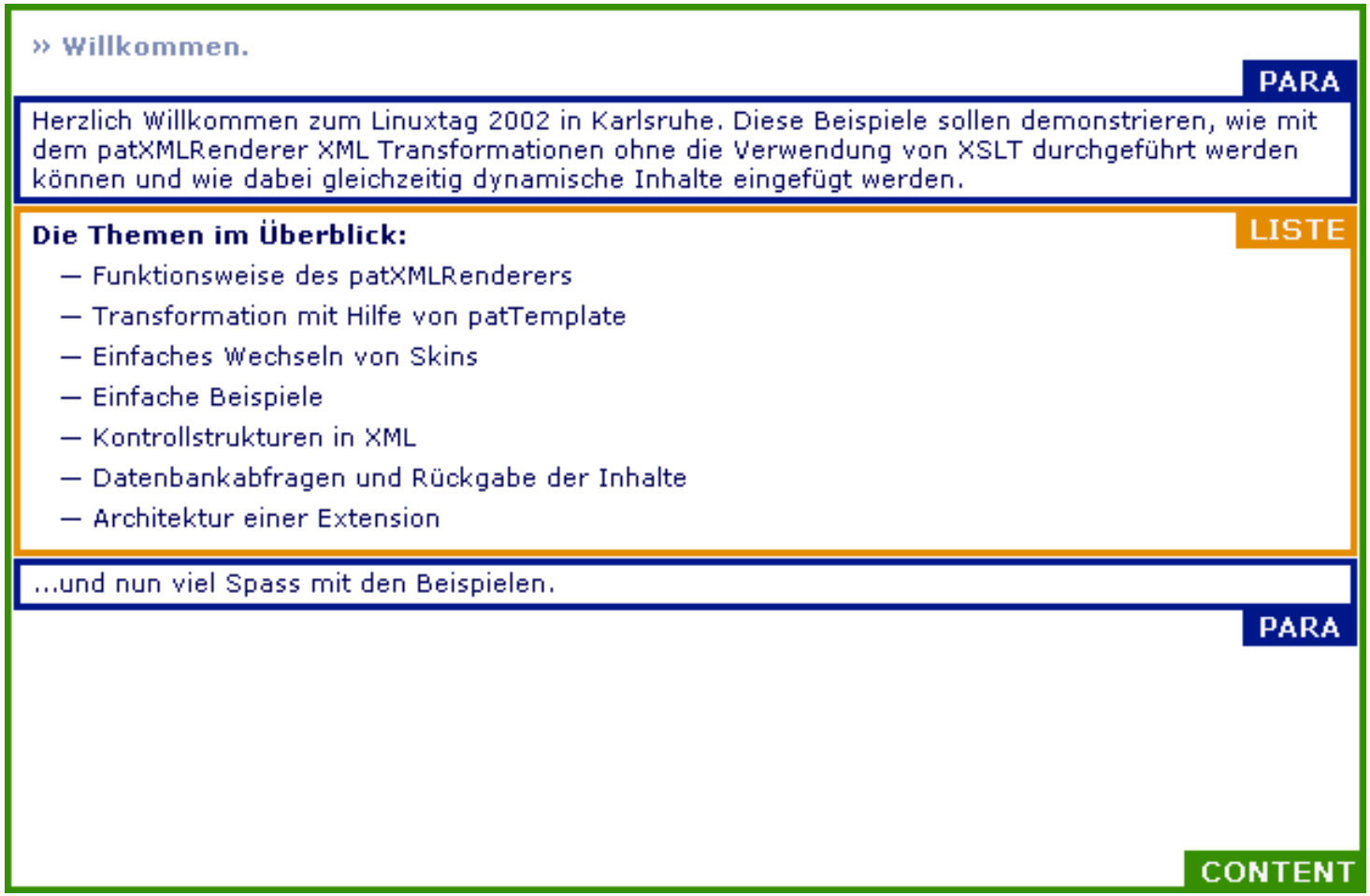
- Split content into logical elements
  - » navigation
  - » Sections and parargraphs
  - » Lists and list elements
- Tags should describe their content, e.g. use <important> for important parts in you page
- No layout-elements like
, use ... instead
- Define and write down page structure
  - » XML Schema
  - » DTD

Building a content structure 2



Building a content structure 3

Building a content structure 4



Basic XML Rules

- Each document needs a root element
- Each tag has to be closed, i.e. … instead of just
- Use trailing slash to indicate an empty element (
 instead of
)
- Elements may not overlap (……… instead of …… …)
- Attribute values have to be enclosed in double quotes
- Entities for <, >, &, “ and ‘ have to be used

XML Example

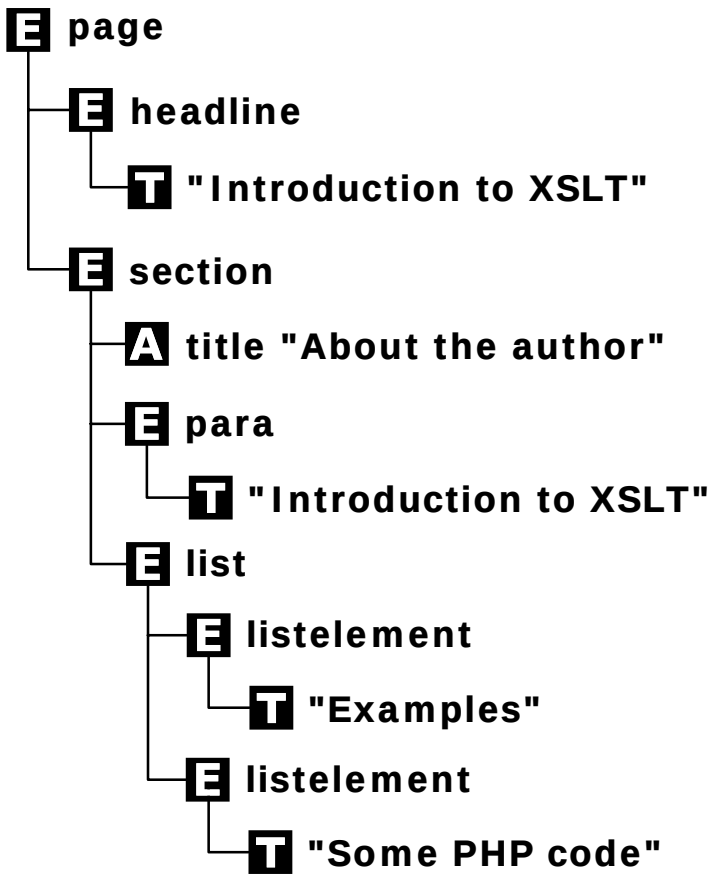
```
<page>
    <headline>Introduction to XSLT</headline>
    <section title="Abstract">
        <para>The presentation includes</para>
        <list>
            <listelement>Examples</listelement>
            <listelement>Some PHP code</listelement>
        </list>
    </section>
    <section title="About the author">
        <para>Web developer from Germany</para>
    </section>
</page>
```

Tree view of a document

Each piece of information in an XML document is referred to as a node:

- Element Nodes (tags)
- Attribute Nodes (attributes of a tag)
- Text Nodes (text and Cdata sections)
- Comment Nodes (<!-- ... -->)
- Processing Instruction Nodes (<?PHP ... ?>)
- Namespace Nodes (xmlns:myns="...")

Tree view of a document (example)



XML Transformations

Each tag has to be transformed into an HTML representation:

```
<headline>Introduction to XSLT</headline>
```

...is transformed to:

```
<font size="+1">
    <br><b>Introduction to XSLT</b><br><br>
</font>
```

Introduction to XSLT

- Stylesheet must be an XML document
  (well-formed and valid)
- Based on pattern-matching
  („When you see something that looks like this, do something else“)
- Free of sideeffects
  (One rule may not influence or modify others)
- Use of iteration and recursion

„Hello World“ Example (XML)

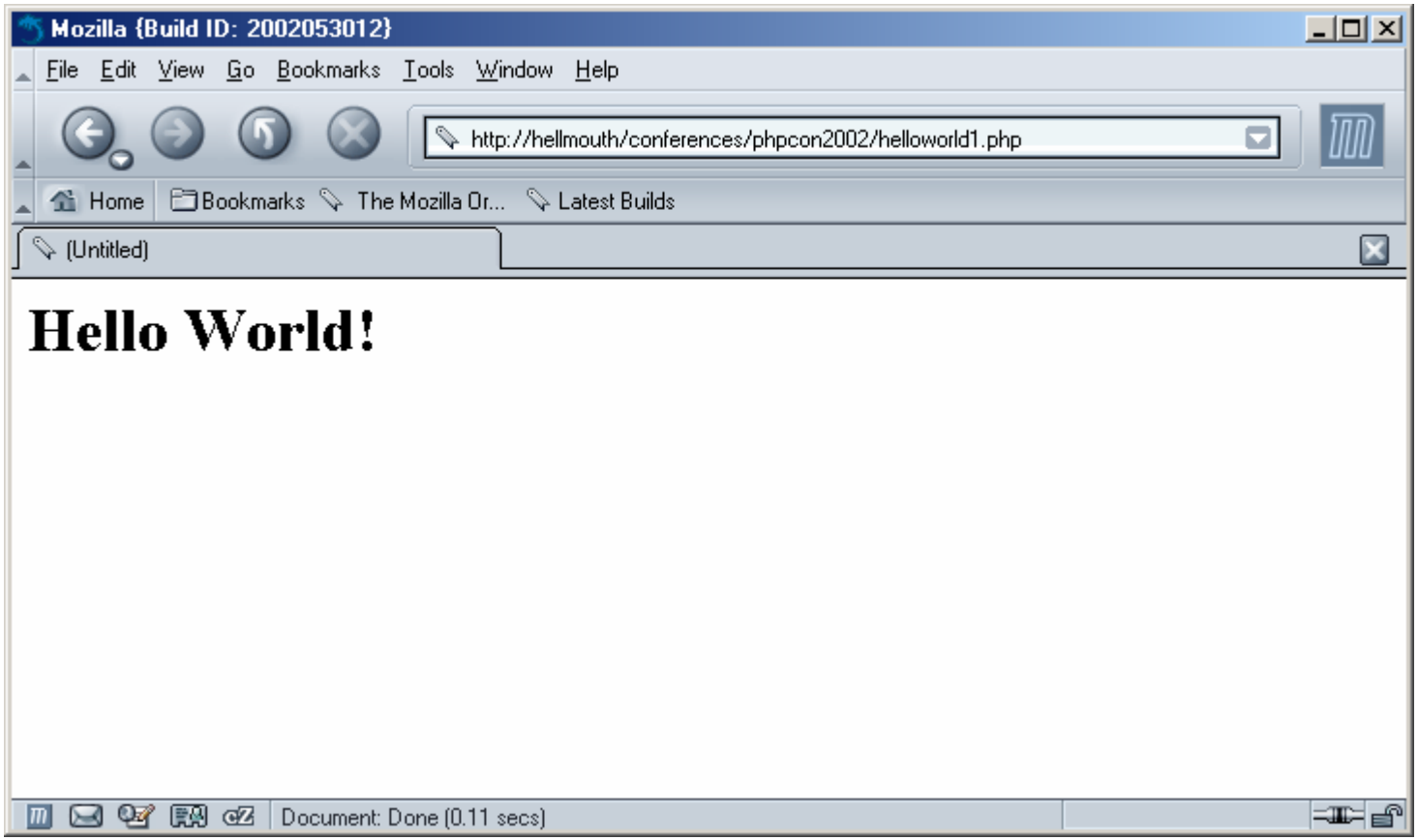
Sample Document:

```
<?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)



Closer look at the XSL template

- <xsl:stylesheet> root element and its attributes are mandatory
- <xsl:output/> defines output format (sets some properties in the XSLT processor)
- Two <xsl:template> tags define how XML elements are transformed
- <xsl:apply-templates> invokes other templates

Transformation Workflow

- Read stylesheet, so that each part may be easily accessed
- Read the XML document and build tree view of the document
- Transform the document
  - » Check for nodes to process („the context“)
  - » Get the next node from the context
  - » Get transformation rule and transform the node

As a rule may invoke other rules, the document is processed recursively.

Using PHP‘s XSLT functions

Run configure with the --enable-xslt --with-xslt-sablot options.

1. Create XSLT processor with xslt_create()
2. Apply stylesheet to XML document with xslt_process() (Strings or Files may be used)
3. Free memory with xslt_free()

Take a look at the example.

Using PHP's XSLT functions

```
$xmlFile   =   "xml/helloworld.xml";
$xslFile   =   "xsl/helloworld.xsl";

if( !$xp = xslt_create() )
    die( "XSLT processor could not be created." );

if( $result = @xslt_process( $xp, $xmlFile, $xslFile ) )
{
    echo  $result;
}
else
{
  echo "An error occured: " .xslt_error( $xp )." (error code ". xslt_errno( $xp ).")";
}

xslt_free( $xp );
```

xslt_process()

Accepts six parameters:

1. A handle, representing the XSLT processor
2. The name of the file or buffer containing the XML data
3. The name of the file or buffer containing the XSL data
4. The name of the file to save the result to (optional)
5. An associative array of argument buffers
   (for transforming strings on-the-fly)
6. An associative array of XSLT parameters
   (may be used as $paramName in stylesheet)

Other XSLT functions

- xslt_error() and xslt_errno() to retrieve the last error
- xslt_set_error_handler() to define a custom error handler (not stable in PHP 4.3)
- xslt_set_log() to log processor messages to a logfile

Too complicated?

» use patXSLT...

patXSLT

- Soon available for download at http://www.php-tools.net
- Simple interface for all xslt_* functions:

```
require_once( "include/patXSLT.php" );

$proc = new patXSLT();

$proc->setXMLFile( "xml/helloworld.xml" );
$proc->setXSLFile( "xsl/helloworld.xsl" );
$proc->enableLogging( "logs/xslt.log" );
$proc->addParam( "pagetitle ", $pageTitle );

$proc->transform();
```

XPath expressions

- Describe parts of an XML document (elements, attributes, text, ...)
- May be used in select and match attributes of various elements
- Melting pot of programming languages ($x*4) and Unix-like path expressions (like /page/section/para)
- Works with the parsed version of XML documents, which means no access to entities and no possibility to decide whether a text node was text or Cdata
- Example XPath expressions: „/“, „greetings/greeting“ and „greeting“

Location Paths

- One of the most common uses of XPath
- Always in respect of the context („the current directory“)
- Relative and absolute expressions (always evaluated from the root node)
- Simple location paths:
  - » <xsl:template match=“/“> (root node)
  - » <xsl:value-of select=“.“/> (context node)
  - » <xsl:value-of select=“..“/> (parent node)
  - » <xsl:apply-templates select=“greeting“/>
  - » <xsl:apply-templates select=“/page/section“/>

Selecting Things Besides Elements

- Selecting attributes

  /page/section/@title

- Selecting text nodes

  /page/title/text()

- Selecting comments and processing instructions

  /page/comment() and /page/processing-instruction()

Example: The World Answers (1)

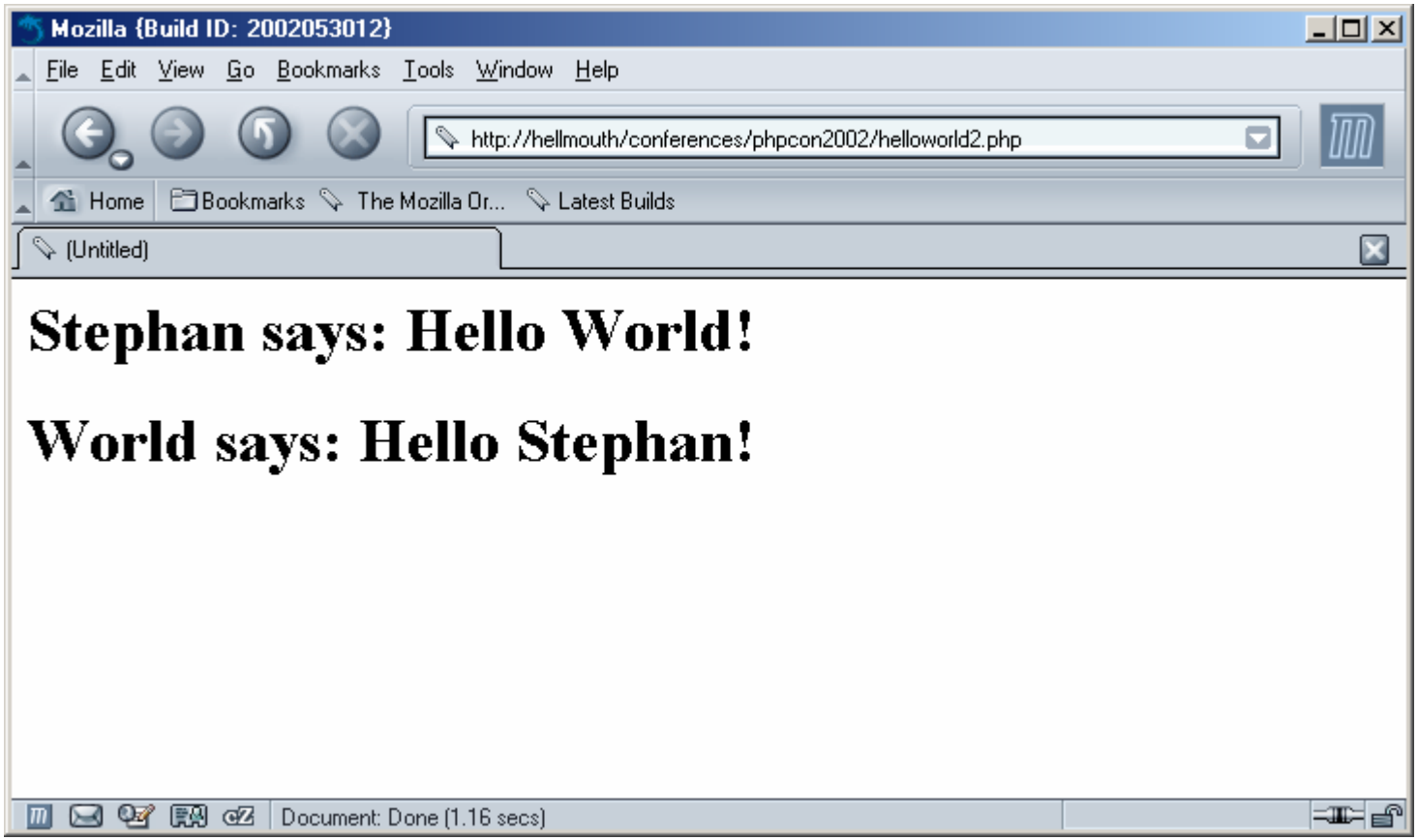
Sample document:

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

Example: The World Answers (2)

```
<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="@author"/>
        <xsl:text> says: </xsl:text>
        <xsl:value-of select="."/>
      </h1>
  </xsl:template>
</xsl:stylesheet>
```

Example: The World Answers (3)



Axes in a document

By default child axis is used, but there are several axes:

- Child axis (default)
- Parent axis (..)
- Self axis (.)
- Attribute axis (@)
- Ancestor axis (ancestor::)
- Descendant axis (descendant::)
- Preceding-sibling axis (preceding-sibling::)
- Following-sibling axis (following-sibling::)

Control Elements

- <xsl:if test="..."> implements an if statement ☺
  The expression in test attribute is converted to a boolean value. Examples:

```
<xsl:if test="count(section)&gt;3">
<xsl:if test="@title">
```

- <xsl:choose>, <xsl:when> and <xsl:otherwise> implement a switch/case statement. Also used to implement an if-then-else statement.

Example needed?

Example: A conversation (1)

Sample document:

```
<?xml version="1.0"?>
<greetings>
    <greeting author="Stephan">Hello World!</greeting>
    <greeting author="World">Hello Stephan!</greeting>
    <greeting>Hello both of you!</greeting>
</greetings>
```

Example: A conversation (2)

```
<xsl:template match="greeting">
   <h1>
      <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>
    </h1>
</xsl:template>
```

Example: A conversation (3)

„Procedural XSLT“

- <xsl:template> may have an „name“ attribute
- <xsl:call-template name=„...“> is used to call a template by name
- <xsl:with-param name=„...“ select=„...“> is used to pass parameters to the called template

Additional XSLT Features

- Wildcards (*, @*, page//para)
- Predicates (line[3], entry[position() mod 2 = 0])
- <xsl:for-each> to process several nodes
- <xsl:template mode="..."> and
  <xsl:apply-templates mode="...">
- Global parameters, that may be passed from PHP to the processor
- Variables (may not be changed!)
- Sorting and grouping (using some tricks)

Drawbacks of XSLT

- Needs to be installed (requires Sablotron)
- Designer needs to learn XSLT tags
- The XSLT-processor is a black-box:
  - » No possibilty to pause or influence the process once it has started
  - » It is not possible to include dynamic content into an XML file while processing.

Transforming XML with PHP

- HTML Template for each XML element
- Attribute values are used as template variables
- The content of an element is a special template variable: {CONTENT}
- The XML document is parsed recursively using SAX-based parser

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>
```

Including Dynamic Content

By using callbacks, each PHP function may be used at any time.

Simple Example: <time:current/> should insert current time.

Solution:
switch/case statement in element handler that executes PHP code instead of replacing the tag with its template.

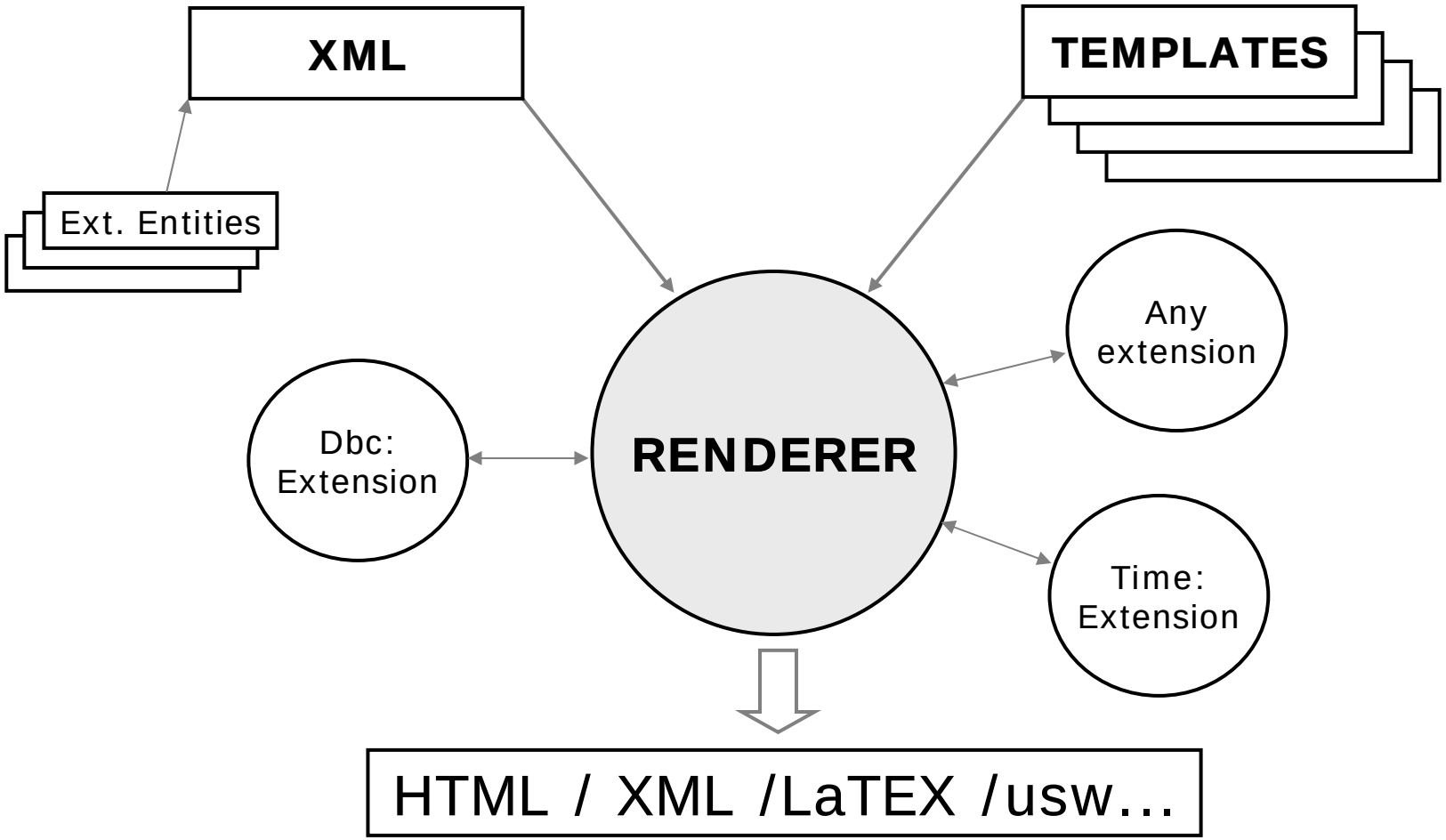
- » code is hard to maintain

Improvement:
extensible parser, that invokes callbacks for specific namespaces.

Randy - patXMLRenderer

- Open source (LGPL License)
- Uses the template class patTemplate
- Supports external entities („XML includes“)
- Unlimited amount of extensions. Tags will be passed to an extension according to namespaces
- The renderer only transforms static content and content returned by extensions
- Unlimited amount of designs
- Other features: caching, logging, etc...

Architecture



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>
```

Existing Extensions

- Repository on http://www.php-tools.net
- Documentation is generated automatically
- Examples:
  - » **<time:...>** date and time functions
  - » **<dbc:...>** database interface
  - » **<var:...>** access to variables
  - » **<control:...>** control structures
  - » **<randy:...>** XML-API for patXMLRenderer
  - » **<file:...>** file operations
  - » and many more...

Similar Applications

At least two more applications:

- PEAR::XML_Transformer
  http://pear.php.net
  Only overloads tags with functions and methods.

- phpTagLib
  http://chocobot.d2g.com
  Only transforms XML to HTML.

Comparison

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

XSLT vs PHP Transformers

Pro XSLT:

- Much faster than PHP transformers
- W3C Standard (works with any XSLT Processor)

Pro PHP Transfomers:

- Easily extendable
- Easier to learn
- Script has complete control during the transformation process

The End

Thank you!

More information:

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

Thanks to:

Sebastian Mordziol, Gerd Schaufelberger, Stephan Eisler, Metrix Internet Design GmbH