XML and Web services with PHP5 and PEAR

Stephan Schmidt & Tobias Schlitt

Welcome

- Welcome to our power workshop
- We hope you'll enjoy the session and that you will see some interessting stuff
- When questions occur, please do not hesitate to ask directly

Agenda

- Introduction
- Introduction to XML
- XML in PHP 5.0/5.1 & PECL
- PEAR
- XML in PEAR
- Introduction to Webservices
- Webservices in PHP 5.0/5.1 & PECL
- Webservices in PEAR
- Q&A session

Agenda - Introduction

- Introduction
- Introduction to XML
- XML in PHP 5.0/5.1 & PECL
- PEAR
- XML in PEAR
- Introduction to Webservices
- Webservices in PHP 5.0/5.1 & PECL
- Webservices in PEAR
- Q&A session

Who is who?

- Stephan Schmidt
- schst@php.net
- http://pear.php.net/user/schst
- Working for 1&1 Internet AG
- Founding member of PHP Application Tools (pat)
- Active PEAR developer
- Writing for several PHP magazines
- Written the XML & Web Services chapters for O'Reilly's new PHP5 Cookbook (in German)

Who is who?

- Tobias Schlitt
- toby@php.net
- http://pear.php.net/user/toby
- Student of Computer Science (University of Dortmund)
  - Freelancing IT Consultant / Trainer
  - Former Software Architect at Deutsche Bank
- Member of the PEAR project since 2002
  - Package maintainer
  - Member of the Core QA Team
  - Working on the website
- Zend certified engineer

Who is who?

And who are you?

- Your name?
- Your location?
- Your field of work?
- Your experiences with XML, Webservices, PHP5 and PEAR?

Buzzword Parade

XHTML

FOAF

XSL

XML-RPC

UDDI

- XML
  Extensible Markup Language

RDF

- Webservice
  A program / protocoll to allow communication on maschine level through huge networks.

Technochrati

RSS

Famous bookstore starting with A

Famous web auction starting with E

Famous websearch starting with G

Trackback

DTD

Schema

Atom

XSLT

XPath

Rest

Sax

Agenda

- Introduction
- Introduction to XML
- XML in PHP 5.0/5.1 & PECL
- PEAR
- XML in PEAR
- Introduction to Webservices
- Webservices in PHP 5.0/5.1 & PECL
- Webservices in PEAR
- Q&A session

Agenda – Introduction XML

- Introduction
- Introduction to XML
  - XML in general
  - DTD
  - Schema
  - Relax NG
  - XPath
  - XSL(T)
- ...

XML example

```
<?xml version="1.0"?>
<package packagerversion="1.4.0a4">
 <name>Services_Trackback</name>
 <channel>pear.php.net</channel>
</package>
```

XML basic rules

- Root element (encapsulates all tags)
  - <package></package>
- Case sensitive
  - <name /> != <NAME /> != <nAmE /> != ...
- Well formed
  - Close all opened tags
  - Escape special chars
- Namespaces
  - <namespace:tag />
  - <tag namespace:att=”...” />

XML appliance

- A general data describtion format
- Typical appliances:
  - configuration
  - data exchange
  - content structuring
  - layout abstraction

XML related technologies

- DTD
  - W3C standard (http://www.w3.org/TR/REC-xml/)
  - Used for validation purposes
  - Outdated
  - Example:

```
<!ELEMENT package (name,extends?,summary)>
<!ATTLIST
    package type (source|binary|empty) "empty"
    version CDATA #REQUIRED>
<!ELEMENT name (#PCDATA)>
...
```

XML related technologies

- XML Schema
  - W3C standard (http://www.w3.org/XML/Schema)
  - Better validation features than DTD
  - More flexible (more datatypes (10 (DTD), 44+ (Schema))
  - Up-2-date
  - Easier (written in the same style as instance documents)

XML related technologies

- XML Schema
  - Example:

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                  targetNamespace="http://www.books.org"
                  xmlns="http://www.books.org"
                  elementFormDefault="qualified">
        <xsd:element name="Title" type="xsd:string"/>
        <xsd:element name="Author" type="xsd:string"/>
        <xsd:element name="Date" type="xsd:string"/>
        <xsd:element name="ISBN" type="xsd:string"/>
        <xsd:element name="Publisher" type="xsd:string"/>
</xsd:schema>
```

XML related technologies

- Relax NG
  - ISO standard (http://www.relaxng.org/) (OASIS - Organization for the Advancement of Structured Information)
  - Up-2-date
  - Even easier than Schema
  - 2 describtion formats (XML and non-XML)

XML related technologies

- Relax NG
  - Example (XML syntax):

```
<element name="addressBook"
   xmlns="http://relaxng.org/ns/structure/1.0">
  <zeroOrMore>
    <element name="card">
      <element name="name"> <text/> </element>
      <optional>
         <element name="note"> <text/> </element>
      </optional>
    </element>
  </zeroOrMore>
</element>
```

XML related technologies

- Relax NG
  - Example (compact syntax):

```
element addressBook {
  element card {
    element name { text },
    element note { text }?
  }*
}
```

XML related technologies

- Xpath
  - W3C standard (http://www.w3.org/TR/xpath)
  - Find data nodes in documents
  - Mirrors XML tree structure
  - Filesystem-like path syntax
  - Up-2-date
  - Part of the XSL family

XML related technologies

- Xpath
  - Example: */bookstore/book/title*

```
<bookstore>
  <book>
    <title>PEAR is sexy</title>
    <author>Tobias Schlitt</author>
  </book>
  <book>
    <title>PECL is cool</title>
    <author>Stephan Schmidt</author>
  </book>
</bookstore>
```

XML related technologies

- Xpath
  - Example: *//author*

```
<bookstore>
  <book>
    <title>PEAR is sexy</title>
    <author>Tobias Schlitt</author>
  </book>
  <book>
    <title>PECL is cool</title>
    <author>Tobias Schlitt</author>
  </book>
</bookstore>
```

XML related technologies

- XSL(T)
  - W3C standard (http://www.w3.org/Style/XSL/)
  - Define style information for XML data
  - Saying XSL you mostly mean XSLT
  - Transform XML docs into XML docs
  - Using Xpath for navigation
  - “XSLT sucks”

XML related technologies

- XSL(T)
  - Example: Input XML

```
<catalog>
  <cd>
    <title>Black Album</title>
    <artist>Metallica</artist>
  </cd>
...
</catalog>
```

XML related technologies

- XSL(T)
  - Example: Stylesheet

```
<xsl:template match="/">
...
  <ul>
  <xsl:for-each select="catalog/cd">
    <li><xsl:value-of select="title"/>,
    <xsl:value-of select="artist"/></li>
  </xsl:for-each>
  </ul>
...
</xsl:template>
```

XML related technologies

- XSL(T)
  - Example: Output XML

```
...
    <ul>
      <li>Black Album,
      Metallica</li>
    </ul>
...
```

Usefull links

- http://www.w3.org/XML/
- http://www.w3.org/Style/XSL/
- http://www.w3.org/TR/REC-xml/
- http://www.w3.org/XML/Schema
- http://www.relaxng.org/
- http://www.w3.org/TR/xpath
- http://www.w3.org/TR/xslt
- http://www.w3schools.com/xsl/
- http://pear.php.net/package/XML_Transformer
-

Agenda

- Introduction
- Introduction to XML
- XML in PHP 5.0/5.1 & PECL
- PEAR
- XML in PEAR
- Introduction to Webservices
- Webservices in PHP 5.0/5.1 & PECL
- Webservices in PEAR
- Q&A session

Agenda: XML in PHP5

- SAX
- DOM
- SimpleXML
- XPath
- XSLT

XML in PHP5

- completely revamped
- based on libxml2 and libxslt
- easy to compile
- mostly standards compliant

XML Example

```xml
<teams>
    <!-- Justice League of America -->
    <team id="JLA">
        <name>Justice League of America</name>
        <members>
            <member alias="Superman" gender="male">
                <name secret="yes">Clark Kent</name>
                <powers>
                    <power>Super-Strength</power>
                    <power>Heat Vision</power>
                </powers>
            </member>
        </members>
    </team>
</teams>
```

SAX

- Simple API for XML
- event-based processing of XML documents
- enabled by default
  (use `--disable-xml` to disable it)
- available since PHP 3.0.6

```
$p = xml_parser_create();
```

SAX: Introduction

- cursor moves through the document
- tokenizes the document
- triggers callbacks for
  - tags (opening and closing)
  - character data
  - entities
- No way to move the cursor backwards

SAX: Example

```
$parser = xml_parser_create();
xml_set_element_handler($parser, 'startElement',
  'endElement');

xml_set_character_data_handler($parser, 'cData');

$fp = fopen('example.xml', 'r');
while ($data = fread($fp, 1024)) {
    $result = xml_parse($parser, $data);
}
fclose($fp);
```

SAX: Callbacks

```
function startElement($p, $name, $atts) {
    print "opening $name\n";
}
function endElement($p, $name) {
  print "closing $name\n";
}
function cData($p, $data) {
    $data = trim($data);
  if (!empty($data)) {
      print "data: $data\n";
    }
}
```

SAX: Result

```
opening TEAMS
opening TEAM
opening NAME
data: Justice League of America
closing NAME
opening MEMBERS
opening MEMBER
opening NAME
data: Clark Kent
closing NAME
opening POWERS
opening POWER
data: Super-Strength
closing POWER
…
```

SAX: Object-Oriented

```
class ParserObj {
   var $current = null;
   var $heroes  = array();
   var $data    = null;
   function startElement($p, $name, $atts) {
    $this->data = null;
    switch ($name) {
       case 'MEMBER':
          $this->current = $atts['ALIAS'];
          $this->heroes[$this->current] = array();
          $this->heroes[$this->current]['gender'] =
                                                 $atts['GENDER'];
          break;
```

SAX: Object-Oriented

```
    case 'POWERS':
      $this->heroes[$this->current]['powers'] =
                                               array();
      break;
   }
 }
 function endElement($p, $name) {
    if ($this->current === null) {
     return true;
   }
  switch ($name) {
    case 'NAME':
      $this->heroes[$this->current]['name']
                                        = $this->data;
      break;
```

SAX: Object-Oriented

```
    case 'MEMBER':
      $this->current = null;
      break;
   }
  }
  function cData($p, $data) {
    $data = trim($data);
    if (!empty($data)) {
      $this->data = $data;
    }
  }
}
```

SAX: Object-Oriented

```
$parserObj = new ParserObj();
$parser = xml_parser_create();
xml_set_element_handler($parser, 'startElement',
                                 'endElement');
xml_set_character_data_handler($parser, 'cData');
xml_set_object($parser, $parserObj);
$fp = fopen('example.xml', 'r');
while ($data = fread($fp, 1024)) {
  $result = xml_parse($parser, $data);
   if ($result === false) {
    die(sprintf("XML error: %s at line %d",
     xml_error_string(xml_get_error_code($xml_parser)),
     xml_get_current_line_number($xml_parser)));
   }
}
```

SAX: Result

```
Array (
    [Superman] => Array (
            [gender] => male
            [name] => Clark Kent
            [powers] => Array (
                    [0] => Super-Strength
                    [1] => Heat Vision
                )

        )
    ...
)
```

SAX: Advantages

- very low memory footprint
- available nearly everywhere
- easy-to-use, once you've understood the principle

SAX: Disadvantages

- slow, as it uses PHP callbacks
- not possible to modify a document
- not possible to create new documents
- not possible to influence the parsing process
- requires a lot of work
  (use PEAR::XML_Parser)

DOM

- Document Object Model
- Official W3C standard
- available in several languages
- PHP5 implements DOM level 2
- Builds a tree of the XML document in memory that consists of nodes

DOM: Nodes

- Nodes provide means for iteration
  - childNodes, firstChild
  - nextSibling, previousSibling
  - parentNode
- Different node types
  - XML elements
  - text data
  - comments
  - ...

DOM: Example (readonly)

```
$dom = DOMDocument::load('example.xml');
function process_node($node) {
  if ($node->hasChildNodes()) {
    foreach($node->childNodes as $n) {
      process_node($n);
    }
  }
  switch ($node->nodeType) {
    case XML_TEXT_NODE:
      print rtrim($node->nodeValue) . "\n";
}
     break;
    case XML_ELEMENT_NODE:
     $name = "Tag: " . $node->nodeName ."\n";
  }
}
process_node($dom->documentElement);
```

DOM: Result

```
Justice League of America
Tag name
Clark Kent
Tag name
Super-Strength
Tag power
Heat Vision
Tag power
Tag powers
Tag member
Arthur Curry
Tag name
Super-fast Swimmer
Tag power
Commands Sea Life
……
```

DOM: Example 2

```
$dom = DOMDocument::load('example.xml');

$powers = $dom->getElementsByTagname('power');
for ($i = 0; $i < $powers->length; $i++) {
  $power = $powers->item($i);
  print $power->textContent . "\n";
}
```

- Fetch a NodeList of all `<power/>` tags
- textContent is a PHP5 addition to DOM for `$power->firstChild->nodeValue`

DOM: Example 2

```
Super-Strength
Heat Vision
Super-fast Swimmer
Commands Sea Life
Super-Strength
Flight
Canary Cry
Flight
Warrior Skills
```

DOM: Modifying documents

Superman wants to join the JSA.

- Fetch the `<team id="JSA"/>` node and then the `<members/>` node
- Fetch the `<member alias="Superman"/>` node
- Make a copy of the Superman node
- add it as a child node to the members node

DOM: Modifying the tree

```
$dom = new DOMDocument;
$dom->preserveWhiteSpace = false;
$dom->formatOutput = true;
$dom->load('example.xml');

$root = $dom->documentElement;
```

- Ignore unneeded white space
- Use indentation when serializing the XML document

DOM: Modifying the tree

```
for ($i = 0; $i < $root->childNodes->length; $i++) {
 if ($root->childNodes->item($i)->nodeType !=
   XML_ELEMENT_NODE) {
    continue;
 }
 if ($root->childNodes->item($i)->nodeName != 'team') {
    continue;
 }
 if ($root->childNodes->item($i)->getAttribute('id')
                                          != 'JSA' ) {
    continue;
 }
 $jsa = $root->childNodes->item($i);
 $members = $jsa->childNodes->item(1);
}
```

DOM: Modifying the tree

```
$heroes = $dom->getElementsByTagname('member');
for ($i = 0; $i < $heroes->length; $i++) {
  if ($heroes->item($i)->getAttribute('alias') !==
   'Superman') {
    continue;
  }
  $superman = $heroes->item($i);
  break;
}
$supermanClone = $superman->cloneNode(true);

$members->appendChild($supermanClone);
print $dom->saveXML();
```

DOM: Result

```
<?xml version="1.0"?>
<teams>
  …
  <team id="JSA">
    <name>Justice Society of America</name>
    <members>
      …
      <member alias="Superman" gender="male">
        <name secret="yes">Clark Kent</name>
        <powers>
          <power>Super-Strength</power>
          <power>Heat Vision</power>
        </powers>
      </member>
      …
```

DOM: Creating documents

- DOMDocument object provides methods to create nodes
  - `createElement()`
  - `createTextNode()`
  - `createComment()`
- Build a tree of nodes using `appendChild()`
- save it to a string or file

DOM: Example (New Tree)

```
$dom = new DOMDocument('1.0', 'iso-8859-1');
$dom->formatOutput = true;

$teams = $dom->createElement('teams');
$dom->appendChild($teams);
$teams->appendChild($dom->createComment('Avengers'));

$avengers = $dom->createElement('team');
$avengers->setAttribute('id', 'Avengers');
$name = $avengers->appendChild(
                      $dom->createElement('name'));
$name->appendChild(
              $dom->createTextNode('The Avengers'));
```

DOM: Example (New Tree)

```
$members = $avengers->appendChild(
                      $dom->createElement('members'));
$cap = $members->appendChild(
                       $dom->createElement('member'));
$cap->setAttribute('alias', 'Captain America');
$cap->setAttribute('gender', 'male');
$nameTag = $cap->appendChild(
                         $dom->createElement('name'));
$nameTag->setAttribute('secret', 'no');
$nameTag->appendChild(
               $dom->createTextNode('Steven Rogers'));
$teams->appendChild($avengers);

print $dom->saveXML();
```

DOM: Example (New Tree)

```
<?xml version="1.0" encoding="iso-8859-1"?>
<teams>
<!--Avengers-->
  <team id="Avengers">
    <name>The Avengers</name>
    <members>
      <member alias="Captain America" gender="male">
        <name secret="no">Steven Rogers</name>
      </member>
    </members>
  </team>
</teams>
```

DOM: HTML documents

- DOM allows you to read non-wellformed HTML documents
- Use the same methods on these documents
  - Iterate through the tree
  - modify the tree
  - save it as HTML
- Even XPath is possible

DOM: HTML example

```
$dom = new DOMDocument();
$dom->loadHTMLFile('http://pear.php.net');

$links = $dom->getElementsByTagname('a');
foreach ($links as $link) {
  print $link->getAttribute('href') . "\n";
}
```

```
/account-request.php
/login.php?redirect=/index.php
/manual/
/packages.php
/support/
/bugs/
…
```

DOM: xInclude

- Allows you to split XML-documents in smaller parts
- PHP5 DOM supports the streams API
  - HTTP, FTP
  - GZIP
  - Custom streams

DOM: Example (xInclude)

```
<teams xmlns:xi="http://www.w3.org/2001/XInclude">
  <team id="JSA">
    <name>Justice Society of America</name>
      <members>
      <member alias="Power Girl" gender="female">
      …
      </member>
      <xi:include href="hawkgirl.xml">
        <xi:fallback>
          <error>Could not load Hawkgirl</error>
        </xi:fallback>
      </xi:include>
    </members>
  </team>
</teams>
```

DOM: Example (xInclude)

```
$dom = new DOMDocument();
$dom->preserveWhiteSpace = false;
$dom->formatOutput = true;

$dom->load('example2.xml');
$dom->xInclude();

print $dom->saveXML();
```

DOM: Example (xInclude)

```
<?xml version="1.0"?>
<teams xmlns:xi="http://www.w3.org/2001/XInclude">
  <team id="JSA">
    <name>Justice Society of America</name>
    <members>
      …
      <member alias="Hawkgirl" gender="female"
              xml:base="path/to/hawkgirl.xml">
        <name secret="yes">Kendra Saunders</name>
          <powers>
            <power>Flight</power>
          </powers>
         </member>
    </members>
  </team>
</teams>
```

DOM: Validating documents

- DOM supports
  - DTD
  - XML Schema
  - RelaxNG
- Validation errors are PHP notices

DOM: DTD Validation

```
$dom = new DOMDocument;
$dom->load('example.xml');
if (!$dom->validate('superheroes.dtd')) {
  print "The document is not valid.\n";
}
```

DOM: Schema Validation

```
$dom = new DOMDocument;
$dom->load('example.xml');
if (!$dom->schemaValidate('superheroes.xsd')) {
  print "The document is not valid.\n";
}
```

DOM: RelaxNG Validation

```
$dom = new DOMDocument;
$dom->load('example.xml');
if (!$dom->relaxNGValidate('superheroes.rng')) {
  print "The document is not valid.\n";
}
```

DOM: Advantages

- W3C standard, leads to "portable" code
- Extremely powerful
- Access any part of the document at any time
- modify existing documents
- create new documents

DOM: Disadvantages

- High Memory Footprint
- Complex tree structures
- Code using DOM is extremely verbose

SimpleXML

- Makes clever use of PHP5's overloading features
- access an XML-document as it were a tree of PHP objects
  `$teams->team->name;`
- Multiple occurrences of a tag result in an array
  `$teams->team[0]->name;`

SimpleXML

- Attributes are accessed using array syntax
  ```
  $teams->team[0]['id'];
  ```
- Can be traversed using foreach() or the SimpleXMLIterator in SPL
- Supports XPath

SimpleXML: Example

```
$teams = simplexml_load_file('example.xml');
foreach ($teams->team as $team) {
  print $team['id'] . " - ";
  print $team->name . "\n";
  foreach ($team->members->member as $member) {
    print ' - ' . $member['alias'];
        print ' (' . $member->name . ")\n";
  }
}
```

SimpleXML: Example

```
JLA - Justice League of America
 - Superman (Clark Kent)
 - Aquaman (Arthur Curry)
JSA - Justice Society of America
 - Power Girl (Karen Star)
 - Black Canary (Dinah Laurel Lance)
 - Hawkman (Carter Hall)
```

SimpleXML: Example 2

```
$teams = simplexml_load_file('example.xml');

// Hide Superman's identity
$teams->team[0]->members->member[0]->name = 'Unknown';

print $teams->asXML();
```

SimpleXML: Example 2

```
<teams>
    <!-- Justice League of America -->
    <team id="JLA">
        <name>Justice League of America</name>
        <members>
            <member alias="Superman" gender="male">
                <name secret="yes">Unknown</name>
                <powers>
                    <power>Super-Strength</power>
                    <power>Heat Vision</power>
                </powers>
            </member>
            …
        </members>
     </team>
</teams>
```

SimpleXML: Example 3

```
$teams = simplexml_load_file('example.xml');
$teams->team[0]->asXML('jla.xml');
```

```
<team id="JLA">
    <name>Justice League of America</name>
    <members>
        <member alias="Superman" gender="male">
            <name secret="yes">Clark Kent</name>
            <powers>
                <power>Super-Strength</power>
                <power>Heat Vision</power>
            </powers>
        </member>
      ……
</team>
```

SimpleXML: Advantages

- Easy-to-use
- Interoperability with DOM
- provides XPath support
- Easy to extract parts of a document

SimpleXML: Disadvantages

- XML is not simple as the name implies
- Working with mixed content documents can get really messy (use DOM instead)
- Namespace support is buggy

XPath

- The SQL for XML
- Address portions of an XML-document using a non-XML language
- Available with DOM and SimpleXML in PHP5

XPath: Introduction

URI-like syntax

- `/teams/team` addresses all `<team/>` tags inside `<teams></teams>`
- `//name` addresses all `<name/>` tags regardless of their position
- `//member[@gender="female"]` addresses all female heroes

XPath: DOM

```
$dom   = DOMDocument::load('example.xml');
$xpath = new DOMXPath($dom);

$query  = '/teams/team';
$teams = $xpath->query($query);
foreach ($teams as $team) {
  print $dom->saveXML($team);
}
```

XPath: DOM

```
$dom   = DOMDocument::load('example.xml');
$xpath = new DOMXPath($dom);

$query  = '//member[@gender="female"]/name/text()';
$names = $xpath->query($query);

print "The female heroines are:\n";
foreach ($names as $name) {
  print $name->nodeValue . "\n";
}
```

```
The female heroines are:
Karen Star
Dinah Laurel Lance
```

XPath: Context Node

```
$dom   = DOMDocument::load('example.xml');
$xpath = new DOMXPath($dom);

$query  = '/teams/team[@id="JSA"]';
$teams = $xpath->query($query);
$jsa   = $teams->item(0);

$query2 = 'members/member[@gender="male"]/name/text()';
$names  = $xpath->query($query2, $jsa);

print "The male members of the JSA are:\n";
foreach ($names as $name) {
  print $name->nodeValue . "\n";
}
```

```
The male members of the JSA are:
Carter Hall
```

XPath: SimpleXML

```
$teams = simplexml_load_file('example.xml');

$query  = '//member[@gender="male"]';
$heroes = $teams->xpath($query);

print "The male heroes are:\n";
foreach ($heroes as $hero) {
  printf("%s (%s)\n", $hero->name, $hero['alias']);
}
```

```
The male heroes are:
Clark Kent (Superman)
Arthur Curry (Aquaman)
Carter Hall (Hawkman)
```

XSLT

- eXtensible XML Stylesheet Transformations
- W3C Standard
- ext/xsl, compiled using --with-xsl=[DIR]

XSLT: Introduction

- Transform an XML document to another XML (or HTML) format
- Functional programming language
- provides if, switch/case, functions, loops
- XSLT is XML as well
- XSL-FO transforms XML to PDF
- Extremely verbose

XSLT: Example

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="ISO-8859-1"/>
<xsl:template match="team">
  <h1><xsl:value-of select="@id"/></h1>
  <ul>
  <xsl:for-each select="members/member">
    <li>
      <xsl:value-of select="@alias"/>
      (<xsl:value-of select="name"/>)
    </li>
  </xsl:for-each>
  </ul>
</xsl:template>
</xsl:stylesheet>
```

XSLT: Example (toXML)

```
$xsl = DomDocument::load("stylesheet.xsl");
$xml = DomDocument::load("example.xml");

$proc = new XsltProcessor();
$proc->importStylesheet($xsl);

print $proc->transformToXML($xml);
```

XSLT: Example Result

```
<h1>JLA</h1>
<ul>
  <li>Superman (Clark Kent)</li>
  <li>Aquaman (Arthur Curry)</li>
</ul>
<h1>JSA</h1>
<ul>
  <li>Power Girl (Karen Star)</li>
  <li>Black Canary (Dinah Laurel Lance)</li>
  <li>Hawkman (Carter Hall)</li>
</ul>
```

XSLT: Example (toDoc)

```
$xsl = DomDocument::load("stylesheet.xsl");
$xml = DomDocument::load("example.xml");

$proc = new XsltProcessor();
$proc->importStylesheet($xsl);

$newDom $proc->transformToDoc($xml);
print $newDom->saveXML();
```

- Transform DOM to new DOM-tree
- Allows multiple transformations
- modify the document after the transformation

XSLT: Parameters

```
<?xml version="1.0" encoding="iso-8859-1" ?>
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
<xsl:output method="html" encoding="ISO-8859-1"/>
<xsl:template match="team">
  <h1><xsl:value-of select="@id"/></h1>
  <ul>
  <xsl:for-each select="members/member[@gender=$gender]">
    <li>
      <xsl:value-of select="@alias"/>
     (<xsl:value-of select="name"/>)
    </li>
  </xsl:for-each>
  </ul>
</xsl:template>
</xsl:stylesheet>
```

XSLT: Parameters

```
$xsl = DomDocument::load("stylesheet2.xsl");
$xml = DomDocument::load("example.xml");

$proc = new XsltProcessor();

$proc->importStylesheet($xsl);
$proc->setParameter('', 'gender', 'male');

print $proc->transformToXML($xml);
```

Pass parameters from PHP to the stylesheet to influence the transformation.

XSLT: php:functionString()

```
<xsl:stylesheet version="1.0"
   xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
   xmlns:php="http://php.net/xsl">
<xsl:output method="html" encoding="ISO-8859-1"/>
<xsl:template match="team">
  <h1><xsl:value-of select="@id"/></h1>
  <ul>
  <xsl:for-each select="members/member">
    <li>
      <xsl:value-of select="@alias"/>
    <xsl:value-of select="php:functionString('md5',
   name)"/>)
        </li>
  </xsl:for-each></ul>
</xsl:template>
```

XSLT: php:functionString()

```
$xsl = DomDocument::load("stylesheet3.xsl");
$xml = DomDocument::load("example.xml");

$proc = new XsltProcessor();
$proc->registerPhpFunctions();
$proc->importStylesheet($xsl);
print $proc->transformToXML($xml);
```

- Call any PHP function available in userland
- `php:function()` passed the DOMNode instead of a string

XSLT: php:functionString()

```
<h1 xmlns:php="http://php.net/xsl">JLA</h1>
<ul xmlns:php="http://php.net/xsl">
  <li>Superman (d82f6c9e46b92c3100ea87c0777c805d)</li>
  <li>Aquaman (28cab32809f8bdcd136abe0c6e927eb4)</li>
</ul>
<h1 xmlns:php="http://php.net/xsl">JSA</h1>
<ul xmlns:php="http://php.net/xsl">
  <li>
    Power Girl (5a25ff27398b7874e7eb64a9e315763c)
  </li>
  <li>
    Black Canary (845bcd11235ee36cd19b942d02d1c194)
   </li>
  <li>Hawkman(d24a0fc8b0c0943ca4229cb7a94687b6)</li>
</ul>
```

XSLT: Disatvantages

- Extremely verbose
- No access to PHP functions or the stylesheet is not portable anymore
- **`registerPHPFunctions()`** may open security holes
- The XSLT processor is a blackbox, no way to control the transformation, once it has been started.

Agenda: XML in PHP 5.1

- xmlReader
- DOM improvements
- XPath improvements
- XSL improvements
- Error Handling
- Interop between extensions

xmlReader

- Available through PECL
- Bundled with PHP 5.1.x
- Uses XML-Pull
  - Traverses document like SAX
  - No callbacks, you control the cursor in the document

xmlReader: Example

```
$reader = new xmlReader();
$reader->open('example.xml');

while ($reader->read()) {
  switch ($reader->nodeType) {
    case XMLREADER_ELEMENT:
          echo "Tag: " . $reader->name . "\n";
          break;
    case XMLREADER_TEXT:
      print "Data: ".$reader->value . "\n";
      break;
        }
}
$reader->close();
```

xmlReader: Example

```
$reader = new xmlReader();
$reader->open('example.xml');

while ($reader->read()) {
  switch ($reader->nodeType) {
    case XMLREADER_ELEMENT:
          echo "Tag: " . $reader->name . "\n";
          break;
    case XMLREADER_TEXT:
      print "Data: ".$reader->value . "\n";
      break;
        }
}
$reader->close();
```

xmlReader: Example

```
Tag: teams
Tag: team
Tag: name
Data: Justice League of America
Tag: members
Tag: member
Tag: name
Data: Clark Kent
Tag: powers
Tag: power
Data: Super-Strength
Tag: power
Data: Heat Vision
…
```

xmlReader: Example 2

```
$reader = new xmlReader();
$reader->open('example.xml');

$current = null;
$heroes = array();
$currentTeam = null;

while ($reader->read()) {
  switch ($reader->nodeType) {
    case XMLREADER_ELEMENT:
      switch ($reader->name) {
        case 'team':
          $current = null;
          $currentTeam = $reader->getAttribute('id');
          break;
```

xmlReader: Example 2

```
    case 'member':
        $current = $reader->getAttribute('alias');
        $heroes[$current] = array();
        $heroes[$current]['gender'] =
                     $reader->getAttribute('gender');
        $heroes[$current]['powers'] = array();
        $heroes[$current]['team']   = $currentTeam;
        break;
```

xmlReader: Example 2

```
    case 'name':
        if ($current === null) {
          continue;
        }
        if ($reader->getAttribute('secret') == 'yes') {
          $heroes[$current]['name'] = 'Confidential';
          continue;
        }
        $reader->read();
        $heroes[$current]['name'] = $reader->value;
        break;
```

xmlReader: Example 2

```
         case 'power':
           $reader->read();
           array_push($heroes[$current]['powers'],
                      $reader->value);
           break;
       }
       break;
    }
}
$reader->close();
print_r($heroes);
```

xmlReader: Example 2

```
Array (
    [Superman] => Array (
            [gender] => male
            [powers] => Array (
                    [0] => Super-Strength
                    [1] => Heat Vision
                )
            [team] => JLA
            [name] => Confidential
        )
     [Aquaman] => Array (……)
     …
)
```

xmlReader: next()

- **`read()`** always moves the pointer to the next token
- If searching for data, **`next()`** lets you skip several tags by moving the cursor to the next tag on the same level:

```
// cursor is on a <team/> tag
if ($reader->getAttribute('id') !== 'JSA') {
  // move to next team
  $reader->next();
}
```

xmlReader: Features

- Supports validation
- full namespace support
- support for xmlLang
- iterate attributes without knowing their names
- parse from string instead of a file

xmlReader vs. SAX

- both have very low memory footprint
- xmlReader is faster (no callbacks)
- xmlReader supports validation
- SAX is available everywhere

If you can, use xmlReader.

DOM: Broken XML

```
$xml = <<<EOT
<hero>
  <name>Clark Kent</name>
  <alias>Superman
</hero>
EOT;

$dom = new DOMDocument();
$dom->recover = true;
$dom->loadXML($xml);
print $dom->saveXML();
```

DOM: Broken XML

PHP 5.0.x

```
$ php5 dom.php
Warning: DOMDocument::loadXML(): Opening and ending tag
   mismatch: alias line 3 and hero in Entity, line: 4
   in dom.php on line 11
Warning: DOMDocument::loadXML(): Premature end of data
   in tag hero line 1 in Entity, line: 4 in dom.php on
   line 11
<?xml version="1.0"?>
```

DOM: Broken XML

PHP 5.1-dev

```
$ php5-dev dom6.php
Warning: DOMDocument::loadXML(): Opening and ending tag
   mismatch: alias line 3 and hero in Entity, line: 4
   in /dom.php on line 11
Warning: DOMDocument::loadXML(): Premature end of data
   in tag hero line 1 in Entity, line: 4 in dom.php on
   line 11
<?xml version="1.0"?>
<hero>
  <name>Clark Kent</name>
  <alias>Superman
</alias></hero>
```

XPath: evaluate()

- `DOMXPath::query()` only returns DOMNodeList objects
- Not possible to return types results
- `DOMXPath::evaluate()` allows you to use expressions like `count()`

XPath: evaluate()

```
$dom   = DOMDocument::load('example.xml');
$xpath = new DOMXPath($dom);

$query  = 'count(//member[@gender="female"])';
$result = $xpath->evaluate($query);
print "The teams have $result female members.\n";

$query  = 'count(//power[.="Flight"])';
$result = $xpath->evaluate($query);
print "$result heroes are able to fly.\n";
```

```
The teams have 2 female members.
2 heroes are able to fly.
```

XSL: improved security

```
// register all functions
$xsl->registerPHPFunctions();


$xsl->registerPHPFunctions(
                array("date","myFunc")
    );
$xsl->registerPHPFunctions("date");
```

Great for stylesheets from untrusted sources like file uploads

XML: error handling

Ability to disable PHP notices and fetch all errors at once.

```
libxml_use_internal_errors(true);
$ret = $dom->load($file);
if (!$ret) {
  $errors = libxml_get_errors();
  foreach ($errors as $error) {
    printf("%s in file %s on line %d\n",
          $error->message, $error->file, $error->line);
  }
}
```

Interop between extensions

xmlReader to DOM:

```
$reader = new xmlReader();
$reader->open('example.xml');
while ($reader->read()) {
  if ($reader->nodeType != XMLREADER_ELEMENT) {
    continue;
  }
  if ($reader->name != 'member') {
    continue;
  }
  break;
}
$domNode = $reader->expand();
print $domNode->getAttribute('alias');
```

Interop between extensions

simpleXML to DOM to simpleXML:

```
// load simplexml
$teams = simplexml_load_file('example.xml');

// import <team id="JLA"/> to dom
$dom   = dom_import_simplexml($teams->team[0]);
print $dom->getAttribute('id') . "\n";

// import this DOM object to a new simpleXML document
$jla   = simplexml_import_dom($dom);
print $jla->members->member[0]->name . "\n";
```

Agenda: XML in PECL

- xmlReader (already dealt with it)
- xmlWriter

xmlWriter

- Available through PECL
- Write XML-documents to a stream
  - memory
  - filesystem
- Extremely fast
- Always creates well-formed documents
- Supports indentation

xmlWriter: Example

```
$heroes = array(
    'Superman' => array(
           'name'   => 'Clark Kent',
           'gender' => 'male',
           'powers' => array('Flight', 'Strength')
                      ),
    'Power Girl' => array(
           'name'   => 'Karen Star',
           'gender' => 'female',
           'powers' => array('Strength')
                     ),
          );
```

xmlWriter: Example

```
$xw = xmlwriter_open_memory();

xmlwriter_set_indent($xw, 1);
xmlwriter_set_indent_string($xw, '   ');

xmlwriter_start_document($xw, '1.0');
xmlwriter_start_element($xw, 'team');
xmlwriter_write_attribute($xw, 'id', 'Allstars');
```

xmlWriter: Example

```
foreach ($heroes as $alias => $hero) {
  xmlwriter_write_comment($xw, " $alias ");
  xmlwriter_start_element($xw, 'member');
  xmlwriter_write_attribute($xw, 'alias', $alias);
  xmlwriter_write_attribute($xw, 'gender',
                                   $hero['gender']);
  xmlwriter_start_element($xw, 'name');
  xmlwriter_text($xw, $hero['name']);
  xmlwriter_end_element($xw);

  xmlwriter_start_element($xw, 'powers');
   …
  xmlwriter_end_element($xw);
 xmlwriter_end_element($xw);
}
```

xmlWriter: Example

```
xmlwriter_end_element($xw);
xmlwriter_end_document($xw);
print xmlwriter_output_memory($xw, true);
```

```
<?xml version="1.0"?>
<team id="Allstars">
   <!-- Superman -->
   <member alias="Superman" gender="male">
      <name>Clark Kent</name>
      <powers>
         <power>Flight</power>
         <power>Strength</power>
      </powers>
   </member>
   …
</team>
```

Agenda

- Introduction
- Introduction to XML
- XML in PHP 5.0/5.1 & PECL
- PEAR
- XML in PEAR
- Introduction to Webservices
- Webservices in PHP 5.0/5.1 & PECL
- Webservices in PEAR
- Q&A session

Agenda - PEAR

- ...
- PEAR
  - What is PEAR?
  - Obtaining PEAR
  - The installer
  - Using PEAR
  - PEAR_Error
  - Getting help
  - Future outlook
- ...

What is PEAR?

- Collection of high quality PHP components
- Nearly 300 packages and growing fast
- Almost 200 package maintainers, 500 developers
- 100% free (PHP, Apache, BSD, LGPL licenses)
- Standardization institution
- Founded by Stig S. Bakken in 1999

Obtaining PEAR

- PEAR installer shipped with PHP since 4.3.0
- Automatically installed on Windows
- Per default activated when compiling on *nix
  (do not use “--without-pear”)
- For earlier PHP versions bootstrap from
  http://go-pear.org
- On *nix try
  ```
  lynx -source http://pear.php.net/go-pear | php -q
  ```
- On Windows, save source and call PHP manually
  ```
  php -q go-pear.php
  ```

The PEAR Installer

- Different interfaces:
  - Console (build in)
  - Web
  - GTK
- Easy usage
- Perform a lot of actions on packages:
  - List local/remote
  - Install/Uninstall/Upgrade directly from the web
  - Get package information
  - Dependencies
  - Package packages
  - Test packages

The PEAR Installer

- Important PEAR Installer commands
  - `$> pear [un]install [PackageName]`
  - `$> pear upgrade[-all] [PackageName]`
    - Use -f option to force action
    - Instead of PackageName point to package.xml or tar.gz in the filesystem or URL
  - `$> pear list[-upgradeable]`
  - `$> pear config-show`
  - `$> pear config-set`
  - `$> pear package[-validate] [package.xml]`

The PEAR Installer

Live demo

(Hopefully the network is available...)

Using PEAR

- Important precondition:
  - include_path must contain the correct path to PEAR!
- Packages contain a main file, which has to be included (no others).
- Package names map to their location in PEAR:
  - DB --> DB.php
  - Net_FTP --> Net/FTP.php
- Class names map to package names:
  - DB --> DB()
  - Net_FTP --> Net_FTP()
  - (attention, most packages do not use direct instantiation)

PEAR_Error

- PEAR standard for error handling
- Designed for PHP4
- Somewhat following the exception concept
- Works with error handling
- Allows definition of global and local error handlers:
  - PEAR_ERROR_RETURN
  - PEAR_ERROR_DIE
  - PEAR_ERROR_CALLBACK
- Works in cooperation with
  - PEAR_ErrorStack
  - PEAR_Exception

PEAR_Error

- Example:

```
<?php
function foo () {
    PEAR::raiseError('An error occurred', -1);
}
function errorHandler ($error) { echo $error->getMessage(); }

if (PEAR::isError(foo()) { ... }

PEAR::setErrorHandling(PEAR_ERROR_CALLBACK, 'errorHandler');
foo();

?>
```

Getting help

1. The PEAR Website
   - http://pear.php.net/package/<package_name>
   - http://pear.php.net/manual/en/
2. Google! (http://www.google.com)
3. Support Mailinglist <pear-general@lists.php.net>
4. IRC channel: #pear@EFNet
5. Direct maintainer contact (see package website!)
6. PEAR QA <pear-qa@lists.php.net>

- Emergency: PEAR Group <pear-group@lists.php.net>
- Website problems <pear-webmaster@lists.php.net>

Future outlook

- General route
  - More and more PHP5 specific
  - Installer improvments
  - Growing collection of packages
- The PEAR Installer
  - Version 1.4 currently in alpha stadium:
    - Channel support
    - Automatic dependency resolving
    - ...
      Wanna know more? **PEAR session on Wendsday!**

Usefull links

- http://pear.php.net
- http://pear.php.net/support/slides.php
- http://www.php-mag.net

Agenda

- Introduction
- Introduction to XML
- XML in PHP 5.0/5.1 & PECL
- PEAR
- XML in PEAR
- Introduction to Webservices
- Webservices in PHP 5.0/5.1 & PECL
- Webservices in PEAR
- Q&A session

Agenda – XML in PEAR

- ...
- XML in PEAR
  - Overview
  - General XML packages
    XML_Parser, XML_Util, XML_Serializer, XML_FastCreate
  - XML processing packages
    - XML_RSS, XML_FOAF
  - XML creation packages
    - XML_XUL, XML_sql2xml
  - Misc XML packages
    - XML_Statistics, XML_Beautifier
- ...

Overview

- XML is one of the largest sections
- Growing very fast
- Provides a solid basis for XML directed development
- Will make XML development a lot easier for you

- The PEAR Group holds a very high ensurance on Stephan's live!

XML_Parser

- OO wrapper to ext/XML (SAX based)
- Provides 2 easy callback APIs:
  - func (calling different methods for tags)
  - event (calling a single method for tags)
- Provides convenience methods
  - Reading XML from any stream
  - Using strings and resources as source
- Not used directly, class must be extended for usage
- Used in many PEAR packages
- Uses PEAR error handling

XML_Parser

- Example -1-:

```
class myParser extends XML_Parser
{
  function myParser() { parent::XML_Parser(); }
  function startHandler($xp, $name, $attribs) {
    printf('handle start tag: %s<br />', $name);
  }
  function endHandler($xp, $name) {
    printf('handle end tag: %s<br />', $name);
  }
  function cdataHandler($xp, $cdata) {
    print $cdata;
  }
}
```

XML_Parser

- Example -2-:

```
$p = &new myParser();
$result = $p->setInputFile('example.xml');
$result = $p->parse();
```

Usefull links

- http://pear.php.net/package/XML_Parser
- http://cvs.php.net/pear/XML_Parser/examples/

XML_Util

- Collection of commonly needed functions for XML manipulation:
  - Collapsing empty tags (<foo></foo> => <foo />)
  - Rendering tags (by tagname, attributes array, automatic escaping...)
  - Validation of data
- Advantages
  - No more syntax errors when creating XML
  - Much faster development

try $> `pear install XML_Util`

XML_Util

- Example:

```
// Creating a tag
$tag = XML_Util::createTag(
           'xsl:stylesheet',
           array('version' => '1.0'),
           'Place your content here',
           'http://www.w3.org/1999/XSL/Transform');
// Validate tag name
$result = XML_Util::isValidName("my Tag");
// Collapse empty XHTML tags
$result = XML_Util::collapseEmptyTags(
           '<p><img ...></img></p><p></p>',
           XML_UTIL_COLLAPSE_XHTML_ONLY);
```

Usefull links

- http://pear.php.net/package/XML_Util

XML_Serializer

- Swiss Army Nife for generation of XML
- Serialize/Unserialize complete data structures from PHP to XML:
  - Scalars
  - Arrays
  - Objects
- Advantages:
  - Create XML dialects (like RSS & Co.) very easily and fast.
  - Transport PHP data structures easily.
  - Save PHP data in a human readable way (unlike serialze()).

XML_Serializer

- Example -1-:

```
$fruits = array('pear', 'banana', 'smarty');
$serializer_options = array (
   'addDecl' => TRUE,
   'encoding' => 'ISO-8859-1',
   'indent' => '  ',
   'rootName' => 'fruits',
   'defaultTagName' => 'fruit',
);
$Serializer = &new XML_Serializer($serializer_options);
$result = $Serializer->serialize($fruits);
header('Content-type: text/xml');
echo $Serializer->getSerializedData();
```

XML_Serializer

- Result from example -1-:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<fruits>
  <fruit>pear</fruit>
  <fruit>banana</fruit>
  <fruit>smarty</fruit>
</fruits>
```

XML_Serializer

- Example -2-:

```
class Basket {
   var $pears = 10;       var $bananas = 2;
}
$serializer_options = array (..., 'typeHints' => true);
...
$xml = $serializer->getSerializedData();
$unserializer_options = array(
    'returnResult' => true,
);
$unserializer = &new XML_Unserializer($unserializer_options);
var_dump($unserializer->unserialize($xml));
```

XML_Serializer

- Results from Example -2-:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<basket _class="basket"
        _type="object">
  <pears _type="integer">10</pears>
  <bananas _type="integer">2</bananas>
</basket>

object(basket)(2) {
  ["pears"]=>
  int(10)
  ["bananas"]=>
  int(2)
}
```

Usefull links

- http://pear.php.net/package/XML_Serializer

XML_FastCreate

- Easy way to create valid XML
- Driver based output (String, XML_Tree, ...)
- DTD validation
- HTML -> XHTML conversion

XML_FastCreate

- Example:

```
$x =& XML_FastCreate::factory('Text',
    array(
        'doctype'   => XML_FASTCREATE_DOCTYPE_XHTML_1_0_STRICT,
    )
);
$x->html(
    $x->head(  $x->title('Fruitmix website') ),
    $x->body(  $x->p(  $x->cdata('Welcome to Fruitmix!') ) )
);
$x->toXML();
```

XML_FastCreate

- Result generated by example:

```
<?xml version="1.0" encoding="UTF-8" standalone="no" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
<head><title>Fruitmix website</title></head>
<body>
<p>/*<![CDATA[*/Welcome to Fruitmix! /*]]>*/</p>
</body>
</html>
```

Usefull links

- http://pear.php.net/package/XML_FastCreate

XML_RSS

- RSS saves articles as an RDF format
- Retreive RSS feeds in PHP
- Comfortable retrieval of RSS data.
- Supports multiple RSS versions.
- Example:

```
$rss =& new XML_RSS('http://www.planet-php.net/rdf/');
$rss->parse();
foreach ($rss->getItems() as $item) {
    echo $item['link'] . '<br />';
    echo $item['title'] . '<br />';
}
```

Usefull links

- http://pear.php.net/package/XML_RSS

XML_FOAF

- FOAF = Friend Of A Friend
- Sweet little XML dialect :)
- Describes people, what they do and their relations to other people using RDF syntax
- Used to build virtual networks (like Orkut, OpenBC, Friendster, ...)
- Package allows parsing and creation

XML_FOAF

- Example:

```
$foaf = new XML_FOAF();

$foaf->newAgent('person');
$foaf->setName('Tobias Schlitt');
$foaf->setTitle('Mr');
$foaf->setFirstName('Tobias');
$foaf->setSurname('Schlitt');
$foaf->addMbox('mailto:toby@php.net',TRUE);
$foaf->addHomepage('http://www.schlitt.info');

echo $foaf->get();
```

XML_FOAF

- Result from example:

```
<rdf:RDF ... xmlns:foaf="http://xmlns.com/foaf/0.1/">
<foaf:Person>
<foaf:name>Tobias Schlitt</foaf:name>
<foaf:title>Mr</foaf:title>
<foaf:firstName>Tobias</foaf:firstName>
<foaf:surname>Schlitt</foaf:surname>
<foaf:mbox_sha1sum>92c00d31...</foaf:mbox_sha1sum>
<foaf:homepage rdf:resource="http://www.schlitt.info" />
...
</foaf:Person>
</rdf:RDF>
```

Usefull links

- http://pear.php.net/package/XML_FOAF
- http://www.foaf.org

XML_XUL

- XUL = XML User Interface Language
- Invented by the Mozilla Foundation to describe rich applications in a browser
- Widly used for Mozilla extensions
- Not yet used massivly for web applications
- Package allows creation of XUL interface definitions through PHP

XML_XUL

- Example:

```
$doc = &XML_XUL::createDocument( );
$doc->addStylesheet('chrome://global/skin/');
$win = &$doc->createElement('window', array('title'=> 'Example'));
$doc->addRoot($win);
$box = &$doc->createElement('box', array('align' => 'center', 'orient' => 'vertical'));
$win->appendChild($box);
$browser = &$doc->createElement('browser', array('width' => 800, 'height'=> 500, 'src'
    => 'http://pear.php.net', 'id' => 'myBrowser'));
$box->appendChild($browser);
...
$doc->send();
```

XML_XUL

- Result from example:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<?xml-stylesheet href="chrome://global/skin/" type="text/css"?>
<window title="Example" xmlns="http://www.mozilla.org/...">
  <box align="center" orient="vertical">
    <browser height="500" id="myBrowser" src="http://pear.php.net"
     width="800" />
...
  </box>
</window>
```

XML_XUL

Live demo

Usefull links

- http://pear.php.net/package/XML_XUL
- http://www.xulplanet.com/
- http://www.mozilla.org/projects/xul/

XML_sql2xml

- Create XML docs from different input:
  - SQL statement
  - DB_Result object
  - Array
- Usefull when you like XML more than anything else
  - XSLT lovers
  - Popoon users
  - ...
- Supports nested result sets

```
try $> pear install XML_sql2xml
```

XML_sql2xml

- Example

```
$sql2xml = new xml_sql2xml("mysql://...");
$xml = $sql2xml->getxml("select * from peardevs");
```

- Result

```
<root> <result> <row>
        <id>1</id>
        <name>PEAR</name>
        <birth_year>1999</birth_year>
        <founder>Stig S. Bakkenl</founder>
</row> ... </result> </root>
```

Usefull links

- http://pear.php.net/package/XML_sql2xml

XML_Statistics

- Generate statistics about different emelemts of your XML documents
  - Tags
  - Attributes
  - Processing instructions
  - Entities
  - CDATA blocks
- Filter elements to count

XML_Statistics

- Example

```
$stats = new XML_Statistics(array("ignoreWhitespace" => true));
$res = $stats->analyzeFile("example.xml");

$stats->countTag();                        // all tags
$stats->countAttribute("id");              // all attributes id
$stats->countAttribute("id", "section");   // attributes id in tags section
$stat->countTagsInDepth(2);                // tags on the second tag level
```

Usefull links

- http://pear.php.net/package/XML_Statistics

XML_Beautifier

- Beautify your XML
- Fix
  - Indentation
  - Line breaks
  - Entities
  - Comments
- Easy to use
- Allows much better reading of XML docs

XML_Beautifier

- Example

```
$xml = '<?xml version="1.0" encoding="ISO-8859-1"?>
    <document title="On pears and pecls"><meta project="PHP" id="1">
      <keywords/><description/><author>Toby</author>
    <getMetaNav/></meta><code type="php">
    <?php
      echo $pears . " & " . $pecls;
    ?><!-- This Comment has more
      than one line.  -->
    </code></document>';
$beauty = new XML_Beautifier();
var_dump($beauty->formatString($xml));
```

XML_Beautifier

- Result from example

```
<?xml version="1.0" encoding="ISO-8859-1" standalone="yes"?>
<document title="On pears and pecls">
   <meta id="1" project="PHP">
      <keywords />
      <description />
      <author>Toby</author>
      <getMetaNav />
   </meta>
   <code type="php">
      <?php
      echo $pears . " & " . $pecls;
      ?>
```

```
<!--
                  This Comment has more
                  than one line.

-->
      </code>
</document>
```

try `$> pear install XML_Beautifier`

Usefull links

- http://pear.php.net/package/XML_Beauttifier

Agenda

- Introduction
- Introduction to XML
- XML in PHP 5.0/5.1 & PECL
- PEAR
- XML in PEAR
- Introduction to Webservices
- Webservices in PHP 5.0/5.1 & PECL
- Webservices in PEAR
- Q&A session

Agenda: Webservices

- Why?
- How?
- XML-RPC
- SOAP
- REST
- Related technologies

Webservices: Why?

- Businesses, applications and websites grow
- Heterogenic environment
- Need to connect the different systems using a standard that is agreed upon
- interoperability between various software applications running on disparate platforms

Webservices: How?

- Use proven technologies and protocols
- HTTP (or SMTP) for transportation
- XML for encoding

XML-RPC

- XML Remote Procedure Call
- Created by Dave Winer of Userland Software in 1995
- Call procedures on a remote host
  - HTTP as protocol
  - XML as the encoding
- As simple as possible

XML-RPC

- Simple and complex data types possible
  - strings, booleans, integers, doubles, date/time, arrays, structs, base64-encoded data
- Server returns Fault-object on error
- Parameters are ordered, not named
- Implementations for nearly any language available

XML-RPC

```
<?xml version='1.0' encoding="iso-8859-1" ?>
<methodCall>
  <methodName>
    pat.checkForUpdate
  </methodName>
  <params>
    <param>
      <value><string>patTemplate</string></value>
    </param>
    <param>
      <value><float>3.0.1</float></value>
    </param>
  </params>
</methodCall>
```

SOAP

- Once the abbreviation for Simple Object Access Protocol
- Since v1.1 only SOAP
- Evolved from XML-RPC
- Developed by Microsoft and Dave Winer
- Uses namespaces and XML schema

REST

- Representational State Transfer
- Data-centered
- leverages the HTTP protocol
  - GET, POST, PUT, DELETE are all verbs you need
  - parameters are passed using name=value
- Returns XML

REST

- A lot easier to implement
- loosely typed / everything is a string
- interpretation of the data is left to the developer
- Becomes more and more popular
  - Amazon, eBay, Yahoo, Flickr, del.ico.us
- Even the new PEAR installer uses it

REST: Example

```
GET /WebSearchService/V1/webSearch?appid=…&query=PHP5
Host: api.search.yahoo.com
```

```
<?xml version="1.0" encoding="UTF-8"?>
<ResultSet …… totalResultsAvailable="414045">
  <Result>
    <Title>PHP: Hypertext Preprocessor</Title>
    <Summary>…What is PHP? PHP is a…</Summary>
    <Url>http://www.php.net/</Url>
    <ClickUrl>http://rds.yahoo.com/S=96857..</ClickUrl>
    <ModificationDate>1111305600</ModificationDate>
    <MimeType>text/html</MimeType>
 </Result>
...
</ResultSet>
```

Related Technologies

- Tons of buzzwords surround webservice technologies
- Most of them are XML applications
  - UDDI
  - WSDL
- And you won't need most of them

WSDL

- Web Service Description Language
- Defines the API of a web service
- Allows clients to consume web services without knowing which functions the service offers
- Defines custom data-types

WSDL: Example

```
<?xml version ='1.0' encoding ='UTF-8' ?>
<definitions name='Encrypt'
  targetNamespace='http://example.org/Encrypt'
  xmlns:tns=' http://example.org/Encrypt '
  xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:soapenc='http://schemas.xmlsoap.org/soap/encod
   ing/'
  xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
  xmlns='http://schemas.xmlsoap.org/wsdl/'>
```

WSDL: Example

```
<message name='encryptRequest'>
  <part name='passwort' type='xsd:string'/>
  <part name='methode' type='xsd:string'/>
</message>
<message name='encryptResponse'>
  <part name='Result' type='xsd:string'/>
</message>

<portType name='encryptPortType'>
  <operation name='encrypt'>
    <input message='tns:encryptRequest'/>
    <output message='tns:encryptResponse'/>
  </operation>
</portType>
```

WSDL: Example

```
<binding name='encryptBinding'
         type='tns:encryptPortType'>
  <soap:binding style='rpc'
    transport='http://schemas.xmlsoap.org/soap/http'/>
  <operation name='encrypt'>
    <soap:operation
     soapAction='urn:xmethods-delayed-quotes#encrypt'/>
    <input>
      <soap:body use='encoded'
            namespace='urn:xmethods-delayed-quotes'
            encodingStyle='..'/>
    </input>
```

WSDL: Example

```
    <output>
      <soap:body use='encoded'
            namespace='urn:xmethods-delayed-quotes'
            encodingStyle='http://.../soap/encoding/'/>
    </output>
  </operation>
</binding>
<service name='encryptService'>
  <port name='encryptPort' binding='encryptBinding'>
    <soap:address location='http://example.com/soap'/>
  </port>
</service>
</definitions>
```

UDDI

- Universal Description, Discovery, and Integration
- XML-based registry for businesses
- Never used this in real-life
- In 99.9% of all cases, you know which service you need to consume
- If not, ask Google :)

Agenda

- Introduction
- Introduction to XML
- XML in PHP 5.0/5.1 & PECL
- PEAR
- XML in PEAR
- Introduction to Webservices
- Webservices in PHP 5.0/5.1 & PECL
- Webservices in PEAR
- Q&A session

Agenda: PHP5 Webservices

- Nothing had been done for XML-RPC
- Completely revamped SOAP extension (sponsored by Zend)
- SOAP extension supports WSDL
- REST can be easily consumed
  - Streams provide HTTP support
  - Choose any XML-extension to parse the result

XML-RPC

- Nothing new on the XML-RPC front in PHP5
- XML-RPC functions are available since PHP 4.1
- Clumsy and not very intuitive
- Alternative: PEAR XML_RPC, which uses those functions, if available

SOAP

- Completely revamped SOAP extension
- Supports WSDL
- Provides client and server implementations
- Uses PHP5's new object overloading to build intuitive proxy clients

SOAP: Client Example

```
$apiKey = 'Go and register your own…';
$client = new
   SoapClient('http://api.google.com/GoogleSearch.wsdl'
   );
$result = $client->doGoogleSearch(
  $apiKey, "php5", 0, 10, false, '', true, '', '', '');

printf("Estimated total result of %d pages\n",
       $result->estimatedTotalResultsCount);

$i = 0;
foreach ($result->resultElements as $page) {
    printf("%d. %s\n",++$i,utf8_decode($page->title));
}
```

SOAP: Client Example

```
Estimated total result of 396000 pages
1. PHP: Downloads
2. PHP: Hypertext Preprocessor
3. PHP: PHP 5 ChangeLog
4. PHP: Classes and Objects (PHP 5) - Manual
5. Zend Technologies - PHP 5 InfoCenter - Information,
   Articles and <b>...</b>
6. PHPBuilder.com, the best resource for PHP tutorials,
   templates <b>...</b>
7. <b>PHP5</b>: Coming Soon to a Webserver Near You
   [PHP &amp; MySQL Reviews and <b>...</b>
8. PHPVolcano
9. ONLamp.com: Why PHP 5 Rocks!
…
```

WSDL Support

- WSDL file will be parsed on the first request
- Can be cached by ext/soap using php.ini settings

```
soap.wsdl_cache_enabled = "1"
soap.wsdl_cache_dir = "/tmp"
soap.wsdl_cache_ttl = "86400"
```

SOAP: Without WSDL

```
$client = new SoapClient(NULL,
 array(
  "location"   => "http://api.google.com/search/beta2",
  "uri"        => "urn:GoogleSearch",
  "style"      => SOAP_RPC,
  "use"        => SOAP_ENCODED,
  "exceptions" => 0
     )
);
```

SOAP: Without WSDL

```
$params = array(
       new SoapParam('Get you own key!', 'key'),
       new SoapParam('php5', 'q'),
       new SoapParam(0, 'start'),
       new SoapParam(10, 'maxResults'),
       new SoapParam(false, 'filter'),
       new SoapParam('', 'restrict'),
       new SoapParam(false, 'safeSearch'),
       new SoapParam('', 'lr'),
       new SoapParam('', 'ie'),
       new SoapParam('', 'oe')
         );
```

SOAP: Without WSDL

```
$options = array(
     'uri'        => 'urn:GoogleSearch',
     'soapaction' => 'urn:GoogleSearch#doGoogleSearch'
                );
$result = $client->__call('doGoogleSearch', $params,
                          $options);

printf("Estimated total result of %d pages\n",
       $result->estimatedTotalResultsCount);
$i = 0;
foreach ($result->resultElements as $page) {
    printf("%d. %s\n",++$i,utf8_decode($page->title));
}
```

SOAP: Error handling

Constructor allows you to define the desired error handling.

- Exceptions (default)
- SoapFault-Object as return value.

Which one you are using is just a matter of preference.

SOAP: Exceptions

```
$options = array(
                  'exceptions' => 1
               );
$client = new SoapClient('…', $options);
try {
  $result = $client->doGoogleSearch(…);
} catch (SoapFault $f) {
  print "Error using SOAP-Service:\n";
  print $f;
}
```

SOAP: Error Object

```
$options = array(
                  'exceptions' => 0
               );
$client = new SoapClient('…', $options);
$result = $client->doGoogleSearch(…);
if (is_soap_fault($result)) {
  print "Error using SOAP-Service:\n";
  print $result->faultstring . "\n";
}
```

SOAP: Server

- Write a class or functions using plain PHP syntax
- Write your WSDL file
- Bind PHP class to the service using WSDL
- Create new server object
- Start the server

SOAP: Server Example

```
class CryptServer {
  function encrypt($pass, $type) {
    switch ($type) {
      case 'md5':
        return md5($passwort);
        break;
      case 'md5rev':
        return strrev(md5($passwort));
        break;
      default:
       throw new SoapFault('Server', 'Unkown type');
    }
  }
}
```

SOAP: Server Example

```
<message name='encryptRequest'>
  <part name='pass' type='xsd:string'/>
  <part name='methode' type='xsd:string'/>
</message>
<message name='encryptResponse'>
  <part name='Result' type='xsd:string'/>
</message>

<portType name='encryptPortType'>
  <operation name='encrypt'>
    <input message='tns:encryptRequest'/>
    <output message='tns:encryptResponse'/>
  </operation>
</portType>
```

SOAP: Server Example

```
<service name='encryptService'>
  <port name='encryptPort' binding='encryptBinding'>
    <soap:address
            location='http://example.com/soap.php'/>
  </port>
</service>
```

SOAP: Server Example

```
$server = new SoapServer("cryptServer.wsdl");
$server->setClass("CryptServer");
$server->handle();
```

Consuming the service

```
$client = new
   SoapClient('http://example.com/cryptServer.wsdl');
try {
  $crypt   = $client->encrypt('myPass', 'md5');
  $cryptw  = $client->encrypt('myPass', 'md5rev');
} catch (SoapFault $f) {
  print $f;
}
```

SOAP: Persistence

- Allows you to save data on the server between several requests
- Only works with methods that have been exported from a class using **`setClass()`**
- Only works with clients that use PHP5's ext/soap
- Uses PHP's session handling

SOAP: Problems

- SOAP is extremely complex
- Specifications are too imprecise in some fields
- Often problems when using different client implementations, e.g.
  - Java Server
  - PHP, C# or ASP client
- Too verbose

Webservices in PECL

- Currently only XMLRPCi

XMLRPCi

- Available through PECL
- Meant to replace PHP's XML-RPC extension
- Uses PHP5's object overloading to create intuitive clients
- very new and still in beta stage

XMLRPCi: Client Example

```
$client = new
  XMLRPC('http://betty.userland.com/RPC2'
  , 'examples.');


$state  = $client->getStateName(32);
print "I love $state!\n";
```

```
I love New York!
```

XMLRPCi: Request

```
<?xml version='1.0' encoding="iso-8859-1" ?>
<methodCall>
  <methodName>
    examples.getStateName
  </methodName>
  <params>
    <param>
      <value><int>32</int></value>
    </param>
  </params>
</methodCall>
```

XMLRPCi: Response

```
<?xml version="1.0"?>
<methodResponse>
  <params>
    <param>
      <value>New York</value>
    </param>
  </params>
</methodResponse>
```

XMLRPCi: Example 2

```
$package = new
  XMLRPC('http://pear.php.net/xmlrpc.php',
  'package.');

try {
    $result = $package->info('HTTP_Server');
} catch (XMLRPC_Fault $fault) {
    echo "Error sending request\n";
    exit();
}
print $result['description'] . "\n";
```

XMLRPCi: Server

- XMLRPCi 1.0 does not provide a server implementation
- CVS already provides a server
- Could not get it to work with latest CVS version of PHP 5.1-dev

XMLRPCi: Server

```
$server = new XMLRPCServer();
$server->addFunction("multiply");
$a->handle();

function myfunction($a, $b) {
  if (!is_int($a) || !is_int($b)) {
    throw new XMLRPCFault("You must pass
                                integers!", 10);
  }
  return $a * $b;
}
```

Agenda

- Introduction
- Introduction to XML
- XML in PHP 5.0/5.1 & PECL
- PEAR
- XML in PEAR
- Introduction to Webservices
- Webservices in PHP 5.0/5.1 & PECL
- Webservices in PEAR
- Q&A session

Agenda – XML in PEAR

- ...
- Webservices in PEAR
  - XML_RPC
  - SOAP
  - Non-standard Webservices
    - Services_Google
    - Services_Amazon
    - Services_Delicious
    - Services_Yahoo
    - Services_Ebay
    - Services_Trackback
- ...

XML_RPC

- PHP based implementation of the XML-RPC protocoll
- Uses HTTP as underlying protocoll
- Allows to create clients and servers
- Nearly outdated: ext/XML-RPCi

XML_RPC

- Example (XML-RPC client)

```
$params = array(new XML_RPC_Value(1, 'int'));

$msg = new XML_RPC_Message('release.getRecent', $params);
$cli = new XML_RPC_Client('/xmlrpc.php', 'pear.php.net');
$resp = $cli->send($msg);

$val = $resp->value();
$data = XML_RPC_Decode($val);

echo $data[0]['name'] . ' is at version ';
echo $data[0]['version'];
```

XML_RPC

- Example (XML-RPC server)

```
function returnSquare($params) {
    $param = $params->getParam(0);
    $val = new XML_RPC_Value(sqr($param->scalarval()), 'int');
    return new XML_RPC_Response($val);
}
$server = new XML_RPC_Server(
        array(
            'suqare' => array('function' => 'returnSquare')
        )
);
```

Usefull links

- http://pear.php.net/package/XML_RPC http://www.xmlrpc.com/
- http://ws.apache.org/xmlrpc/
-

SOAP

- PHP based implementation of the SOAP protocoll
- Allows the creation of clients and servers
- Supports WSDL (auto proxy generation)
- Allows http-proxy usage (caching)
- Outdated, PHP5's ext/SOAP is the current standards

SOAP

- Example (client without WSDL):

```
class MyClient {
    var $_client;    var $_nameSpace;
    function MyClient ($url,$nameSpace) {
        $this->_client = new SOAP_Client($url);
        $this->_nameSpace= $nameSpace;
    }
    function hello($name) {
        $params=array($name);
       return $this->client->call('hello', $params,
                                         $this->nameSpace);
} }
$myClient = new MyClient($url,$namespace);
```

SOAP

- Example (client with WSDL):

```
$wsdl=new SOAP_WSDL('http://localhost/myserver.wsdl');
$myClient=$wsdl->getProxy();
```

SOAP

- Example:

```
$url='http://localhost/soap_client_2.php';
$namespace='http://localhost/#Test1';

echo ( $myClient->hello('World!').'\n' );
```

SOAP

- Example:

```
class MyServer {
    var $_server;
    function Test1 () {
        $this->_server = new SOAP_Server;
$this->soapServer->addObjectMa($this,'http://localhost#Test1');
$this->soapServer->service($GLOBALS['HTTP_RAW_POST_DATA']);
    }
    function hello($name) { return 'Hello '.$name; }
}

$myServer = new MyServer();
```

Usefull links

- http://pear.php.net/package/SOAP
- http://www.php.net/SOAP
- http://www.w3.org/TR/soap/
- http://ws.apache.org/soap/

Services_Google

- Query Google from through PHP (including similarity hints,...)
- Wraps around the Google SOAP interface
- Google API key needed (free dev account)

Services_Google

- Example:

```
$google = new Services_Google("KEY HERE");
$google->queryOptions['limit'] = 100;
$google->search("Tobias Schlitt");

foreach($google as $key => $result) {
    echo $key."\t".$result->title."\n";
}
```

Usefull links

- http://pear.php.net/package/Services_Google
- http://www.google.com/apis/

Services_Delicious

- Communicate with the del.icio.us webervices API
- Delicous == socal bookmarking
- Based on REST

Services_Delicious

- Example:

```
$dlc = &new Services_Delicious($username, $password);


var_dump($dlc->getRecentPosts());
var_dump($dlc->getTags());
var_dump(
    $dlc->addPost(
        'http://pear.php.net',
        'PEAR',
        'The PHP Extension and Application Repository',
        'php')
);
```

Usefull links

- http://pear.php.net/package/Services_Delicious
- http://del.icio.us

Services_Yahoo

- Query Yahoo! from PHP
- OO model for the Yahoo! websearch API
- Until now, no further services available by Yahoo!
- Based on REST
- PHP5 only
- Uses PHP5 iterators

Services_Yahoo

- Example:

```
try {
    $search = Services_Yahoo_Search::factory("web");
    $search->setQuery("Stephan Schmidt");
    $results = $search->submit();
    echo "Total: " . $results->getTotalResultsReturned() . "\n\n";
    foreach ($results as $result) {
        echo $result['Title'] . "\n";
    }
} catch (Services_Yahoo_Exception $e) {
    die('Query failed');
}
```

try `$> pear install Services_Yahoo`

Usefull links

- http://pear.php.net/package/Services_Yahoo
- http://developer.yahoo.net/

Services_Ebay

- Wraps around Ebay's non-standard webservice (not the SOAP one).
- Very powerfull services, supports 70 API calls (everything you can do on eBay, except bidding)
- 50 API calls available in Services_Ebay, yet.
- PHP5 only

try $> `pear install Services_Ebay`

Services_Ebay

- Example

```
$session = Services_Ebay::getSession($devId, $appId, $certId);
$session->setToken($token);
$ebay = new Services_Ebay($session);
$item = $ebay->GetItem(4501333179, 2);
print_r($item->toArray());
$item->Title = 'The new title of the item';
$ebay->ReviseItem($item);
```

Usefull links

- http://pear.php.net/package/Services_Ebay
- http://pear.php.net/manual/en/package.webserv
- http://sandbox.ebay.com/
- http://developer.ebay.com/

Services_Trackback

- Generic class for trackbacks
- Allows sending and receiving trackbacks
- Supports autodiscovery of trackbacks and generation of autodiscovery code
- Integrated spam protection

try `$> pear install Services_Trackback`

Services_Trackback

- Example:

```
$trackback = Services_Trackback::create(array(
        'id' => 'Test',
        'url' => 'http://pear.php.net/package/Net_FTP'));
var_dump($trackback->autodiscover());

$trackback->set('title', 'Testing Services_Trackback');
$trackback->set('url', 'http://www.example.com');
$trackback->set('excerpt', 'Foo bar baz...');
$trackback->set('blog_name', 'Testing Services_Trackback');

var_dump($trackback->send());
```

Usefull links

- http://pear.php.net/package/Services_Trackback
- http://www.movabletype.org/trackback/
- http://www.movabletype.org/trackback/beginners/

The end

Thank you for your attention!

Are there

- questions?
- suggestions?
- critics?