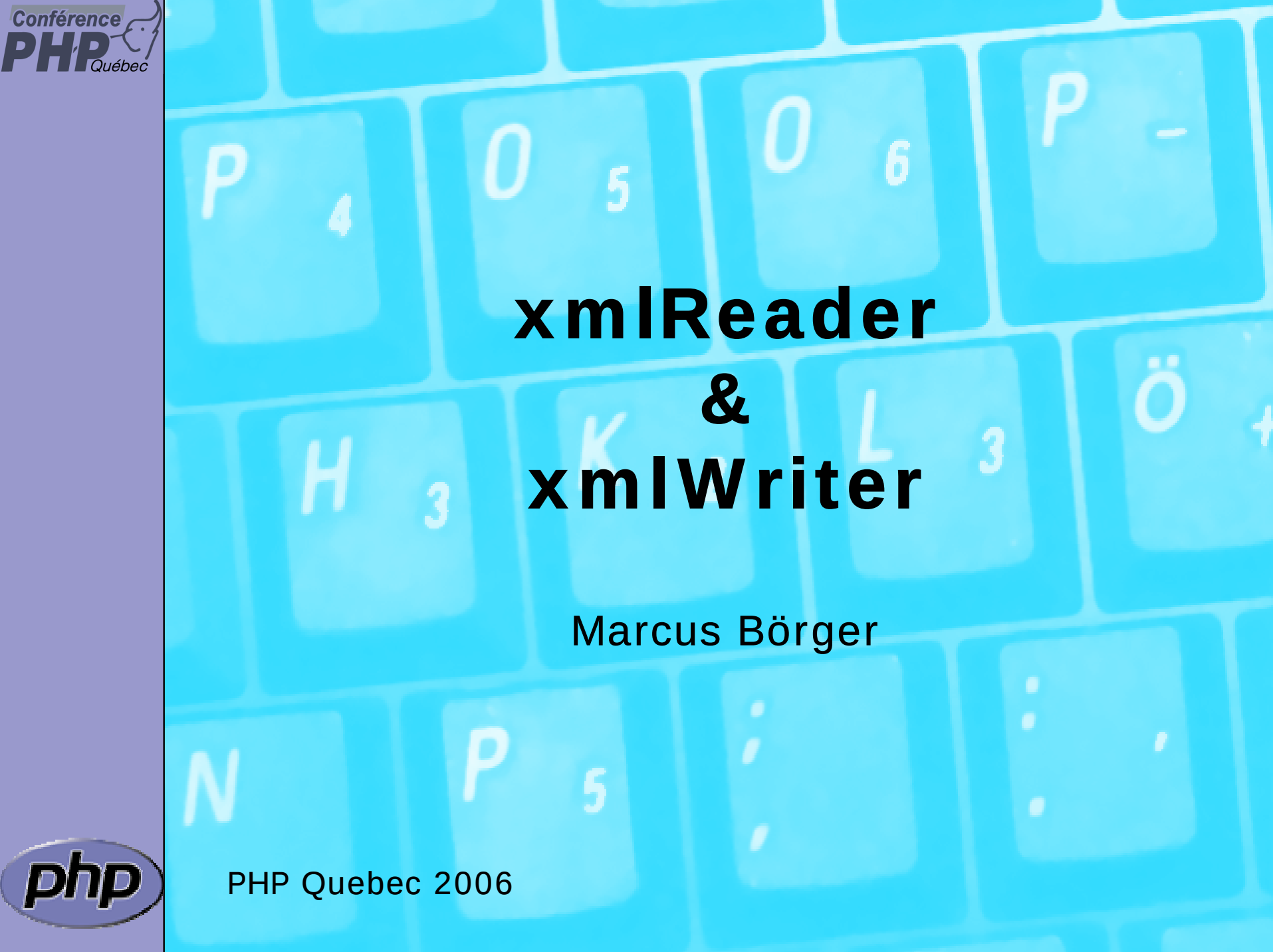


xmlReader & xmlWriter

Marcus Börger

PHP Quebec 2006

xmlReader & xmlWriter

- ☑ Brief review of SimpleXML/DOM/SAX
- ☑ Introduction of xmlReader
- ☑ Introduction of xmlWriter

DOM

- ☑ Full W3C compatible DOM support
- ☑ Fast XPath support
- ☑ Validation support
- ☑ Fast/direct access to any piece of you XML data
- ☑ No problems with namespaces
- ☑ Good PHP mapping

- ☒ Needs to build full DOM tree before you can use it
- ☒ Memory intensive

SimpleXMLElement

- ☑ Natural object relation from xml to php
  - ☑ Object value — Content
  - ☑ Properties — Elements
  - ☑ ArrayAccess — Attributes
- ☑ XPath support
- ☑ Can easily switch from DOM to SimpleXML
- ☑ Iterator based

- ☒ Problems with handling namespaces
- ☒ Builds full dom tree prior to map it to php objects
- ☒ No support for validation

SAX

- ☑ Fast event based parsing
- ☑ No overhead whatsoever

- ☒ Programmer has to do everything himself
- ☒ No XPath support
- ☒ No validation
- ☒ Push parser tells you exactly how to parse data

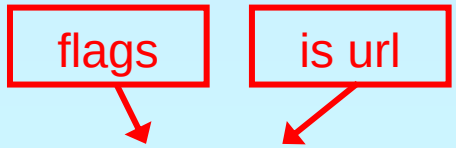
xmlReader

- ☑ Fast and flexible event based parsing
- ☑ Pull parser operates like you use it
- ☑ Validation support (DTD, XSD, RNG)
- ☑ Can load defaults from definition (DTD)
- ☑ Direct access to all attriutes of an element
- ☑ C# XmlTextReader API
- ☑ Allows to generate DOM tree from current element

- ☑ No XPath support
- ☑ XSD Support limited in libxml2

SimpleXMLIterator

- ☑ SPL makes SimpleXML recursion aware
  - ☑ Use `simplexml_load_(file|string)` with 2nd param
  - ☑ Or `SimpleXmlIterator` direct by constructor




```php
<?php

$xml = new SimpleXmlIterator($argv[1], 0, true);

foreach(new RecursiveIteratorIterator($xml) as $e)
{
   if (isset($e['href']))
   {
      echo $e['href'] . "\n";
   }
}

?>
```

Strip href with xmlReader

- ☑ Create a reader and read everything

```
$reader = new XMLReader();
if ($reader->open($argv[1])) {
   while ($reader->read()) {
      if ($reader->nodeType == XMLReader::ELEMENT &&
         $reader->hasAttributes)
      {
         $href = $reader->getAttribute('href');
         if (isset($href))
         {
            echo $href . "\n";
         }
      }
   }
}
$reader->close();
```

Strip href with xmlReader

- ☑ Create a reader and read everything
- ☑ Check for attributes on all elements

```
$reader = new XMLReader();
if ($reader->open($argv[1])) {
  while ($reader->read()) {
     if ($reader->nodeType == XMLReaader::ELEMENT &&
        $reader->hasAttributes)
     {
        $href = $reader->getAttribute('href');
        if (isset($href))
        {
           echo $href . "\n";
        }
     }
  }
}
$reader->close();
```

Strip href with xmlReader

- ☑ Create a reader and read everything
- ☑ Check for attributes on all elements
- ☑ Check for the specific attribute we're interested in

```
$reader = new XMLReader();
if ($reader->open($argv[1])) {
  while ($reader->read()) {
     if ($reader->nodeType == XMLReader::ELEMENT &&
        $reader->hasAttributes)
     {
        $href = $reader->getAttribute('href');
        if (isset($href))
        {
           echo $href . "\n";
        }
     }
  }
}
$reader->close();
```

Up to 5.1.2 xmlReader returns an empty string for non existing attributes

ArrayAccess

- ☑ You may overload xmlReader

```
class MyXMLReader extends XMLReader
   implements ArrayAccess
{
   function offsetSet($ofs, $value) {
      throw new Exception('Cannot set attributes');
   }

   function offsetUnset($ofs) {
      throw new Exception('Cannot unset attributes');
   }

   // ...
```

ArrayAccess

- ☑ Testing whethe an attribute exists

```
function offsetExists($ofs) {
   $result = false;
   if ($this->hasAttributes
   || $this->nodeType == self::ATTRIBUTE) {
      $n = $this->nodeType == self::ATTRIBUTE
         ? $tihs->name : NULL;
      for ($p = $this->attributeCount; $p; ) {
         $this->moveToAttributeNo(--$p);
         if ($this->name == $ofs) {
            $result = true;
         }
      }
      if (isset($n)) {
         $this->moveToAttribute($n);
      } else {
         $this->moveToElement();
      }
   }
   return $result;
}
```

ArrayAccess

- ☑ Reading an attribut by name

```
   function offsetGet($ofs) {
      $result = NULL;
      if ($this->hasAttributes
      || $this->nodeType == self::ATTRIBUTE) {
         $n = $this->nodeType == self::ATTRIBUTE
            ? $tihs->name : NULL;
         for ($p = $this->attributeCount; $p; ) {
            $this->moveToAttributeNo(--$p);
            if ($this->name == $ofs) {
               $result = $this->value;
            }
         }
         if (isset($n)) {
            $this->moveToAttribute($n);
         } else {
            $this->moveToElement();
         }
      }
      return $result;
   }
} // MyXMLReader
```

Strip href with xmlReader

- ☑ Change to use the overloaded class

```
$reader = new MyXMLReader();
if ($reader->open($argv[1])) {
  while ($reader->read()) {
     if ($reader->nodeType == XMLReader::ELEMENT &&
        $reader->hasAttributes)
     {
        $href = $reader->getAttribute('href');
        if (isset($href))
        {
           echo $href . "\n";
        }
     }
  }
}
$reader->close();
```

Strip href with xmlReader

- ☑ Change to use overloaded class

```
$reader = new MyXMLReader();
if ($reader->open($argv[1])) {
  while ($reader->read()) {
    if ($reader->nodeType == XMLReader::ELEMENT &&
      $reader->hasAttributes)
    {
      if (isset($reader['href']))
      {
        echo $reader['href'] . "\n";
      }
    }
  }
}
$reader->close();
```

What can be read

- ☑ `read()` method and `nodeType` property
  - ☑ Elements — `ELEMENT`
  - ☑ Element closing — `END_ELEMENT`
  - ☑ Processing instruction — `PI`
  - ☑ Comment — `COMMENT`
  - ☑ Text/Content — `TEXT`
  - ☑ CDATA — `CDATA`
  - ☑ Entity — `ENTITY`
  - ☑ End entity — `END_ENTITY`
  - ☑ Whitespace — `SIGNIFICANT_WHITESPACE`
  - ☑ Attribute — `ATTRIBUTE`
  - ☑ Nothing as in end of data — `NONE = 0`

Parser configuration

- ☑ You can control how parsing operates
  - ☑ Loading a DTD — `LOADDTD`
  - ☑ Using default attribute values — `DEFAULTATTRS`
  - ☑ Validating against a DTD — `VALIDATE`
  - ☑ Whether entities are substituted — `SUBST_ENTITIES`

```
$reader = new XMLReader();
$reader->open($file);
$reader->setParserProperty(XMLReader::LOADDTD, TRUE);
$reader->setParserProperty(XMLReader::VALIDATE, TRUE);
```

- ☑ You can verify parsing operation

```
$reader->getParserProperty(XMLReader::LOADDTD);
```

RelaxNG validation

- ☑ Before reading data you can validate against RNG

```
$reader = new XMLReader();
$reader->open($file);
if ($reader->setRelaxNGSchema($relaxngfile)) {
  while ($reader->read());
}
if ($reader->isValid()) {
   print "File is ok\n";
} else {
   print "File could not be validated:\n";
   print libxml_error_get_errors();
}
$reader->close();
```

Helpful properties

- Some helping readonly properties
  - Node type `$r->nodeType`
  - Name of the node `$r->name`
  - Local name `$r->localName`
  - Prefix `$r->prefix`
  - Namespace URI `$r->namespaceURI`
  - Base URI `$r->baseURI`
  - Whether element is empty `$r->isEmptyElement`
  - Value of text node `$r->value`
  - Does element have attributes `$r->hasAttributes`
  - Number of attributes `$r->attributeCount`
  - Is attribute value the default `$r->isDefault`
  - Depth of element `$r->depth`

Basic functions

- ☑ Is the reader in a valid state `$r->isValid()`
- ☑ Move forward to next node `$r->next()`
- ☑ Move from attribute to element `$r->moveToElement()`
- ☑ Expand current node to DOM `$r->expand()`

The following both read up to the next node named 'book':

```
while($reader->isValid() && $reader->name != 'book') {
   $reader->next();
}
```

```
while($reader->read() && $reader->name != 'book') ;
```

Attribute functions

- ☑ Attribute traversal
  - ☑ moveToFirstAttribute()
  - ☑ moveToNextAttribute()
  - ☑ moveToAttribute(string name)
  - ☑ moveToAttributeNo(int index)
  - ☑ moveToAttributeNs(string name, string namespaceURI)
- ☑ Attribute access
  - ☑ getAttribute(string name)
  - ☑ getAttributeNo(int index)
  - ☑ getAttributeNs(string name, string namespaceURI)

Some XML data

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
<book title='Eragon (Inheritance, Book 1)'
      date='August 26, 2003'
      publisher='1'
      pages='544'>
   <author id='1'/>
</book>
<book title='Eldest (Inheritance, Book 2)'
      date='August 23, 2005'
      publisher='1'
      pages='704'>
   <author id='1'/>
</book>
<author id='1' name='Christopher Paolini'/>
<publisher id='1' name='Knopf Books for young readers'/>
</books>
```

Simply accessing all data

- ☑ Using SimpleXML any data is directly accessible

```
<html>
<head><title>Books</title></head>
<body>
<dl>
<?php
$x = simplexml_load_file($_GET['xml']);
foreach($x->book as $book) {
   echo "<dt>" . $book['title'] . "</dt>\n";
   $id = $book->author['id'];
   $a = $x->xpath('/books/author[@id="'.$id.'"]/text()');
   echo "<dd>Author: " . $a[0] . "</dd>\n";
}
?>
</dl>
</body>
</html>
```

Some other XML data

- ☑ Using a DTD/Layout that suits a streaming parser

```
<?xml version="1.0" encoding="UTF-8"?>
<books>
<author id='1' name='Christopher Paolini'/>
<publisher id='1' name='Knopf Books for young readers'/>
<book date='August 26, 2003'
      publisher='1'
      pages='544'
      author id='1'>Eragon (Inheritance, Book 1)
</book>
<book date='August 23, 2005'
      publisher='1'
      pages='704'>
      author id='1'>Eldest (Inheritance, Book 2)
</book>
</books>
```

Reading xml data

- ☑ Provide the page structure, create & open a reader

```
<html>
<head><title>Books</title></head>
<body>
<dl><?php
$author = array(); $publisher = array();
$reader = new XmlReader();
$reader->open($argv[1]);
while($reader->read()) {
   if ($reader->nodeType == XMLReader::ELEMENT) {
      switch($reader->name) {
      case 'author':  read_author($reader); break;
      case 'book':    read_book($reader); break;
      }
   }
}
?></dl>
</body>
</html>
```

Reading xml data

- ☑ Read until end of xml data

```
<html>
<head><title>Books</title></head>
<body>
<dl><?php
$author = array(); $publisher = array();
$reader = new XmlReader();
$reader->open($argv[1]);
while($reader->read()) {
   if ($reader->nodeType == XMLReader::ELEMENT) {
      switch($reader->name) {
      case 'author':  read_author($reader); break;
      case 'book':    read_book($reader); break;
      }
   }
}
?></dl>
</body>
</html>
```

Reading xml data

- ☑ For each element of interest use dedicated handler

```
<html>
<head><title>Books</title></head>
<body>
<dl><?php
$author = array(); $publisher = array();
$reader = new XmlReader();
$reader->open($argv[1]);
while($reader->read()) {
   if ($reader->nodeType == XMLReader::ELEMENT) {
      switch($reader->name) {
      case 'author':  read_author($reader); break;
      case 'book':    read_book($reader); break;
      }
   }
}
?></dl>
</body>
</html>
```

Reading xml data

- Store author information in a global array
  - If the element has some content (it is not empty)
  - Use text node as author info
  - Before using the text node read the id attribute

```
function read_author($reader)
{
   global $author;

   if (!$reader->isEmptyElement) {
      $id = $reader->getAttribute('id');
      $reader->read();
      $author[$id] = $reader->value;
   }
}
```

Reading xml data

- ☑ For all books handle its attributes and sub nodes
  - ☑ Lookup the author in the global array
  - ☑ Access all text nodes

```
function read_book($reader)
{
   global $author;

   $id = $reader->getAttribute('author');
   echo "<dt>" . get_text($reader) . "</dt>\n";
   echo "<dd>Author: " . $author[$id] . "</dd>\n";
}
```

Reading xml data

- ☑ Reading only the text nodes, concatenating them
  - ☑ Store the current depth
  - ☑ Read until end of element at stored depth
  - ☑ If node is a text node append its value

```
function get_text($reader)
{
   $t = '';
   $l = $reader->depth;
   while($reader->read() && ($reader->depth > $l
   || $reader->nodeType != XMLReader::END_ELEMENT)) {
      if ($reader->nodeType == XMLReader::TEXT) {
         $t .= $reader->value;
      }
   }
   return trim($t);
}
```

xmlWriter

- ☑ xmlWriter is used for easy creation of XML data
  - ☑ Automatically cares for escaping
  - ☑ Can directly write to a stream or memory
  - ☑ Allows to control indendation
  - ☑ Checks validity and ends any open tag on close

xmlWriter

- ☑ Providing some data

```
$author = array(1 => 'Christopher Paolini');
$publisher = array(
   1=>array('name'=>'Knopf Books for young readers'));
$books = array(
   array('date'=>'August 26, 2003',
      'publisher'=>'1',
      'pages'=>'544',
      'author'=>'1',
      'title'=>'Eragon (Inheritance, Book 1)'),
   array('date'=>'August 23, 2005',
      'publisher'=>'1',
      'pages'=>'704',
      'author'=>'1',
      'title'=>'Eldest (Inheritance, Book 2)'),
   );
```

Initial steps

- ☑ Creating, Opening, Indent control, Document start

```php
$writer = new XMLWriter();
//$w->openURI($filename);
$writer->openMemory();
$writer->setIndent(true);
$writer->setIndentString('  ');
$writer->startDocument('1.0', 'UTF-8');
```

- ☑ Creating the root element

```php
$writer->startElement('books');
```

Writing data

- ☑ Creating an element
- ☑ Adding attributes
- ☑ Closing the element

```
foreach($publisher as $id => $name)
{
   $writer->startElement('publisher');
   $writer->writeAttribute('id', $id);
   $writer->writeAttribute('name', $name);
   $writer->endElement();
}
```

Writing some data

- ☑ Create the root element
- ☑ Create more elements
  - ☑ Add attributes
  - ☑ Add content

```
foreach($author as $id => $name) {
   $writer->startElement('author');
   $writer->writeAttribute('id', $id);
   $writer->text($name);
   $writer->endElement();
}
```

Writing more data

- ☑ Writing more data

```
foreach($books as $book)
{
   $writer->startElement('book');
   foreach($book as $attr => $val)
   {
      if ($attr != 'title') {
         $writer->writeAttribute($attr, $val);
      }
   }
   $writer->text($book['title']);
   $writer->endElement();
}
```

Closing down

- ☑ Closing the document and writing the xml file

```
$writer->endDocument();

echo $writer->outputMemory();

// $writer->flush();
```

THANK YOU

- ☑ This Presentation
  http://somabo.de/talks/
- ☑ PHP Manual
  http://php.net/xmlreader
- ☑ Libxml2
  http://xmlsoft.org