Web Services in PHP

By: Ilia Alshanetsky

The REST Way

- Representational State Transfer (REST)
  - Concept coined by Roy Fielding
- Uses the web or to be precise HTTP or HTTPS exclusive for transmitting web service request and response.
- Classic setup:
  - Input via GET/POST
  - Output as XML document

REST Request

`http://site.com/forum/rss.php?latest=1`

That's all folks

Want to make a remote request?

No problem!

```
$url = "...";
$response = file_get_contents($url);
```

REST Response

```
<?xml version="1.0"?>
<forum uri="http://fudforum.org/index.php">

  <item id="1">
    <title>First Post!!!</title>
    <link>http://fudforum.org/index.php/m/1</link>
    <description>1st message in the forum.</description>
  </item>

  <item id="2">
    <title>Re: First Post!!!</title>
    <link>http://fudforum.org/index.php/m/1</link>
    <description>Almost like Slashdot.</description>
  </item>

</forum>
```

Parsing XML Response

- To parse the returned XML we turn to any number of extensions found in PHP.
  - XML extension
    - Basic XML parser based on SAX methodology found in all PHP versions.
  - SimpleXML
    - Arguably the simplest XML parser to use.
  - DOM
    - Maximum flexibility for both parsing and creating XML
  - XMLReader
    - Pull parser, that combines ease of use with high performance.

XML Parsing Methodologies

SAX

(Simple API for XML)

An event based approach where by each action, such "found new tag" needs to be handled. The triggerable events include:

- open tag
- close tag
- tag's data

DOM

(Document Object Model)

Loads the entire document into memory, creating a "tree" representing the XML data. The tree can then be traversed in multitude of ways.

Simple API for XML

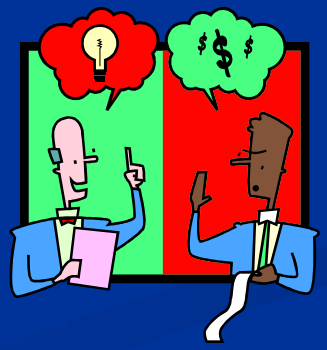
- Uses little memory.
- Allows work to starts immediately.
- Works well with remote XML data source.
- Same parsing API in PHP 4 and 5

- All those handler calls are slow.
- Only Sequential data access.
- Can't easily retrieve a particular document segment.
- Requires lot's of PHP code.
- Read-only.

Document Object Model

- Very fast for small documents.
- Access anything anytime.
- Simpler PHP interface.
- Underlying XML parsing library, libXML2 is better suited for DOM.

- "All your memory are belong to DOM".
- Data only usable after the complete document is retrieved parsed.
- You'll need PHP 5+.



PHP's XML Parsers

SAX

(Simple API for XML)

- XML
- XMLReader (PECL)
- XMLWriter (PECL)

DOM

(Document Object Model)

- SimpleXML
- DOM (PHP5)
- DOMXML (PHP 4)
- XSLT (dom engine)
- SOAP (dom engine)
- XML-RPC (dom engine)

Biggest Gripe About Sax

- One of the biggest complaints about SAX is that it PHP requires the developer to write a lot of code and be fully aware of the XML being parsed.

It can't be that bad, can it?

```
class xmlParser {
  private $x, $file, $cur_tag, $cur_id;
  public $data_store = array(), $n_entries = 0;
  function __construct($xml_file) {
    $this->file = $xml_file;
    $this->x = xml_parser_create();

    xml_set_object($this->x, $this);
    xml_set_element_handler($this->x, "startTag", "endTag");
    xml_set_character_data_handler($this->x, 'tagContent');
  }
  function parse() {
    $fp = fopen($this->file, "r");
    while (!feof($fp))
      xml_parse($this->x, fread($fp, 1024), feof($fp));
    fclose($fp);
  }
  function __destruct() { xml_parser_free($this->x); }
```

Callbacks

```
function startTag($parser, $tag_name, $attribute) {
  if ($tag_name == 'ITEM') {
    $this->cur_id = (int)$attribute['ID'];
    $this->data_store[$this->cur_id] = array();
  } else if ($tag_name != 'FORUM')
     $this->data_store[$this->cur_id][$tag_name] = '';

  $this->cur_tag = $tag_name;
}
function endTag($parser, $tag_name) {
  if ($tag_name == 'ITEM') ++$this->n_entries;
}

function tagContent($parser, $data) {
  if (in_array($this->cur_tag,                array
   ('TITLE','LINK','DESCRIPTION')))
    $this->data_store[$this->cur_id][$this->cur_tag] .= $data;
}
```

Case Sensitivity

- One of things XML shares with HTML is the inherent case-insensitivity of the tags.

- The XML extensions automatically "solves" this problem for you by case-folding all tags to uppercase.
  - To disable this functionality use:

```
xml_parser_set_option($parser, XML_OPTION_CASE_FOLDING, 0);
```

Why have parser in handlers?

- The parser reference allows retrieval additional information about the XML content.
  - **`xml_get_current_line_number(resource parser)`**
    - The current line number in the XML file.
  - **`xml_get_current_byte_index(resource parser)`**
    - The current file position (starts at 0).
  - **`xml_get_current_column_number(resource parser)`**
    - The position in the current column (starts at 0).

What About Errors?

- In the event of an error `xml_parse()` will return 0.

```
function parse() {
  $fp = fopen($this->file, "r");
  while (!feof($fp)) {
    if (!xml_parse($this->x, fread($fp, 1024), feof($fp))) {
      printf("Error #%d: '%s' on %s:%d\n",
                ($c = xml_get_error_code($this->x)),
                xml_error_string($c),
                $this->file,
                       xml_get_current_line_number($this->x));
    }
  }
  fclose($fp);
}
```

Putting it all together

```
$a = new xmlParser("xml.xml");
$a->parse();
echo "Found {$a->n_entries} Messages\n";
foreach ($a->data_store as $id => $v) {
        echo "{$id}) {$v['TITLE']}\n";
}
```

Output:

```
Messages Found: 2
1) First Post!!!

2) Re: First Post!!!
```

SimpleXML to the Rescue!

Thanks to the `SimpleXML` extension, it can.

```
<?php
foreach (simplexml_load_file("xml.xml") as $v) {
        echo "{$v['id']}) '{$v->title}'\n";
}
?>
```

By making use of the new PHP 5 OO features such as `__toString()` and object overloading it makes parsing XML so very simple.

DOM Extension

- DOM extension is a complete rewrite of the DOMXML extension that was available in PHP 4. The core functionality includes:
  - Read/Write/Create functionality.
  - XPath queries.
  - Several document validation mechanisms.
  - Namespace Support
  - HTML parsing capabilities.

Basic Usage

- Reading data from XML document via DOM is conceptually not much different from SimpleXML.

```
<?php
$dom = new domDocument();
$dom->load("xml.xml");

foreach ($dom->getElementsByTagName('title') as $k => $node) {
        $id = $dom->getElementsByTagName('item')->
                        item($k)->getAttribute('id');
         echo "{$id}) {$node->nodeValue}\n";
}
?>
```

Creating XML

```
$dom = new domDocument("1.0","ISO-8859-1");
$dom->formatOutput = 1;
$root = $dom->createElement('books');

$branch = $dom->createElement('book');

$branch->setAttribute('ISBN', '0973862106');

$leaf = $dom->createElement('title');
$leaf->appendChild(
  $dom->createTextNode(`PHP Guide to
  Security'));
$branch->appendChild($leaf);
```

Creating XML Step 2

```
$leaf = $dom->createElement('price');
$leaf->appendChild(
  $dom->createTextNode('32.99'));
$branch->appendChild($leaf);

$leaf = $dom->createElement('url');
$leaf->appendChild(
  $dom->createCDATASection(‘amazon.com/…’);
$branch->appendChild($leaf);
```

Creating XML Step 3

```
$root->appendChild($branch);
$dom->appendChild($root);
echo $dom->saveXML();
```

What do we have?

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<books>
  <book ISBN="0596007647">
    <title>PHP Guide to Security</title>
    <price>26.37</price>
    <url><![CDATA[amazon.com/...]]></url>
  </book>
</books>
```

Modifying Existing Documents

```
$dom = new domDocument();
$dom->load("xml.xml");

$item = $dom->createElement('item');
$item->setAttribute('id', '1.5');
foreach (array('title', 'link','description') as $v) {
  $leaf = $dom->createElement($v);
  $leaf->appendChild($dom->createTextNode($v));
  $item->appendChild($leaf);
}

$inl = $dom->getElementsByTagName('forum')->item(0);
$ref_node = $dom->getElementsByTagName('item')->item(1);
$inl->insertBefore($item, $ref_node);
$dom->save("new_xml.xml");
```

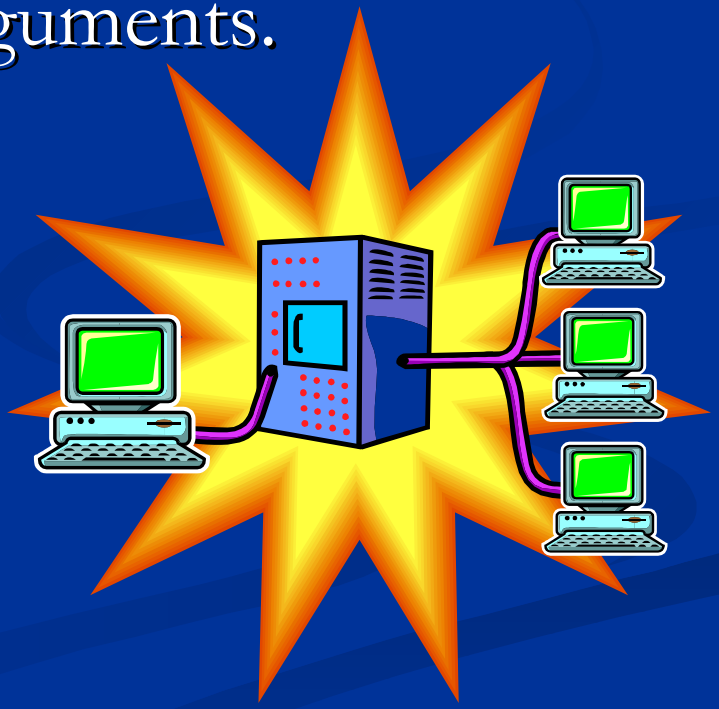
Generated XML

```
<?xml version="1.0"?>
<forum uri="http://fudforum.org/index.php">

  <item id="1">
    <title>First Post!!!</title>
    <link>http://fudforum.org/index.php/m/1</link>
    <description>1st message in the forum.</description>
  </item>

  <item
   id="1.5"><title>title</title><link>link</link><description>desc
   ription</description></item>
  <item id="2">
    <title>Re: First Post!!!</title>
    <link>http://fudforum.org/index.php/m/2</link>
    <description>Almost like Slashdot.</description>
  </item>

</forum>
```

XML-RPC

- XML-RPC allows a computer to execute functions and class methods on another computer with any given arguments.

- Uses HTTP for transporting requests that are encoded using XML.



XMLRPC Request

```
<?xml version="1.0"?>
<methodCall>
    <methodName>package.info</methodName>
    <params>
        <param>
            <value>
                <string>XML_RPC</string>
            </value>
        </param>
    </params>
</methodCall>
```

XMLRPC Response

```
<?xml version="1.0" encoding="iso-8859-1"?>
<methodResponse>
<params>
<param>
  <value>
   <struct>
    <member>
     <name>packageid</name>
     <value>  <string>17</string> </value>
    </member>
    <member>
     <name>name</name>
     <value> <string>XML_RPC</string> </value>
    </member>
   </struct>
  </value>
</param>
</params>
</methodResponse>
```

XMLRPC Server

```
<?php
function cur_date() {
        return date("r");
}
// create a new XMLRPC server instance
$xs = xmlrpc_server_create();
// expose local cur_date function as date method
xmlrpc_server_register_method($xs, "date",
   "cur_date");
// listen for requests coming via POST
echo xmlrpc_server_call_method($xs,
   $HTTP_RAW_POST_DATA, NULL);
// free server instance
xmlrpc_server_destroy($xs);
?>
```

Response Formatting Options

- **verbosity:** determine compactness of generated xml. options are `no_white_space`, `newlines_only`, and `pretty`.
  - default = pretty
- **escaping:** determine how/whether to escape certain characters. 1 or more values are allowed. If multiple, they need to be specified as a sub-array. options are: `cdata`, `non-ascii`, `non-print`, and `markup`.
  - default = non-ascii, non-print, markup
- **version:** version of xml vocabulary to use. currently, three are supported: xmlrpc, soap 1.1, and simple.
  - default = xmlrpc
- **encoding:** the encoding that the data is in.
  - default = iso-8859-1

XMLRPC Client

```
$req = xmlrpc_encode_request("date", array());

$opts = array(
        'http'=>array(
                'method' => "POST",
                'content' => $req
        )
);

$context = stream_context_create($opts);
$ctx = @file_get_contents(
            "http://localhost/xmlrpc_server.php",
            NULL,
            $context);
echo xmlrpc_decode($ctx);
```

Pros & Cons

- Easy to understand, implement and debug.
- Quite fast.
- Stable
- Can be emulated with PEAR

- No new functionality being added.
- Not completely buzzword compliant.
- Very few big providers use XMLRPC.

SOAP

- Formerly known as Simple Object Access Protocol.
- A messaging format primary used for RPC.
- Uses XML.
- View Source protocol.
- Developed jointly by Microsoft, IBM and W3C.

SOAP Rules

- Each SOAP document, be it a request or response must follow a set of formatting rules:
  - Must have a top-level Envelope namespaced to http://schemas.xmlsoap.org/soap/envelope/
  - Must contain Body element.
  - May contain an optional Header and Fault elements.
  - The contents of Header and Body must be properly namespaced.

SOAP Document

```
<?xml version="1.0"?>
<soap:Envelope
xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

<soap:Header>
  <m:Trans xmlns:m="http://www.example.com/header/"
                                soap:mustUnderstand="1">
   234
  </m:Trans>
</soap:Header>

<soap:Body>
    <soap:Fault></soap:Fault>
    <purge xmlns="http://fud.prohost.org/message">
          <documentid>298</documentid>
    </purge>
</soap:Body>
</soap:Envelope>
```

SOAP Faults

- Faults can be of four basic types:
  - **VersionMismatch:** invalid namespace for the SOAP Envelope
  - **MustUnderstand:** a required header was not understood by the server
  - **Client:** the message sent by the client was not properly formed, or did not contain the necessary information to fulfill the request.
  - **Server:** the message could not be processed
- Faults can also contain other information, such as a basic message describing the fault, the URI of the originator of the fault, and a detail field that can be used to provide any other data.

SOAP Fault

```
<?xml version="1.0"?>
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"
  xmlns:m="http://www.example.org/timeouts"
  xmlns:xml="http://www.w3.org/XML/1998/namespace">
<env:Body>
  <env:Fault>
    <env:Code>
      <env:Value>env:Sender</env:Value>
      <env:Subcode>
        <env:Value>m:MessageTimeout</env:Value>
      </env:Subcode>
    </env:Code>
    <env:Reason>
      <env:Text xml:lang="en">Sender Timeout</env:Text>
    </env:Reason>
    <env:Detail> <m:MaxTime>P5M</m:MaxTime> </env:Detail>
  </env:Fault>
</env:Body>
</env:Envelope
```

SOAP Client

```
<?php
// create a new SOAP client based on Google WSDL
$client = new SoapClient("./GoogleSearch.wsdl");
// retrieve content of ilia.ws from Google's Page cache
$google_cache = $client->doGetCachedPage($developer_id,
   "http://ilia.ws");
// display retrieved page
echo base64_decode($google_cache);
?>
```

Web Service Description Language

- WSDL is machine readable description (XML) of a web service.
  - The service provider uses WSDL to describe the methods offers, their parameters and the return values the client may expect.
  - The client parses the WSDL to determine the available methods, how to encode the parameters to them and how to deal with the returned data.

Captcha WSDL

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

<message name='captcha_in'>
</message>

<message name='captcha_out'>
  <part name='id' type='xsd:int'/>
</message>
```

Captcha WSDL

```
<message name='check_captcha_in'>
  <part name='text' type='xsd:string'/>
  <part name='id' type='xsd:int'/>
</message>

<message name='check_captcha_out'>
  <part name='value' type='xsd:boolean'/>
</message>
<portType name='CaptchaPortType'>
  <operation name='captcha'>
    <input message='tns:captcha_in'/>
    <output message='tns:captcha_out'/>
  </operation>
  <operation name='check_captcha'>
    <input message='tns:check_captcha_in'/>
    <output message='tns:check_captcha_out'/>
  </operation>
</portType>
```

Captcha WSDL

```
<binding name='CaptchaBinding' type='tns:CaptchaPortType'>
  <soap:binding style='rpc'
    transport='http://schemas.xmlsoap.org/soap/http'/>
  <operation name='captcha'>
    <soap:operation soapAction='urn:cap-value#captcha'/>
    <input>
      <soap:body use='encoded' namespace='urn:cap-value'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
    </input>
    <output>
      <soap:body use='encoded' namespace='urn:cap-value'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
    </output>
  </operation>
  <operation name='check_captcha'>
    <soap:operation soapAction='urn:cap-value#check_captcha'/>
    <input>
      <soap:body use='encoded' namespace='urn:capc-value'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
    </input>
    <output>
      <soap:body use='encoded' namespace='urn:capc-value'
        encodingStyle='http://schemas.xmlsoap.org/soap/encoding/'/>
    </output>
  </operation>
</binding>
```

Captcha WSDL

```
<service name='CaptchaGenerator'>
  <port name='CaptchaPort' binding='CaptchaBinding'>
    <soap:address location='http://localhost/captchaServer.php'/>
  </port>
</service>
</definitions>
```

- As you can see to provide a very simple web services, only offering two functions takes several pages of WDSL web service description.

Server Code

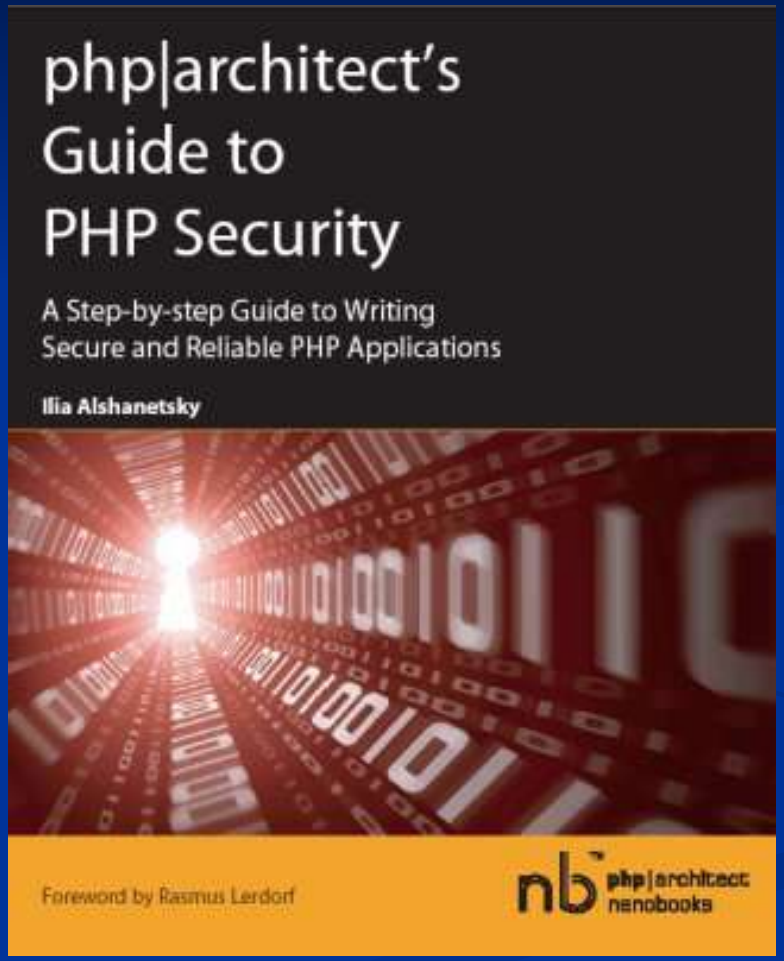
```
// instantiate SOAP server
$server = new SoapServer("./captcha.wsdl");
// Register exposed method
$server->addFunction('captcha'); // generate captcha
$server->addFunction('check_captcha'); // check captcha ID
$server->handle(); // listen of requests via POST
```

- As far as the PHP implementation goes, the needed code is extremely simple, in many ways making up for the complexities of WSDL.

Client Interface

```
<?php
$a = new SoapClient("./captcha.wsdl");
if (!empty($_POST)) {
   if ($a->check_captcha($_POST['text'], $_POST['id']))
        echo Validation Passed<br />';
   else
        echo 'Validation Failed<br />';
}
?>
<form method="post" action="<?php echo basename(__FILE__); ?>">
<img src="<?php echo ($id = $a->captcha()); ?>.png" />
<br />
<input type="hidden" name="id" value="<?php echo $id; ?>" />
The text is: <input type="text" name="text" value="" />
<input type="submit" />
</form>
```

<?php include “/book/plug.inc”; ?>



Questions