Designing & Implementing RESTful Web Services

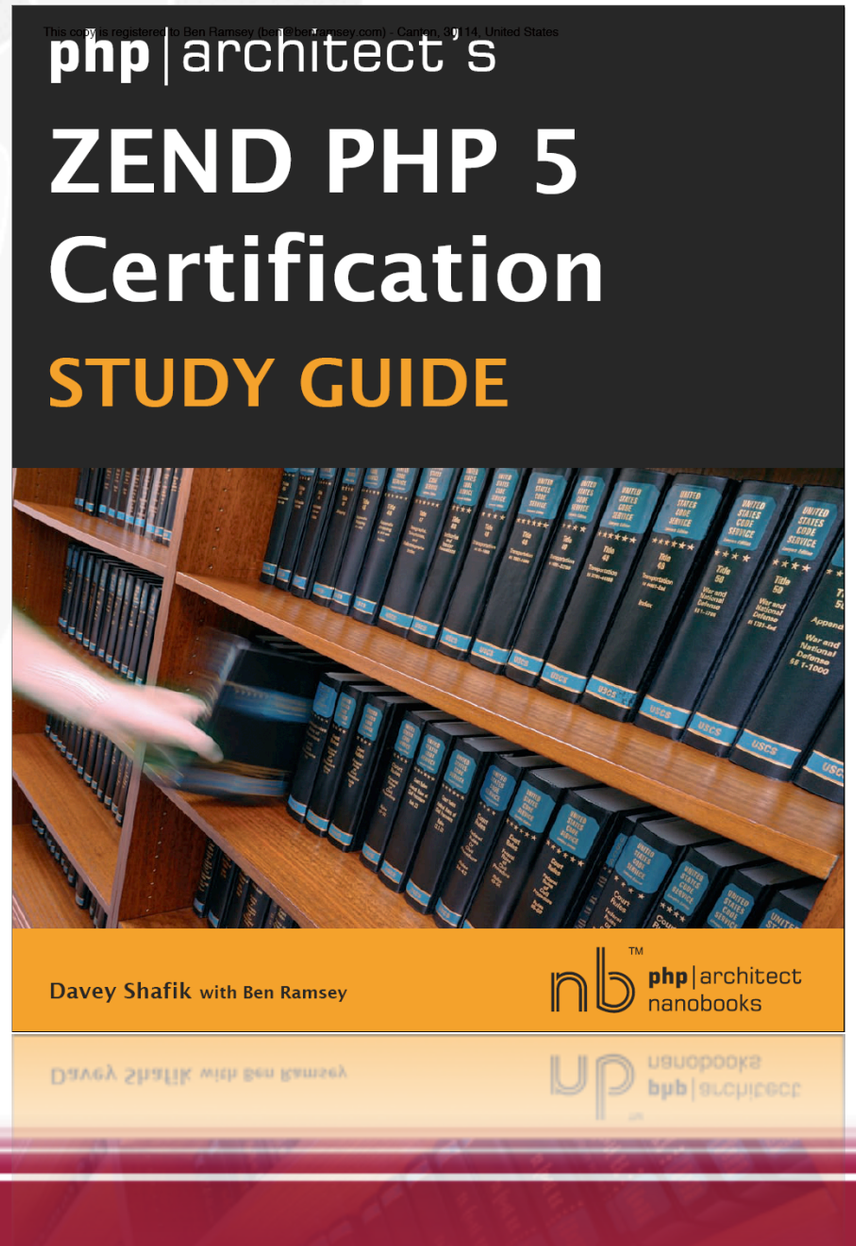
Ben Ramsey

International PHP Conference

7 November 2006

Welcome

- BenRamsey.com
- I work for Art & Logic, Inc.
- PHP 5 Certification Study Guide co-author
- Organizer of Atlanta PHP user group



Overview

- Web Services
- REST Overview
- Methods of Data Transport
- Example RESTful Web Services
- Creating RESTful Web Services

Web Services

What is a Web Service?

- Public interface (API)
- Provides access to data and/or procedures
- On a remote/external system (usually)
- Often uses XML for data exchange

Types of Web Services

- XML-RPC
- SOAP
- REST

REST Overview

What is REST?

- Representational State Transfer
- Term originated in 2000 in Roy Felding's doctoral dissertation about the Web entitled "Architectural Styles and the Design of Network-based Software Architectures"

Theory of REST

- Focus on diversity of resources (nouns), not actions (verbs)
- Every resource is uniquely addressable
- All resources share the same constrained interface for transfer of state (actions)
- Must be stateless, cacheable, and layered

What Does It Mean?

"[REST] is intended to evoke an image of how a well-designed Web application behaves: a network of web pages (a virtual state-machine), where the user progresses through an application by selecting links (state transitions), resulting in the next page (representing the next state of the application) being transferred to the user and rendered for their use." — Roy Felding

Web As Prime Example

- URIs uniquely address resources
- HTTP methods (GET, POST, HEAD, etc.) and content types provide a constrained interface
- All transactions are atomic
- HTTP provides cache control

Well-RESTed

- Applications adhering to REST principles are said to be RESTful
- Extreme advocates of REST are often called RESTafarians

Relaxing REST

- Any simple interface using XML over HTTP (in response to GET requests)
- That is also not RPC-based
- May use JSON, YAML, plain text, etc. instead of XML
- In most PHP applications, this is what we mean when we say “REST”

Methods of Data Transport

XML Over HTTP

- It's an extensible mark-up language
- This makes it very flexible
- Lightweight and easy to parse
- Ease of communication between disparate systems

Parsing XML With PHP

- SimpleXML or DOM

```php
<?php

$uri = 'http://example.org/service/rest/data';

$sxe = new SimpleXMLElement($uri, NULL, TRUE);

foreach ($sxe->node as $node)
{
    // do stuff, e.g.:
    // $node->element
    // $node['attribute']
}

?>
```

JSON Over HTTP

- JavaScript Object Notation
- Makes it easy to pass arrays and objects from PHP to JavaScript and vice versa
- Very useful and efficient in Ajax applications
- More lightweight than XML and easy to parse

Parsing JSON With PHP

- ext/json and Zend_JSON

```php
<?php

$json = <<<EOD
{"isbn":"014143984X","title":"Dracula","author":
{"surname":"Stoker","name":"Bram"},"publisher":
"Penguin Books"}
EOD;

$book = json_decode($json);

// do stuff, e.g.:
// $book->title;
// $book->author->surname;

?>
```

Which Method Is the Best?

- JSON is very lightweight but intended for JavaScript; useful for passing data to/from a front-end
- XML is very flexible and better for many other destinations (front-end, rich clients, other servers, etc.)
- The tools are available; the choice is yours

Example RESTful Web Services

del.icio.us

- Public and authenticated REST access
- All requests over SSL using HTTP-Auth
- Requests a 1-second delay between queries
- Very simple API
- http://del.icio.us/help/api/

delicious.php

```php
<?php

$un = 'your_username';
$pw = 'your_password';
$cache_file = '/tmp/blogroll.xml';

// check for updates to del.icio.us account
$update_req = "https://{$un}:{$pw}@api.del.icio.us/v1/posts/update";
$update = new SimpleXMLElement($update_req, NULL, TRUE);

if (strtotime($update['time']) > filemtime($cache_file))
{
    // del.icio.us has been updated since last cache; recache
    $uri = "https://{$un}:{$pw}@api.del.icio.us/v1/posts/all?tag=blogroll";
    $data = file_get_contents($uri);
    file_put_contents($cache_file, $data);
}

// read links from cached del.icio.us data
$blogroll = new SimpleXMLElement($cache_file, NULL, TRUE);

foreach ($blogroll->post as $blog)
{
    echo '<a href="' . htmlentities($blog['href']) . '">';
    echo htmlentities($blog['description'], ENT_COMPAT, 'UTF-8');
    echo "</a><br />\n";
}

?>
```

Yahoo!

- Web Search Service is RESTful
- Requires an application ID, but no special authentication or handshake
- Limit 5,000 queries per IP address per day
- http://developer.yahoo.com/search/web/V1/webSearch.html

yahoo.php

```php
<?php

$query = 'PHP';

$request  = 'http://search.yahooapis.com/';
$request .= 'WebSearchService/V1/webSearch?';
$request .= 'appid=ramsey&query=' . urlencode($query);

$ResultSet = new SimpleXMLElement($request, NULL, TRUE);

foreach ($ResultSet as $Result)
{
    echo '<p><a href="';
    echo htmlentities($Result->Url, ENT_COMPAT, 'UTF-8');
    echo '">';
    echo htmlentities($Result->Title, ENT_COMPAT, 'UTF-8');
    echo '</a><br/>';
    echo htmlentities($Result->Summary, ENT_COMPAT, 'UTF-8');
    echo '</p>';
}

?>
```

Flickr

- Provides a variety of Web Service interfaces, including REST
- Accomplished in an RPC fashion
- Uses a complex token authentication handshake to access user data
- http://flickr.com/services/api/

login.php

```php
<?php

require_once 'conf.php';

$flickr = $_SESSION['flickr'];

// Create API Signature for authentication
$api_sig  = $flickr['secret'];
$api_sig .= 'api_key' . $flickr['api_key'];
$api_sig .= 'perms' . $flickr['perms'];
$api_sig = md5($api_sig);

// Create link for Flickr authentication
$link  = 'http://flickr.com/services/auth';
$link .= '?api_key=' . urlencode($flickr['api_key']);
$link .= '&perms=' . urlencode($flickr['perms']);
$link .= '&api_sig=' . urlencode($api_sig);

?>

<a href="<?php echo $link; ?>">Authenticate With Flickr</a>
```

flickr.php

```php
<?php

session_start();

// Signature elements to reuse
$sig  = $_SESSION['flickr']['secret'];
$sig .= 'api_key' . $_SESSION['flickr']['api_key'];

// Get frob and convert it to a token for this user
$tok_sig  = $sig;
$tok_sig .= 'frob' . $_GET['frob'];
$tok_sig .= 'methodflickr.auth.getToken';
$tok_sig = md5($tok_sig);

// Create a token request URI
$tok_req  = 'http://api.flickr.com/services/rest/';
$tok_req .= '?api_key=';
$tok_req .= urlencode($_SESSION['flickr']['api_key']);
$tok_req .= '&frob=' . urlencode($_GET['frob']);
$tok_req .= '&api_sig=' . urlencode($tok_sig);
$tok_req .= '&method=flickr.auth.getToken';
```

flickr.php

```php
// Get a token
$tok_rsp = new SimpleXMLElement($tok_req, NULL, TRUE);
if ($tok_rsp->auth)
{
    $token = $tok_rsp->auth->token;
}
else
{
    echo htmlentities($tok_rsp->err['msg'],
                      ENT_COMPAT, 'UTF-8');
    exit;
}
```

flickr.php

```php
// Create authorization signature
$auth_sig  = $sig;
$auth_sig .= 'auth_token' . $token;
$auth_sig .= 'methodflickr.contacts.getList';
$auth_sig = md5($auth_sig);

// Create a contacts request URI
$contact_req  = 'http://api.flickr.com/services/rest/';
$contact_req .= '?api_key=';
$contact_req .= urlencode($_SESSION['flickr']['api_key']);
$contact_req .= '&auth_token=' . urlencode($token);
$contact_req .= '&api_sig=' . urlencode($auth_sig);
$contact_req .= '&method=flickr.contacts.getList';
```

flickr.php

```php
// Get list of contacts
$contact_rsp = new SimpleXMLElement($contact_req, NULL, TRUE);

foreach ($contact_rsp->contacts->contact as $contact)
{
    if (strlen(trim($contact['realname'])) > 0)
    {
        $name = $contact['realname'];
    }
    else
    {
        $name = $contact['username'];
    }

    echo htmlentities($name, ENT_COMPAT, 'UTF-8') . "<br />\n";
}

?>
```

Creating RESTful Web Services

Why Provide a Service?

- You have a service that benefits your users best if they can get to their data from outside the application
- You want others to use your data store in their applications
- All the cool kids are doing it

Designing a RESTful Service

- Adhere to the principles of REST
  - Diverse resources/nouns
  - Unique address for each resource
  - Constrained interface for resources (GET)
  - Transfers are atomic/stateless
- Your URI structure is your API

Designing a RESTful Service

- Example: Catalog of books
- Design the application with a specific URI structure in mind
  - http://example.org/catalog
  - http://example.org/catalog/book
  - http://example.org/catalog/book/1234

Designing a RESTful Service

- We can expand our catalog and service with ease
  - http://example.org/catalog/movie
  - http://example.org/catalog/movie/1234
- Keep the URIs clean and simple
- URIs should indicate the kind of data the consumer will receive

/catalog/book?isbn=014143984X

```xml
<?xml version="1.0" encoding="UTF-8"?>
<catalog>
    <book>
        <isbn>014143984X</isbn>
        <title>Dracula</title>
        <author>
            <surname>Stoker</surname>
            <name>Bram</name>
        </author>
        <publisher>Penguin Books</publisher>
    </book>
</catalog>
```

/catalog/book?isbn=014143984X&format=json


```
{"isbn":"014143984X","title":"Dracula",
"author":{"surname":"Stoker","name":"Bram"},
"publisher":"Penguin Books"}
```

Implementing a RESTful Service

- Use DOM to generate XML documents from a data model
- Use json_encode() to convert arrays/ objects into JSON
- Use Zend_Rest_Server to create a REST server

Using Zend_Rest_Server

- Determine the URI structure of the service
  - /catalog/book?isbn=123456789X
- Create a CatalogController class with a bookAction() method
- Create a catalog class to return data from the model to the REST server class

CatalogController.php

```php
<?php

class CatalogController extends Zend_Controller_Action
{
    public function indexAction() {}

    public function bookAction()
    {
        $filterGet = new Zend_Filter_Input($_GET);

        $request = array(
            'method' => 'book',
            'isbn'   => $filterGet->testAlNum('isbn')
        );

        $server = new Zend_Rest_Server();
        $server->setClass('catalog');
        $server->handle($request);
    }
}

?>
```

catalog.php

```php
class catalog
{
    private $books = array(
        '014143984X' => array(
            'isbn'      => '014143984X',
            'title'     => 'Dracula',
            'author'    => 'Bram Stoker',
            'publisher' => 'Penguin Books'
        )
    );

    public function book($isbn = FALSE)
    {
        if (array_key_exists($isbn, $this->books))
        {
            return $this->books[$isbn];
        }
        else
        {
            return array('error' => 'Book not found.');
        }
    }
}
```

/catalog/book?isbn=014143984X

```
- <catalog generator="zend" version="1.0">
   - <book>
        <isbn>014143984X</isbn>
        <title>Dracula</title>
        <author>Bram Stoker</author>
        <publisher>Penguin Books</publisher>
        <status>success</status>
    </book>
</catalog>
```

Zend_Rest_Server Caveats

- Zend Framework is at Preview 0.2.0
- Zend_Rest_Server is in the “incubator”
- Works only for very simple solutions
- Cannot handle multidimensional arrays yet
- For more than one level of tags, return a SimpleXMLElement object
- Only returns XML

Security Concerns

- A Web Service accepts data from remote applications/machines
  - **Filter all input**
- Output as XML, JSON, etc.
  - **Escape output accordingly**
- For authentication and sensitive data, force the use of SSL

Summary

- Creating RESTful Web Services
- Example RESTful Web Services
- Methods of Data Transport
- REST Overview
- Web Services

Slides & Further Reading

http://benramsey.com/archives/ipc06-slides/

And on the Conference CD-ROM