XML & Web Services With PHP

An Overview

Ben Ramsey
Zend/PHP Conference & Expo
October 31, 2006
Welcome

- BenRamsey.com
- I work for Art & Logic, Inc.
- PHP 5 Certification Study Guide author
- Fart around on #phpc
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
Why XML?

• Extensible Mark-up Language
• Flexible mark-up language
• Lightweight and easy to parse
• Communication between disparate systems
Types of Web Services

- XML-RPC
- SOAP
- REST
XML-RPC
What Is XML-RPC?

• XML Remote Procedure Call
• Specification maintained at xmlrpc.com (but no DTD, XSD, etc.)
• Provides a means to call methods/procedures on a remote server and make changes and/or retrieve data
• POST with XML request body and receive an XML response body
Using XML-RPC

- Most common implementation of XML-RPC used today is that of blog ping services
- Technorati, Flickr, others?
- Use PEAR::XML_RPC to access and create XML-RPC services
- SOAP is its successor
SOAP
What Is SOAP?

• Previously an acronym for Simple Object Access Protocol
• Version 1.2 of the W3C recommendation dropped the acronym
• SOAP is not simple!
• Specification maintained at w3.org
What Is SOAP?

• Provides a mechanism for various messaging patterns

• All messages sent in a SOAP envelope that is an XML wrapper for data read and generated by the SOAP server

• Most common message pattern is the Remote Procedure Call (RPC) pattern
SOAP In Short

- SOAP provides a means to interact with a remote system by sending it commands and getting a response.
- It is the natural successor of XML-RPC.
Using SOAP

• Send a message specifying an action to take, including data for the action
• Receive a return value from the action
• Most SOAP services provide a WSDL file to describe the actions provided by the service
WSDL

- Web Services Description Language
- XML mark-up for describing the functionality provided by a SOAP service
<definitions name="GoogleSearch">
  <targetNamespace>urn:GoogleSearch</targetNamespace>
  <xmlns:typens="urn:GoogleSearch">
    <xsd:schema>
      <xsd:complexType name="GoogleSearchRequest">
        <xsd:sequence>
          <xsd:element name="key" type="xsd:string"/>
          <xsd:element name="q" type="xsd:string"/>
          <xsd:element name="start" type="xsd:int"/>
          <xsd:element name="maxResults" type="xsd:int"/>
          <xsd:element name="filter" type="xsd:boolean"/>
          <xsd:element name="restrict" type="xsd:string"/>
          <xsd:element name="safeSearch" type="xsd:boolean"/>
          <xsd:element name="lr" type="xsd:string"/>
          <xsd:element name="ie" type="xsd:string"/>
          <xsd:element name="oe" type="xsd:string"/>
        </xsd:sequence>
      </xsd:complexType>
    </xsd:schema>
  </xmlns:typens>
  <xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  </xmlns:xsd>
  <xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/">
  </xmlns:soap>
  <xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/">
  </xmlns:soapenc>
  <xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/>
  </xmlns:wsdl>
  <xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">
  </xmlns:wsdl>
</definitions>

<message name="doGoogleSearch">
  <part name="key" type="xsd:string"/>
  <part name="q" type="xsd:string"/>
  <part name="start" type="xsd:int"/>
  <part name="maxResults" type="xsd:int"/>
  <part name="filter" type="xsd:boolean"/>
  <part name="restrict" type="xsd:string"/>
  <part name="safeSearch" type="xsd:boolean"/>
  <part name="lr" type="xsd:string"/>
  <part name="ie" type="xsd:string"/>
  <part name="oe" type="xsd:string"/>
</message>

<service name="GoogleSearchService">
  <port name="GoogleSearchPort" binding="typens:GoogleSearchBinding">
    <soap:address location="http://api.google.com/search/beta2"/>
  </port>
</service>
PHP 5 Makes It Easy to Access a SOAP Service

Example: Google SOAP Search API
<?php
try{
    $client = new SoapClient('http://api.google.com/GoogleSearch.wsdl');
    $results = $client->doGoogleSearch($key, $query, 0, 10, FALSE, '', FALSE, '', '', '');
    foreach ($results->resultElements as $result)
    {
        echo '<a href="'.
        echo htmlentities($result->URL, ENT_COMPAT, 'UTF-8');
        echo '">';
        echo htmlentities($result->title, ENT_COMPAT, 'UTF-8');
        echo '</a><br/>';
    }
}
catch (SoapFault $e)
{
    echo $e->getMessage();
}
?>
Providing a Service

• Create a class that contains public methods for the SOAP server to use

▶ This is the service you want to provide

• Instantiate a SoapServer object using the class

• Optionally create and provide a WSDL file (PHP 5 does not do this for you)
<?php

class MySoapServer
{
    public function getMessage()
    {
        return 'Hello, World!';
    }

    public function addNumbers($num1, $num2)
    {
        return $num1 + $num2;
    }
}

$options = array('uri' => 'http://example.org/soap/server/');
$server = new SoapServer(NULL, $options);
$server->setClass('MySoapServer');
$server->handle();
?>
<?php

$options = array(
    'location' => 'http://example.org/soap/server/server.php',
    'uri' => 'http://example.org/soap/server/
);
$client = new SoapClient(NULL, $options);

$message = $client->getMessage();
$addition = $client->addNumbers(3, 5);

?>
REST
What is REST?

- Representational State Transfer
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
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
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”
Consuming a Service

• Send a GET request:
  http://search.yahooapis.com/WebSearchService/V1/webSearch?appid=ramsey&query=PHP

• Parse the response (with SimpleXML if receiving XML)
<?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 $Result)
{
    echo '<p><a href="';
    echo htmlentitites($Result->Url);
    echo '">
    echo htmlentitites($Result->Title, ENT_COMPAT, 'UTF-8');
    echo '</a><br/>
    echo htmlentitites($Result->Summary, ENT_COMPAT, 'UTF-8');
    echo '</p>';
}

?>
Providing a Service

- No specific REST service library; the design is up to you
- Keep URLs simple and easy to understand
- Each URL (combined with its querystring params) must uniquely identify the resource it requests
- Return XML, JSON, YAML, etc.
- Use a library for generating these formats
Consuming Web Services
Why Use Web Services?

- Access to content/data stores you could not otherwise provide (zip codes, news, pictures, reviews, etc.)
- Enhance site with a service that is not feasible for you to provide (maps, search, products, etc.)
- Combine these services into a seamless service you provide (mash-ups)
What Services Are Available?

- Google
- Yahoo!
- Amazon
- eBay
- Flickr
- del.icio.us
- etc.
Security Concerns

- Regardless of the provider, do not trust the validity of the data; it is tainted

  - Filter all incoming data

- Authentication schemes (HTTP Auth, tokens, etc.)
Providing 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
Which Service Is Right?

- REST provides a unique resource identifier for all data in the system
- SOAP does not but provides a means to send/receive remote procedure calls
- Many services provide multiple APIs
- Matter of preference
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
Further Reading

• See my Web site for slides and links:
  benramsey.com/archives/zendcon06-talk