Web Services Workshop: Tips, Code Samples, Explanations, and Downloads

Jason A. Clark
Head of Digital Access and Web Services
Montana State University Libraries
jaclark@montana.edu

Tips – Getting Started with Web Services

• Play in the sandbox – pick a service, study it
• Yahoo Developer Central: http://developer.yahoo.com/
• Amazon Web Services Developer Connection: http://developer.amazonwebservice.com/connect/
• Google Code: http://code.google.com/

Tips – Consuming Web Services

• Pick a language or parsing tool
• Find a few data sources (APIs) worth learning about
• Make some requests and look at code in your browser
• Think about added value, more efficient workflows
• Browse around – many language libraries are already written

Tips – Building Web Services

• URIs are your friends – that’s your interface
• Use simple CRUD (Create, Read, Update, Delete) functions over HTTP (Get, Delete, Put, Post)
• Keep verbs in API protocol intuitive and memorable
• Start small – simple, read-only requests
• Roll it out, beta version – once it’s public you are restricted

Tips – Web Services Data Sources

• AllCDCovers.com http://www.allcdcovers.com/api
• OpenDOAR http://www.opendoar.org/tools/api.html
• arXiv.org http://export.arxiv.org/api_help/
  * See programmableweb
        http://www.programmableweb.com/apis/directory

Web Services – Sample Applications

• Google Ajax Search API - Federate search of Google Data
• Amazon Reviews & Thumbnails (PHP)
• Flickr API - Display Photos (JSON)
  * View samples and download code at http://www.lib.montana.edu/~jason/files.php

Web Services – Building Blocks

1. REQUEST – learn the protocol, ask for the data
2. RESPONSE – receive the data
3. PARSE – pick the pieces you need
4. DISPLAY – format those pieces for display
Code Sample #1: Google Ajax Search API – Javascript and CSS

xHTML source:
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<link href="http://www.google.com/uds/css/gsearch.css" type="text/css" rel="stylesheet"/>
<style type="text/css">
body {background-color:white;color:black;font-family:Arial,sans-serif;font-size:small;margin:15px;}
.gsc-control {width:400px;}
</style>
<script src="http://www.google.com/uds/api?file=uds.js&amp;v=1.0" type="text/javascript"></script>
</head>
</html>

Javascript source:
(http://www.google.com/uds/api?file=uds.js&amp;v=1.0)
...
if (window['google'] != undefined && window['google']['loader'] != undefined) {
if (!window['google']['search']) {
window['google']['search'] = {};
}
google.search.CurrentLocale = 'en';
google.search.ShortDatePattern = 'MDY';
google.search.Version = '1.0';
google.search.NoOldNames = false;
google.search.JSHash = 'b2cf21b87d5348acb0a314b08588b757';
google.loader.ApiKey = 'notsupplied';
google.loader.KeyVerified = true;
google.loader.LoadFailure = false;
}
google.loader.writeLoadTag("script", google.loader.ServiceBase + "/api/search/1.0/en/b2cf21b87d5348acb0a314b08588b757/default.I.js", false);

Code Sample #1: Google Ajax Search API - Explanation

• Javascript written by Google – heavy lifting
  http://www.google.com/uds/api?file=uds.js&amp;v=1.0
• "google.loader.writeLoadTag" – tells API to run, sets possibilities for search API
• CSS written by Google – formatting and display
  http://www.google.com/uds/css/gsearch.css
• Understand these files, but you probably want to leave them as is – code library
Code Sample #2: Google Ajax Search API - Web page for user interface and display

...<script type="text/javascript">
  //<![CDATA[

  function OnLoad() {
    // Create a search control
    var searchControl = new GSearchControl();

    // Add in a full set of searchers
    var localSearch = new GlocalSearch();
    searchControl.addSearcher(localSearch);
    searchControl.addSearcher(new GwebSearch());
    searchControl.addSearcher(new GvideoSearch());
    searchControl.addSearcher(new GblogSearch());
    searchControl.addSearcher(new GnewsSearch());
    searchControl.addSearcher(new GimageSearch());
    searchControl.addSearcher(new GbookSearch());

    // Set the Local Search center point
    localSearch.setCenterPoint("Bozeman, MT");

    // tell the searcher to draw itself and tell it where to attach
    searchControl.draw(document.getElementById("searchcontrol"));

    // execute an initial search
    searchControl.execute("library books");
  }
  GSearch.setOnLoadCallback(OnLoad);
  //]]>
</script>
<div id="searchcontrol">Loading</div>

Code Sample #2: Google Ajax Search API - Explanation

• xHTML and javascript that gives action to our script
• Create the interface (GUI) controls with “var searchControl = new GSearchControl();”
• Set the local search parameter with “localSearch.setCenterPoint”
• Set the initial query with “searchControl.execute”
• Decide which pieces of Google data to federate with “searchControl.addSearcher”
• <div id="searchcontrol"> will be populated with script messages OR generated xHTML tags received via our Ajax requests
• Any customization begins with these parsing and display functions
Code Sample #1: Amazon Reviews & Thumbnails – The <form>

```html
<form id="checkAmazon" name="checkAmazon" action="<?php echo basename(__FILE__); ?>" method="get">
  <fieldset>
    <h3><label for="id">Enter an ISBN or Amazon ASIN for details:</label></h3>
    <p><input type="text" id="id" name="id" value="0596005601" onfocus="this.value='';
this.onfocus=null;" /></p>
    <p><input class="submit" id="submit" name="submit" type="submit" value="Check Amazon" /></p>
  </fieldset>
</form>
```

Code Sample #1: Amazon Reviews & Thumbnails - Explanation

- xHTML form that makes the web services request happen
- The markup: `<input type="text" id="id" name="id" value="0596005601" ...`
  *When submitted, passes a $_GET variable to Amazon with id number
- Requests a specific item formatted as an XML response

Code Sample #2: Amazon Reviews & Thumbnails – The URL Request

```
```

Code Sample #2: Amazon Reviews & Thumbnails - Explanation

- HTTP Request to Amazon E-Commerce Service
- Anatomy of a REST URL – a closer look behind the scenes of the xHTML <form> and PHP logic
- Name the “Service” Requested
- Identify the developer with “AWSAccessKeyId”
- Specify the action of the request with “Operation”
- Identify the item with “ItemId”
- Specify the types of data returned with “ResponseGroup”
- Identify the version of the API with “Version”
Code Sample #3: Amazon Reviews & Thumbnails – The XML Response

```xml
<?xml version="1.0" encoding="utf-8"?>
  <OperationRequest>
    <HTTPHeaders>
      <Header Name="UserAgent" Value="Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; rv:1.8.1.13) Gecko/20080311 Firefox/2.0.0.13"></Header>
    </HTTPHeaders>
    <RequestId>c804ab64-4b98-4e2b-bf1b-bcf64a446d58</RequestId>
    <Arguments>
      <Argument Name="ItemId" Value="0596005601"></Argument>
      <Argument Name="Service" Value="AWSECommerceService"></Argument>
      <Argument Name="Operation" Value="ItemLookup"></Argument>
      <Argument Name="AWSAccessKeyId" Value="1DBY9V8DKZ6RAK1M7NG2"></Argument>
      <Argument Name="ResponseGroup" Value="Images,ItemAttributes,EditorialReview,Reviews"></Argument>
      <Argument Name="Version" Value="2007-07-16"></Argument>
    </Arguments>
    <RequestProcessingTime>0.0220150000000000</RequestProcessingTime>
  </OperationRequest>
  <Items>
    <Request>
      <IsValid>True</IsValid>
      <ItemLookupRequest>
        <Condition>New</Condition>
        <DeliveryMethod>Ship</DeliveryMethod>
        <IdType>ASIN</IdType>
        <MerchantId>Amazon</MerchantId>
        <OfferPage>1</OfferPage>
        <ItemId>0596005601</ItemId>
        <ResponseGroup>Images</ResponseGroup>
        <ResponseGroup>ItemAttributes</ResponseGroup>
        <ResponseGroup>EditorialReview</ResponseGroup>
        <ResponseGroup>Reviews</ResponseGroup>
        <ReviewPage>1</ReviewPage>
        <ReviewSort>-SubmissionDate</ReviewSort>
        <VariationPage>All</VariationPage>
      </ItemLookupRequest>
      <Item>
        <ASIN>0596005601</ASIN>
        <DetailPageURL>http://www.amazon.com/gp/redirect.html%3FASIN=0596005601%26tag=ws%26code=xm2%26cID=2025%26cmmId=165953%26location=o/ASIN/0596005601%253FSubscriptionId=1DBY9V8DKZ6RAK1M7NG2</DetailPageURL>
        <SmallImage>
          ...
        </SmallImage>
        <MediumImage>
          <Height Units="pixels">160</Height>
          <Width Units="pixels">119</Width>
        </MediumImage>
        <LargeImage>
          ...
        </LargeImage>
      </Item>
    </Request>
  </Items>
</ItemLookupResponse>
```
Amazing Learning PHP 5

This books is amazing for the people who want know the new features in PHP 5.

PHP has gained a following among non-technical web designers who need to add interactive aspects to their sites. Offering a gentle learning curve, PHP is an accessible yet powerful language for creating dynamic web pages. As its popularity has grown, PHP's basic ...
Code Sample #3: Amazon Reviews & Thumbnails - Explanation

• XML that we will parse with PHP
• Tons of structured information
• Request data, Image Data, Item Data, Publisher Data, Price Data, Dewey Data, etc.
• Memorize the XML tag structure – what’s nested? Parent->Child nodes
• Start thinking how to “cherrypick” the data we want

Code Sample #4: Amazon Reviews & Thumbnails – Request with PHP

```php
<?php
  //set Amazon Web Services Developer ID - MUST be changed for personal use, accounts available
  $aws_developer_key = '1DBY9V8DKZ6RAK1M7NG2';

  //build request URL for specific developer and item id

  //make request to Amazon E-Commerce Web Service using "xml load file" function from PHP
  $xml = simplexml_load_file($request) or die("xml response not loading");
  ...
?>
```

Code Sample #4: Amazon Reviews & Thumbnails - Explanation

• Piece of PHP script that builds web services call to Amazon API
• Loads requested data into PHP native function “simplexml_load_file”
• Requested data is loaded and stored in array – ready to be parsed with PHP

Code Sample #5: Amazon Reviews & Thumbnails – Parse with PHP

```php
<?php
  ...
  //set Amazon xml values as specific variables to be printed out below
  $image = $xml->Items->Item->MediumImage->URL;
  ...
  $title = $xml->Items->Item->ItemAttributes->Title;
  $author = $xml->Items->Item->ItemAttributes->Author;
  //simple logic check for author and director values, shows
  if (strlen($author) > 2) {
    $creator = $author;
  } elseif (empty($author)) {
    $creator = $xml->Items->Item->ItemAttributes->Director;
  } else {
    $creator = '* Creator Not Available';
  }
  $asin = $xml->Items->Item->ASIN;
  $uri = $xml->Items->Item->DetailPageURL;
  $editorialReview = $xml->Items->Item->EditorialReviews->EditorialReview->Content;
  ...
?>
```
Code Sample #5: Amazon Reviews & Thumbnails - Explanation

- Using $xml variable created above from line: $xml = simplexml_load_file($request);
- Traverse XML response using PHP simple_xml array notation

For example: the original XML response is structured as...

```xml
<Items>
  <Item>
    <ASIN>

We traverse this structure using the following PHP:

```php
$asin = $xml->Items->Item->ASIN;
```

- Each piece of data that we grab is stored as a $variable to be used later

Code Sample #6: Amazon Reviews & Thumbnails – Display with PHP

```php
<?php
...
//print out Amazon xml values as html
echo '<img class="thumbnail" src="'.$image.'" />
';
echo '<h2 class="mainHeading">'.$title.'</h2>
';
echo '<p>'.$creator.'<br />ID (isbn or asin): '.$asin.'<br /><a href="'.$uri.'">+ Get full details</a></p>
';
echo '<p>Editorial review: '.$html_entity_decode($editorialReview).'</p>
';
echo '<h2 class="mainHeading">What others are saying...</h2>
';
echo '<dl>
  foreach ($xml->Items->Item->CustomerReviews->Review as $review) {
    echo '<dt><strong>'.$html_entity_decode($review->Summary).'</strong></dt>
    echo '<dd>'.$html_entity_decode($review->Content).'</dd>
    echo '<dd>Rating: '.$html_entity_decode($review->Rating).' out of 5</dd>
  }<hr />
  echo '</dl>

...?
```

Code Sample #6: Amazon Reviews & Thumbnails - Explanation

- Using $variables created above, place values within xHTML markup
- foreach ($xml->Items->Item->CustomerReviews->Review as $review) *
  Programming loop that retrieves reviews and ratings using PHP simple_xml array notation
- Page is served up as basic xHTML

Code Sample #1: Flickr API - Display Photos (JSON) – The URL Request

Code Sample #1: Flickr API - Display Photos (JSON) - Explanation

- HTTP Request to Flickr API
  http://www.flickr.com/services/api/
- API provides data as XML feeds (RSS, ATOM)
- Requesting "/feeds/" with a "format" of JSON (Javascript Object Notation)
- Querying API for all public photos tagged "cil2008" with the "tags" parameter

Code Sample #2: Flickr API - Display Photos (JSON) – The URL Request in Javascript

<!-- use script tag to make request to flickr api, specify json format and tag to search -->
<script type="text/javascript"
</script>

Code Sample #2: Flickr API - Display Photos (JSON) - Explanation

- JSON is actually javascript and to make JSON output available we must call it on the page via the <script> tag
- After <script> tag is run, JSON output exists as javascript object ready to be parsed

Code Sample #3: Flickr API - Display Photos (JSON) – JSON Response

```
jsonFlickrFeed({
    "title": "Photos from everyone tagged cil2008",
    "link": "http://www.flickr.com/photos/tags/cil2008/",
    "description": "",
    "modified": "2008-04-07T18:43:16Z",
    "generator": "http://www.flickr.com/",
    "items": [
      {
        "title": "So many floors",
        "link": "http://www.flickr.com/photos/nengard/2395908509/",
        "media": {"m":"http://farm4.static.flickr.com/3182/2395908509_d6452e2d56_m.jpg"},
        "date_taken": "2008-04-07T13:07:53-08:00",
        "description": "So many floors",
        "published": "2008-04-07T18:43:16Z",
        "author": "nobody@flickr.com (nengard)",
        "author_id": "10137764@N00",
        "tags": "hyatt cil2008 cil08"
      },
      ...
    ]
  })
```

Code Sample #3: Flickr API - Display Photos (JSON) - Explanation

- More structured data ready to be parsed
- We’ll extract the values and format for display using the second javascript
Code Sample #4: Flickr API - Display Photos (JSON) – Parse and display with Javascript

<script type="text/javascript">
//run function to parse json response, grab title, link, and media values - place in html tags
function jsonFlickrFeed(fr) {
    for (var i = 0; i < fr.items.length;i++) {
        document.write('<a title="' + fr.items[i].title + '" href="' + fr.items[i].link + '"><img src="' + fr.items[i].media.m + " alt="' + fr.items[i].title + '"></a>');
    }
}
</script>

Code Sample #4: Flickr API - Display Photos (JSON) - Explanation

• Create javascript function “jsonFlickrFeed” to parse JSON response returned from first javascript
• Loop statement: “for (var i = 0; i < fr.items.length;i++)” runs through all JSON data nodes
• “document.write” – native javascript function prints out values from JSON in xHTML markup

Final Thoughts

• Start with simpler data formats – RSS and ATOM are well-supported
• Keep experimenting and learning with a single web service, become a seasoned veteran
• Remember the primary actions for using web services: request, response, parse, display
  * Translate these actions into your favorite tool or scripting language