Web Services for Libraries: Tips, Code Samples, Explanations, and Downloads

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

Karen A. Coombs
Head of Web Services
University of Houston Libraries
kacoombs@uh.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.amazonwebservices.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
• ISBNdb.com http://isbndb.com/docs/api/index.html

• OpenDOAR http://www.opendoar.org/tools/api.html
• arXiv.org http://export.arxiv.org/api_help/
• Google Book Search APIs http://code.google.com/apis/books/
• LibraryThing APIs http://www.librarything.com/services/
• WorldCat Search API http://worldcat.org/devnet/wiki/SearchAPIDetails
* 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>
…
```

Javascript source:
(http://www.google.com/uds/api?file=uds.js&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&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 inital 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”

• 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>

```
<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

http://ecs.amazonaws.com/onca/xml?Service=AWSECommerceService&AWSAccessKeyId=1DBY9V8DKZ6RAK1M7NG2&Operation

Code Sample #2: Amazon Reviews & Thumbnails - Explanation

• HTTP Request to Amazon E-Commerce Service
http://docs.amazonwebservices.com/AWSEcommerceService/2005-02-23/
• 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 version="1.0" encoding="utf-8"?>
<ItemLookupResponse xmlns="http://webservices.amazon.com/AWSECommerceService/
2007-07-16">
<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>
</Request>
```

<Item>
<ASIN>0596005601</ASIN>
<DetailPageURL>http://www.amazon.com/gp/
redirect.html%3FASIN=0596005601%26tag=ws%26lcode=xm2%26cID=2025%26ccmID=165953%26loc
o/ASIN/0596005601%253FSubscriptionId=1DBY9V8DKZ6RAK1M7NG2</DetailPageURL>
<SmallImage>
…
</SmallImage>
<MediumImage>
<URL>http://ecx.images-amazon.com/images/I/51TdB0QiQfL._SL160_.jpg</URL>
<Height Units="pixels">160</Height>
<Width Units="pixels">119</Width>
</MediumImage>
<LargeImage>
<URL>http://ecx.images-amazon.com/images/I/51TdB0QiQfL.jpg</URL>
<Height Units="pixels">500</Height>
<Width Units="pixels">373</Width>
</LargeImage>
...
<ItemAttributes>
<Author>David Sklar</Author>
<Binding>Paperback</Binding>
<DeweyDecimalNumber>005.133</DeweyDecimalNumber>
<EAN>9780596005603</EAN>
<Edition>1st</Edition>
<Format>Illustrated</Format>
<ISBN>0596005601</ISBN>
<Label>O'Reilly Media, Inc.</Label>
<Languages>
<Language>
<Name>English</Name>
<Type>Original Language</Type>
</Language>
</Languages>
<ListPrice>
<Amount>2995</Amount>
<CurrencyCode>USD</CurrencyCode>
<FormattedPrice>$29.95</FormattedPrice>
</ListPrice>
<Manufacturer>O'Reilly Media, Inc.</Manufacturer>
<NumberOfItems>1</NumberOfItems>
<NumberOfPages>368</NumberOfPages>

<PackageDimensions>
<Height Units="hundredths-inches">87</Height>
<Length Units="hundredths-inches">906</Length>
<Weight Units="hundredths-pounds">110</Weight>
<Width Units="hundredths-inches">685</Width>
</PackageDimensions>
<ProductGroup>Book</ProductGroup>
<ProductTypeName>ABIS_BOOK</ProductTypeName>
<PublicationDate>2004-07</PublicationDate>
<Publisher>O'Reilly Media, Inc.</Publisher>
<Studio>O'Reilly Media, Inc.</Studio>
<Title>Learning PHP 5</Title>
<UPC>636920005605</UPC>
</ItemAttributes>
<CustomerReviews>
<AverageRating>3.5</AverageRating>
<TotalReviews>24</TotalReviews>
<TotalReviewPages>5</TotalReviewPages>
<Review>
<ASIN>0596005601</ASIN>
<Rating>5</Rating>
<HelpfulVotes>0</HelpfulVotes>
<CustomerId>A1PYNEXX4J7MQD</CustomerId>
<Reviewer>
<CustomerId>A1PYNEXX4J7MQD</CustomerId>
<Name>Luis Jose Muãiz Rascado</Name>
</Reviewer>
<TotalVotes>0</TotalVotes>
<Date>2007-10-30</Date>
<Summary>Amazing Learning PHP 5</Summary>
<Content>This books is amazing for the people who want know the new features in PHP 5</Content>
</Review>
</CustomerReviews>
<EditorialReviews>
<EditorialReview>
<Source>Book Description</Source>
<Content>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 …
</Content>

<IsLinkSuppressed>0</IsLinkSuppressed>
</EditorialReview>
</EditorialReviews>
</Item>
</Items>
</ItemLookupResponse>

* Full XML response can be viewed at http://ecs.amazonaws.com/onca/xml?Service=AWSECommerceService&AWSAccessKeyId=1DBY9V8DKZ6RAK1M7NG2&Operation=ItemLook

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
//set Amazon Web Services Developer ID - MUST be changed for personal use, accounts
available from https://aws-portal.amazon.com/gp/aws/developer/registration/index.html
$aws_developer_key = '1DBY9V8DKZ6RAK1M7NG2';
//build request URL for specific developer and item id
$request = 'http://ecs.amazonaws.com/onca/
xml?Service=AWSECommerceService&AWSAccessKeyId='.$aws_developer_key.'&Operation=ItemLookup&
//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
…
//set Amazon xml values as specifc 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…

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

We traverse this structure using the following 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
…
//print out Amazon xml values as html
echo '<img class="thumbnail" src="'.$image.'" />'."\n";
echo '<h2 class="mainHeading">'.$title.'</h2>'."\n";
echo '<p>'.$creator.'<br />ID (isbn or asin): '.$asin.'<br /><a href="'.$uri.'">+ Get full
details</a></p>'."\n";
echo '<p>Editorial review: '.html_entity_decode($editorialReview).'</p>'."\n";
echo '<h2 class="mainHeading">What others are saying...</h2>'."\n";
echo '<dl>'."\n";
foreach ($xml->Items->Item->CustomerReviews->Review as $review) {
echo '<dt><strong>'.html_entity_decode($review->Summary).'</strong></dt>'."\n";
echo '<dd>'.html_entity_decode($review->Content).'</dd>'."\n";
```

```
echo '<dd>Rating: '.html_entity_decode($review->Rating).' out of 5</dd>'."\n";
echo '<hr />'."\n";
}
echo '</dl>'."\n";
…
?>
```

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

http://api.flickr.com/services/feeds/photos_public.gne?tags=cil2008&format=json

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" src="http://api.flickr.com/services/feeds/
photos_public.gne?tags=cil2008&format=json">
</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

Library Web Services – Sample Applications

• WorldCat Search API (SRU/W) - Search WorldCat

• LibraryThing API - Rating Information
• Google Book Search API - Full-text Link or Book Preview

Code Sample: WorldCat Search API (SRU/W)

The form

```
<form action="worldcat_search.php" method="get">
  <p>
    <label for="AddWorldCatSearch-SearchType">Search Type</label>
    <select name="AddWorldCatSearch-SearchType" id="AddWorldCatSearch-SearchType">
    <option value="srw.kw">Keyword</option>
    <option value="srw.ti">Title</option>
    <option value="srw.au">Author</option>
    </select>
</p>
<p>
    <label for="AddWorldCatSearch-SearchString">Search </label>
    <input type="text" id="AddWorldCatSearch-SearchString" name="AddWorldCatSearch-
SearchString" value="" />
</p>
<p>
    <label for="AddWorldCatSearch-LibraryLimit">Limit by Specific Library (OCLC
Symbol)</label>
    <input type="text" id="AddWorldCatSearch-LibraryLimit" name="AddWorldCatSearch-
LibraryLimit" value="" />
</p>
</form>
```

Form explanation

Select box for the different indexes which can be searched
Text box for the search string text
Text box for limiting by a particular library's OCLC symbol

The URL Request

http://worldcat.org/webservices/catalog/search/worldcat/
sru?query=srw.kw%3D%22ambient+findability%22+and+srw.li%3D%22TXH%22&recordSchema=info%3

URL Request Explanation

HTTP Request to WorldCat's Search API Service
Specify request protocol - sru

Specify the search being submitted and its components with query
Specify response format with recordSchema
Specify the service level with servicelevel
Identify the developer with wskey

XML Response (snippet)

```
<record xmlns="http://www.loc.gov/MARC21/slim">
<leader>99999cam a22000018a 4500cam  8a </leader>
<controlfield tag="001">61260129</controlfield>
<controlfield tag="008">050712s2005    caua     b    000 0 eng  </controlfield>
<datafield tag="040" ind1=" " ind2=" ">
<subfield code="a">UKM</subfield>
<subfield code="c">UKM</subfield>
<subfield code="d">BAKER</subfield>
<subfield code="d">PSM</subfield>

<subfield code="d">UNA</subfield>
<subfield code="d">DPL</subfield>
<subfield code="d">IXA</subfield>
<subfield code="d">VP@</subfield>
<subfield code="d">MIR</subfield>
<subfield code="d">TWU</subfield>
<subfield code="d">YDXCP</subfield>
<subfield code="d">LVB</subfield>
<subfield code="d">XY4</subfield>

<subfield code="d">UPP</subfield>
</datafield>
<datafield tag="015" ind1=" " ind2=" ">
<subfield code="a">GBA566896</subfield>
<subfield code="2">bnb</subfield>
</datafield>
<datafield tag="016" ind1="7" ind2=" ">
<subfield code="a">013269507</subfield>
<subfield code="2">Uk</subfield>
</datafield>
<datafield tag="020" ind1=" " ind2=" ">
<subfield code="a">0596007655 (pbk.)</subfield>

</datafield>
<datafield tag="020" ind1=" " ind2=" ">
<subfield code="a">9780596007652 (pbk.)</subfield>
</datafield>
<datafield tag="029" ind1="1" ind2=" ">
```

```
<subfield code="a">NLGGC</subfield>
<subfield code="b">286781794</subfield>
</datafield>
<datafield tag="029" ind1="1" ind2=" ">
<subfield code="a">YDXCP</subfield>
<subfield code="b">2257898</subfield>
</datafield>

<datafield tag="050" ind1=" " ind2="4">
<subfield code="a">QA76.9.D26</subfield>
<subfield code="b">M67 2005</subfield>
</datafield>
<datafield tag="082" ind1="0" ind2="4">
<subfield code="a">005.72</subfield>
<subfield code="2">22</subfield>
</datafield>
<datafield tag="100" ind1="1" ind2=" ">
<subfield code="a">Morville, Peter.</subfield>
</datafield>
<datafield tag="245" ind1="1" ind2="0">

<subfield code="a">Ambient findability /</subfield>
<subfield code="c">Peter Morville.</subfield>
</datafield>
<datafield tag="260" ind1=" " ind2=" ">
<subfield code="a">Sebastopol, Calif. ;</subfield>
<subfield code="a">Farnham :</subfield>
<subfield code="b">O'Reilly,</subfield>
<subfield code="c">2005.</subfield>
</datafield>
<datafield tag="263" ind1=" " ind2=" ">
<subfield code="a">200510</subfield>

</datafield>
<datafield tag="300" ind1=" " ind2=" ">
<subfield code="a">xiv, 188 :</subfield>
<subfield code="b">ill. (some col.) ;</subfield>
<subfield code="c">23 cm.</subfield>
</datafield>
<datafield tag="504" ind1=" " ind2=" ">
<subfield code="a">Includes bibliographical references and index.</subfield>
</datafield>
<datafield tag="505" ind1="0" ind2=" ">
<subfield code="a">Lost and found. Definition -- Information literacy -- Business value --
</datafield>
```

```
<datafield tag="520" ind1=" " ind2=" ">
<subfield code="a">How do you find your way in an age of information overload? How can you
</datafield>
<datafield tag="650" ind1=" " ind2="0">
<subfield code="a">Database design.</subfield>
</datafield>
<datafield tag="650" ind1=" " ind2="0">
<subfield code="a">Database searching.</subfield>
</datafield>
<datafield tag="650" ind1=" " ind2="0">
<subfield code="a">Information and retrieval systems</subfield>
<subfield code="x">Design.</subfield>

</datafield>
<datafield tag="650" ind1=" " ind2="6">
<subfield code="a">Bases de donne´es</subfield>
<subfield code="x">Conception.</subfield>
</datafield>
<datafield tag="650" ind1=" " ind2="6">
<subfield code="a">Bases de donne´es</subfield>
<subfield code="x">Interrogation.</subfield>
</datafield>
<datafield tag="650" ind1=" " ind2="6">
<subfield code="a">Syste`mes d'information.</subfield>
</datafield>

<datafield tag="938" ind1=" " ind2=" ">
<subfield code="a">Baker &amp;amp; Taylor</subfield>
<subfield code="b">BKTY</subfield>
<subfield code="c">29.95</subfield>
<subfield code="d">22.46</subfield>
<subfield code="i">0596007655</subfield>
<subfield code="n">0006585485</subfield>
<subfield code="s">active</subfield>

</datafield>
<datafield tag="938" ind1=" " ind2=" ">
<subfield code="a">YBP Library Services</subfield>
<subfield code="b">YANK</subfield>
<subfield code="n">2257898</subfield>
</datafield>
</record>
```


Request with PHP

```
$searchURL = 'http://worldcat.org/webservices/catalog/search/sru?query=' .
$options['AWC_searchType'] . '%3D%22' . urlencode($options['AWC_searchString']) .
'%22';

   // Need to insert maximumRecords syntax
   if ( isset($option['AWC_libraryLimit']) ) {
      $searchURL = $searchURL . '+and+srw.li%3D%22' . $options['AWC_libraryLimit'] .
'%22';
   }

   $searchURL = $searchURL .  '&maximumRecords=' . $options['AWC_numberResults'];

   // djk 08.11.2008: temp fix adding in wskey
   $searchURL .= '&wskey=12345kac';

   $xml = simplexml_load_file($searchURL);
```

Request with PHP Explanation

Build the request URL by using the variables submitted via the form

Parse and display with PHP

```
$xml = simplexml_load_file($searchURL);

   $xml->registerXPathNamespace("marc", "http://www.loc.gov/MARC21/slim");

   foreach($xml->xpath('//marc:record') as $book ) {
      $book['xmlns:marc'] = 'http://www.loc.gov/MARC21/slim';
      $field = simplexml_load_string($book->asXML());
      $title = $field->xpath("marc:datafield[@tag='245']/marc:subfield[@code='a']");
      $publisher = $field->xpath("marc:datafield[@tag='260']/marc:subfield[@code='b']");
      $publication_date = $field->xpath("marc:datafield[@tag='260']/
marc:subfield[@code='c']");
      $isbn = $field->xpath("marc:datafield[@tag='020']/marc:subfield[@code='a']");
      $isbn_1 = substr($isbn[0], 0, strpos($isbn[0], " "));
      $author = $field->xpath("marc:datafield[@tag='100']/marc:subfield[@code='a']");
      $oclcnumber = $field->xpath("marc:controlfield[@tag='001']");

      echo '<p><a href="http://www.worldcat.org/oclc/' . $oclcnumber[0] . '"><span>' .
$title[0] . '</span></a></p>';
   }
```

Parse and display xplanation

Add the marc namespace in order to properly select nodes.
Use xpath to gather a set of all the marc records returned.
Loop through each marc record as book and pull out specific information using xpath: title, publisher, publication_date, isbn, author, and oclc number.
Print HTML to display each book's title with a link to WorldCat.org

Code Sample: WorldCat Search API (OpenSearch)

The form

```
<form action="worldcat_search_atom.php" method="get">
<p>
   <label for="AddWorldCatSearch-SearchString">Search </label>
   <input type="text" id="AddWorldCatSearch-SearchString" name="AddWorldCatSearch-
SearchString" value="" />
</p>
</form>
```

Form explanation

Text box for the search

The URL Request

http://worldcat.org/webservices/catalog/search/worldcat/
opensearch?q=ambient%20findability&format=atom&servicelevel=default&wskey=12345kac

URL Request Explanation

HTTP Request to WorldCat's Search API Service
Specify request protocol - opensearch
Specify the search being submitted and its components with q
Specify the response format with format
Specify the service level with servicelevel
Identify the developer with wskey

XML Response (snippet)

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<feed xmlns="http://www.w3.org/2005/Atom" xmlns:opensearch="http://a9.com/-/spec/opensearc
<title>OCLC Worldcat Search: ambient findability</title>
<id>http://worldcat.org/webservices/catalog/search/worldcat/opensearch?q=ambient+findabili
<updated>2008-09-26T15:40:55-04:00</updated>
<subtitle>Search results for ambient findability at http://worldcat.org/webservices/catalo
```

```
<opensearch:totalResults>1</opensearch:totalResults>
<opensearch:startIndex>1</opensearch:startIndex>

<opensearch:itemsPerPage>10</opensearch:itemsPerPage>
<opensearch:Query role="request" searchTerms="ambient findability" startPage="1"/>
<link rel="alternate" href="http://worldcat.org/webservices/catalog/search/worldcat/opense
<link rel="self" href="http://worldcat.org/webservices/catalog/search/worldcat/opensearch?
<link rel="first" href="http://worldcat.org/webservices/catalog/search/worldcat/opensearch

<link rel="last" href="http://worldcat.org/webservices/catalog/search/worldcat/opensearch?
<link href="http://worldcat.org/oclc/opensearchdescription.xml" type="application/opensear
<entry>
<author>
<name>Morville, Peter.</name>
</author>
<title>Ambient findability</title>
<link href="http://worldcat.org/oclc/61260129"/>
<id>http://worldcat.org/oclc/61260129</id>
<updated>2007-08-14T17:49:39Z</updated>

<summary>How do you find your way in an age of information overload? How can you filter st
</entry>
</feed>
```

Request with PHP

```
$searchURL = 'http://worldcat.org/webservices/catalog/search/worldcat/opensearch?q=' .
urlencode($_REQUEST['AddWorldCatSearch-SearchString']);

   // Format (Atom) WorldCat Service Level and wskey
   $searchURL .= '&format=atom&servicelevel=full&wskey=12345kac';
   $xml = simplexml_load_file($searchURL);
```

Request with PHP Explanation

Build the request URL by using the variables submitted via the form

Parse and Display with PHP

```
   $xml->registerXPathNamespace("opensearch", "http://a9.com/-/spec/opensearch/1.1/
");
   $xml->registerXPathNamespace("atom", "http://www.w3.org/2005/Atom");
      foreach($xml->xpath('//atom:entry') as $book ) {
         $field = simplexml_load_string($book->asXML());
         $title = $field->title;
```

```
$author = $field->author->name;
$link = $field->link['href'];
$summary = $field->summary;

echo '<p><a href="' . $link . '">' . $title . '</a> by ' . $author . '</p>';
echo '<p>Summary: ' . $summary . '</p>';
```

Parse and Display Explanation

Add the atom and opensearch namespaces in order to properly select nodes.
Use xpath to gather a set of all the records returned (entry).
Loop through each record as book and pull out specific information using dom: title, author, link and summary.
Print HTML to display each book's title, author and summary with a link to WorldCat.org

Code Sample: LibraryThing API - Ratings Information

HTML

```
<p><a href="http://www.worldcat.org/oclc/65183407">Ambient Findability</a><br/>
<span id="LT_0596007655"></span><br/>
<noscript><a href="http://www.librarything.com/isbn/0596007655">View Book
Information at LibraryThing</a></noscript>
```

HTML Explanation

Title Ambient Findability linked back to record in WorldCat. Span is a placeholder for LibraryThing rating information to be gathered via Javascript
noscript tag will create a link to the item at LibraryThing if Javascript is disabled

Javascript

```
   <script>
      function LTpop(booksInfo){
         for (i in booksInfo) {
            var book = booksInfo[i];
            if (book.link) {
               var desc = ''; var rating = '' ;
               if (book.reviews && (book.reviews != '0')) desc += book.reviews + ' reviews'
;
               if (book.rating) ratimg = ' <img border="1" src="' + book.rating_img + '"/>'
;
                document.getElementById('LT_'+book.id).innerHTML = ' <a href="' +
book.link + '">' + desc + ' @ LibraryThing ' + ratimg + '</a>';
```

```
          }
        }
      }
    </script>
    <script src='http://www.librarything.com/api/json/
workinfo.js?ids=0596007655&callback=LTpop'></script>
```

Javascript Explanation

The second set of Javascript
<script src='http://www.librarything.com/api/json/
workinfo.js?ids=0596007655&callback=LTpop'></script>
goes out and gets a JSON object from LibraryThing for the ISBN(s) given to it

The first Javascript takes the information returned from LibraryThing as JSON and parses it. If ratings exist then it creates and appropriate image of said rating (number of stars) and inserts it into the blank span with the appropriate id (the one with the same ISBN) in the HTML.

Code Sample: Google Book Search API - Full-text Link or Book Preview

HTML

```
<p><a href="http://www.worldcat.org/oclc/65183407">Ambient Findability</a><br/>
<span id="ISBN:0596007655"></span><br/>
<noscript><a href="http://books.google.com/books?vid=ISBN0596007655">Book Info at
Google Books</a></noscript>
```

HTML Explanation

Title Ambient Findability linked back to record in WorldCat. Span is a placeholder for possible link to preview or fulltext via GoogleBooks. This link will be added via Javascript if preview or fulltext is available.
noscript tag will create a link to book info at Google Books if Javascript is disabled.

Javascript

```
    <script>
     // Construct URL along with required ISBNs
     var isbns = [ISBN:0596007655];
     var api_url ="http://books.google.com/books?jscmd=viewapi&bibkeys=" +
isbns.join(",");
```

```
    // Talk to the server synchronously and get _GBSBookInfo object
    document.write(unescape("%3Cscript src=" + api_url +
      " type='text/javascript'%3E%3C/script%3E"));
  </script>

  <script>
    // Process response from Google booksearch
    for (i in isbns) {
      var element = document.getElementById(isbns[i]);
      var bookInfo = _GBSBookInfo[isbns[i]];
      // Check whether  server returned any data
       if (bookInfo) {
                  if (bookInfo.preview == "full" ||
                        bookInfo.preview == "partial") {

                    element.innerHTML ="<a href=\"" + bookInfo.preview_url + "\">Preview
at Google Books</a>";
                }
    }
    }

  </script>
```

Javascript Explanation

The first script takes the ISBNs and constructs a URL to retrieve a JSON object with information for those ISBNs from Google Books
The second script takes the information retrieved and creates links to Previews of the Books where a preview exists. Links are inserted into the blank span with the appropriate id (the one with the same ISBN) in the HTML.

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