Web Developer's Bootcamp: Tips, Code Samples, Explanations, and Downloads

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

Amanda Hollister
Systems Librarian
Broome Community College
hollisteraj@sunybroome.edu

Tips – Getting Started with Web Services

- Play in the sandbox – pick a service, study it
- Yahoo Query Language: http://developer.yahoo.com/yql/
- Yahoo Developer Central: http://developer.yahoo.com/
- Amazon Web Services Discussion Forums: https://forums.aws.amazon.com/index.jspa
- 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://arxiv.org/help/api/index
- 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
- iTunes and App Store API: http://www.apple.com/itunes/affiliates/resources/documentation/itunes-store-web-service-search-api.html

* See programmableweb for more: http://www.programmableweb.com/apis/directory

Resources and Tools

- New York Times API Tool http://prototype.nytimes.com/gst/apitool/index.html
- YouTube Data API scratchpad http://gdata.youtube.com/demo/index.html
- Google Code Playground http://code.google.com/apis/ajax/playground/
- Yahoo Pipes http://pipes.yahoo.com/pipes/
- Google Chart Wizard http://code.google.com/apis/chart/docs/chart_wizard.html
- Yahoo Query Language Console http://developer.yahoo.com/yql/console/
- Wiztools.org rest-client http://code.google.com/p/rest-client/

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

Using Flash

The following code snippet shows how to make a request to the Google Book Search API using Flash. This example uses JSON from the ActionScript 3.0 (AS3) Core Library.

```
var service:HTTPService = new HTTPService();
service.url =
'https://ajax.googleapis.com/ajax/services/search/books';
service.request.v = '1.0';
service.request.q = 'JavaScript';
service.request.key = 'INSERT-YOUR-KEY';
service.resultFormat = 'text';
service.addEventListener(ResultEvent.RESULT, onServerResponse);
service.send();

private function onServerResponse(event:ResultEvent):void {
  try {
    var json:Object = JSON.decode(event.result as String);
      // now have some fun with the results...
  } catch(ignored:Error) {
  }
}
```

Using Java

The following code snippet shows how to make a request to the Google Book Search API using Java. This example uses the JSON library from json.org.

```
URL url = new
URL("https://ajax.googleapis.com/ajax/services/search/books?" +
"v=1.0&q=barack%20obama&key=INSERT-YOUR-KEY&userip=INSERT-USER-IP");

URLConnection connection = url.openConnection();
```

```
connection.addRequestProperty("Referer", /* Enter the URL of your site
here */);

String line;
StringBuilder builder = new StringBuilder();
BufferedReader reader = new BufferedReader(new
InputStreamReader(connection.getInputStream()));
while((line = reader.readLine()) != null) {
 builder.append(line);
}

JSONObject json = new JSONObject(builder.toString());
// now have some fun with the results...
```

Using PHP

The following code snippet shows how to make a request to the Google Book Search API using PHP. This sample assumes PHP 5.2.

```
$url = "https://ajax.googleapis.com/ajax/services/search/books?" +
      "v=1.0&q=barack%20obama&key=INSERT-YOUR-KEY&userip=INSERT-USER-
IP";

// sendRequest
$ch = curl_init();
curl_setopt($ch, CURLOPT_URL, $url);
curl_setopt($ch, CURLOPT_RETURNTRANSFER, 1);
curl_setopt($ch, CURLOPT_REFERER, /* Enter the URL of your site here
*/);
$body = curl_exec($ch);
curl_close($ch);

// now, process the JSON string
$json = json_decode($body);
// now have some fun with the results...
```

Using Python

The following code snippet shows how to make a request to the Google Book Search API using Python. This sample assumes Python 2.4 or higher. You may need to download and install simplejson.

```
import urllib2
import simplejson

url = ('https://ajax.googleapis.com/ajax/services/search/books?' +
'v=1.0&q=barack%20obama&key=INSERT-YOUR-KEY&userip=INSERT-USER-IP')

request = urllib2.Request(url, None, {'Referer': /* Enter your site
URL */})
response = urllib2.urlopen(request)
```

```
# Process the JSON string.
results = simplejson.load(response)# now have some fun with the
results...
```

Using Perl

The following code snippet shows how to make a request to the Google Book Search API using Perl. This sample relies on the LWP::UserAgent and JSON modules which you can obtain from CPAN. You may also want to use the URI::Escape module.

```
#!/usr/bin/perl

my $url = "https://ajax.googleapis.com/ajax/services/search/books?" +
      "v=1.0&q=barack%20obama&key=INSERT-YOUR-KEY&userip=INSERT-USER-
IP";

# Load our modules
# Please note that you MUST have LWP::UserAgent and JSON installed to
use this
# You can get both from CPAN.
use LWP::UserAgent;
use JSON;

# Initialize the UserAgent object and send the request.
# Notice that referer is set manually to a URL string.
my $ua = LWP::UserAgent->new();
$ua->default_header("HTTP_REFERER" => /* Enter the URL of your site
here */);
my $body = $ua->get($url);

# process the json string
my $json = from_json($body->decoded_content);

# have some fun with the results
my $i = 0;
foreach my $result (@{$json->{responseData}->{results}}){
 $i++;
 print $i.". " . $result->{titleNoFormatting} . "(" . $result->{url} .
")\n";
 # etc....
}
if(!$i){
 print "Sorry, but there were no results.\n";
}
```

Web Services – Sample Applications

- Video Widget (YouTube, jQuery)
- Flickr API - Display Photos (JSON)
- Google Maps API - Local Libraries (XML)
- Aleph X-Server New Books display
- WorldCat Basic API

View samples and download code at http://www.lib.montana.edu/~jason/files.php and at http://librarydev.com/mashups

Code Sample #1: Video Widget (YouTube, jQuery) – Javascript and CSS

xHTML source:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head>
<meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
<title>mediaHub @ MSU</title>
<script type="text/javascript"
src="http://ajax.googleapis.com/ajax/libs/jquery/1.4.3/jquery.min.js">
</script>
<script src="quickpaginate.js" type="text/javascript"></script>
<script type="text/javascript">
$(document).ready(function(){
$('#content').hide();
$('#content').load('feed-youtube.php', function(){
      $('#content').fadeIn('slow');
$('.vids').quickpaginate({ perpage: 2, pager : $("#pagination") });
 });
});
</script>
<style type="text/css">
body {font-family:Arial, Helvetica, sans-serif;font-
size:12px;padding:0px;margin:0px;color:#333;}
#videos{width:309px;padding:12px 0px 0px 0px;background-
color:#fff;height:340px;}
.left{float:left;width:100px;margin-right:10px;}
.right{float:left;width:150px;}
.clear{clear:both;}
.vids{float:left;margin-right:2px;width:300px;padding:4px;}
.video-title{margin-bottom:5px;font-size:13px;}
.video-title a:hover{color:#666;}
.video-title a{color:#000;font-weight:700;padding:2px;text-
decoration:underline;}
.odd{background-color:#ccc;border-bottom:1px solid #999;border-top:1px
solid #999;}
.even{background-color:#dedede;}
#pagination{margin-bottom:5px;padding:5px;text-align:right;}
#pagination a{color:#333;text-decoration:underline;}
.qp_counter{margin-left:5px;margin-right:5px;color:#000;}
```

```
.nojs{background-color:#FFCC00;padding:20px;margin:10px;border:1px
solid #FF9900;}
#title{font-size:20px;margin-left:5px;margin-bottom:5px;}
</style>
</head>
…
```

xHTML and Javascript Explanation:

- Set up your HTML page
- Bring in javascript for behavior. In this case, pagination

Web page for user interface and display

```
...
<div id="videos">
<div id="title">mediaHub @ MSU</div>
  <div id="content">
     <div class="nojs" align="center">You need to enable JavaScript to
<br/>view the video feed</div>
  </div>
  <div class="clear"></div>
  <div id="pagination"></div>
</div>
...
```

Web page for user interface and display explanation

- HTML markup that gives programming hooks for our script
- Interface (GUI) controls use `<div id="pagination">`
- `<div id="videos">` will be populated with script messages OR generated xHTML tags received via our Ajax requests

PHP script to parse and display feed from YouTube API

```
//set default value for user, page will show this user's videos on
initial load
$user = isset($_GET['user']) ? $_GET['user'] : 'msulibrary';
//set default value for API version
$version = isset($_GET['v']) ? $_GET['v'] : '2';
//set default value for page length (number of entries to display);
pagination is an option - use start-index as parameter with limit
parameter
$limit = isset($_GET['limit']) ? $_GET['limit'] : '25';
//set default value for output format (atom is default, options
include atom, rss, json and json-in-script
$format = isset($_GET['format']) ? $_GET['format'] : 'atom';
//set feed URL
$feed =
'http://gdata.youtube.com/feeds/api/users/'.$user.'/uploads?max-
results='.$limit.'&amp;alt='.$format.'&amp;v='.$version.'';
…
```

```
$data = simplexml_load_file($feed);

//parse returned data elements from api call and display as html
function parseRSS($xml) {
      foreach ($xml->entry as $entry) {
      $title = htmlentities($entry->title);
      $creator = $entry->author->name;
      //get unique video id
      $arr = explode('/',$entry->id);
      $id = $arr[count($arr)-1];

…
      if ($title) {
      if($odd = $i%2){$class='odd';}else{$class='even';}
            echo "<div class='vids ".$class."'>";
            echo '<div class="left"><a class="ms-video" target="_top"
href="'.$watch.'"><img width="100" border="0" src="'.$thumbnail.'"
alt="'.$title.'" /></a></div>';
            echo '<div class="right"><div class="video-title"><a
class="ms-video" target="_top"
href="'.$watch.'"><span>'.$title.'</span></a></div>';
            echo '<div class="video-
description">'.$description.'</div>';
            echo  $duration.' mins</div>';
            echo '<div class="clear"></div>';
            echo "</div>";
      }

      $i++;
      //end foreach
      }
}
```

PHP script to parse and display feed from YouTube API

- Server-side script that processes data from YouTube API response
- Format URL request to API using `$feed` variable
- Loop through data returned and place values in HTML

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

The URL Request

http://api.flickr.com/services/feeds/photos_public.gne?tags=il2011&format=json

URL Request: 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

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=cil20
08&format=json">
</script>
```

The URL Request in Javascript - 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

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"
},
…
]
})
```

JSON Response - Explanation

- More structured data ready to be parsed
- We'll extract the values and format for display using the second javascript

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) {
          var container = document.getElementById("feed");
          var markup = '<h1>' + '<a href="' + fr.link+ '">' +
fr.title + '</a>'+ '</h1>';
          for (var i = 0; i < fr.items.length; i++) {
               markup += '<a title="' + fr.items[i].title + '"
href="' + fr.items[i].link + '"><img src="' + fr.items[i].media.m + '"
alt="' + fr.items[i].title + '"></a>';
          }
          container.innerHTML = markup;
     }
</script>
```

Parse and display with Javascript - 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

"container.innerHTML" – native javascript function prints out values from JSON in xHTML markup

Code Sample #3: Google Maps API

Local Libraries (XML) - HTML Markup

```
<div id="map" style="width:75%; height:500px"></div>
```

HTML Explanation:

- Create HTML container where Javascript will load dynamic content
- Add some inline CSS styles to control size of map object

XML File

```
<markers>
 <marker name="St. Johns Co Public Library" address="1960 North Ponce
de Leon Blvd. St. Augustine, Florida 32084"
url="http://www.sjcpls.org/" lat="29.1869" lng="-82.1372" />
 <marker name="Branford Public Library" address="703 Suwannee Ave NW,
Branford, FL 32008-3279" url="http://www.neflin.org/srrl/"
lat="29.963245" lng="-82.93090" />
...
</markers>
```

XML Explanation

- Create XML file with metadata and geographic coordinates; file is named "markers.xml".
- Javascript will use this XML file as instructions for mapping locations

Javascript

```
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script type="text/javascript">
//<![CDATA[
function load() {
  var map = new google.maps.Map(document.getElementById("map"), {
center: new google.maps.LatLng(29.1869, -82.1372),
zoom: 7,
mapTypeId: google.maps.MapTypeId.TERRAIN,
mapTypeControl: true,
mapTypeControlOptions: {style:
google.maps.MapTypeControlStyle.DROPDOWN_MENU},
navigationControl: true,
navigationControlOptions: {style:
google.maps.NavigationControlStyle.SMALL}
  });
  var infoWindow = new google.maps.InfoWindow;
  // Change this depending on the name of your xml file
  downloadUrl("markers.xml", function(data) {
var xml = data.responseXML;
var markers = xml.documentElement.getElementsByTagName("marker");
for (var i = 0; i < markers.length; i++) {
 var name = markers[i].getAttribute("name");
 var address = markers[i].getAttribute("address");
 var url = markers[i].getAttribute("url");
 var point = new google.maps.LatLng(
 parseFloat(markers[i].getAttribute("lat")),
 parseFloat(markers[i].getAttribute("lng")));
 var html = '<p style="height:50px;"><strong><a
href="'+url+'">'+name+'</a></strong><br/>'+address+'</p>';
 var marker = new google.maps.Marker({
map: map,
position: point,
 });
 bindInfoWindow(marker, map, infoWindow, html);
}
  });
}
function bindInfoWindow(marker, map, infoWindow, html) {
  google.maps.event.addListener(marker, 'click', function() {
infoWindow.setContent(html);
infoWindow.open(map, marker);
  });
}
function downloadUrl(url, callback) {
  var request = window.ActiveXObject ?
```

```
 new ActiveXObject('Microsoft.XMLHTTP') :
 new XMLHttpRequest;
  request.onreadystatechange = function() {
if (request.readyState == 4) {
 request.onreadystatechange = doNothing;
 callback(request, request.status);
}
  };
  request.open('GET', url, true);
  request.send(null);
}
function doNothing() {}
//]]>
</script>
```

Javascript Explanation

- Bring in Google Maps API
- Create Google Maps Object
- Download XML file ("markers.xml"); Parse values to create markers
- Create information windows for data from XML file - These are the Google Maps "pop-up" windows

Code Sample #4: Aleph X-Server New Book Display

```
<script type="text/javascript" src="js/google_books.js"></script>


<?php
// Aleph server variables
$domain = "seneca.sunyconnect.suny.edu";
$port = "4620";
$base = "BCC01PUB";

$item_link_base = "http://".$domain.":".$port."/F?func=find-
c&local_base=".$base."&ccl_term=sys%3D"; // search results based on
system number query

// set number of books to show
$request = "(wcl=NEWBK)";
$marc_005_string = "";

// create find request url
// put xml response into a string
$find_url =
"http://".$domain.":".$port."/X?op=find&base=".$base."&request=".urlen
code($request).$marc_005_string;
$find_xml = implode('', file($find_url));
#print $find_url."<br/>\n";
#print $find_xml;
```

```
// regex to grab xml data
preg_match("/\<set\_number\>(.*?)\<\/set\_number\>/", $find_xml,
$set);
preg_match("/\<no\_records\>(.*?)\<\/no\_records\>/", $find_xml,
$no_records);
preg_match("/\<no\_entries\>(.*?)\<\/no\_entries\>/", $find_xml,
$no_entries);
preg_match("/\<session\-id\>(.*?)\<\/session\-id\>/", $find_xml,
$session);

// create the entry numbers string
// limit the number of entries retrieved to 12
if(intval($no_records[1]) > 08){
$entry_numbers = "000000001-000000008";
}
else{
$entry_numbers = "000000001-".sprintf("%09d",intval($no_records[1]));
}

// assemble the parts of the present request

$present_url =
"http://".$domain.":".$port."/X?op=present&set_no=".$set[1]."&set_entr
y=".$entry_numbers."&format=marc";
//print $present_url;
```

Aleph Explanation:

- Send a "find" query to the x-server to pull the set number, and number of items
- Send a "present" query to the x-server to pull the marc records

```
// scan the xml and place records into an array
$xml = simplexml_load_file($present_url);
//echo htmlspecialchars($xml->asXML());


foreach ($xml->xpath('//record') as $record){

          $title_result = $record-
>xpath("metadata/oai_marc/varfield[@id='245']/subfield[@label='a']");

          $subtitle_result = $record-
>xpath("metadata/oai_marc/varfield[@id='245']/subfield[@label='b']");
          $title = trim((empty($title_result[0]) ? '':
rtrim((string)$title_result[0],"/")). ' ' .(empty($subtitle_result[0])
? '' :  rtrim((string)$subtitle_result[0],"/")));

          $publisher = $record-
>xpath("metadata/oai_marc/varfield[@id='260']/subfield[@label='b']");
          $publication_date = $record-
>xpath("metadata/oai_marc/varfield[@id='260']/subfield[@label='c']");
```

```
            $isbn = $record-
>xpath("metadata/oai_marc/varfield[@id='020']/subfield[@label='a']");
                        if (strpos($isbn[0], " ") > 0) {
                                    $isbn_1 = substr($isbn[0], 0,
strpos($isbn[0], " "));
                        } else {
                                    $isbn_1 = $isbn[0];
                        }
            $author = $record-
>xpath("metadata/oai_marc/varfield[@id='100']/subfield[@label='a']");
            $oclcnumber_result = $record-
>xpath("metadata/oai_marc/varfield[@id='035']/subfield[@label='a']");
            $oclcnumber = ltrim($oclcnumber_result[0],"(OCoLC)");
            $google_books_bibkeys[] = "ISBN:".$isbn_1;
```

Explanation:

- Specifies which XML element holds the records
- Loops through the results and extracts the title, author, ISBN, and OCLC number for each item
- Creates an array of ISBNs to send to the Google Books API

Create the HTML for record display:

```
if (strlen($isbn_1) > 1) {
?>
<!-- start book -->

<div class="item_display">

            <div style="text-
align:center;width:160px;height:120px;overflow:hidden;">
                        <div class="google_books_cover"
style="display:none;" id="google_books_cover_ISBN:<?php echo $isbn_1;
?>" bibkey="<?php echo $isbn_1; ?>;">
                                    <!-- start Google Books cover -->
                                    <a id="google_books_preview_ISBN:<?php
echo $isbn_1; ?>" target="_blank"><img class="google_books_thumbnail"
id="google_books_thumbnail_ISBN:<?php echo $isbn_1; ?>" src=""
style="border: 0pt none;" alt="Google Books cover art" /></a>
                                    <!-- end Google Books cover -->
                                    <div
id="google_books_preview_icon_ISBN:<?php echo $isbn_1; ?>"></div>
                        </div>
            </div>

            <div style="text-align:left;font-
size:85%;width:160px;height:70px;overflow:hidden;">
                        <a
href="http://bcc.sunyconnect.suny.edu:4620/F/?func=find-
a&amp;find_code=020&amp;request=<?php echo $isbn_1; ?>"><span><?php
echo $title; ?></span></a>
            </div>
```

```
            <br clear="all"/>
</div>

<!-- end book -->
<?php
}}
?>

<!-- Send request to Googlebooksearch server -->
<script
src="http://books.google.com/books?jscmd=viewapi&amp;bibkeys=<?php
echo implode(",",$google_books_bibkeys);
?>&amp;callback=ProcessGBSBookInfo"></script>
```

Explanation:

- Creates the div tags for Google Books to insert the cover and preview icon
- Creates a link to the item in the Aleph catalog

Code Sample #5: WorldCat Search API + LibraryThing Reviews + Google Book Preview

HTML Form:

```
<h2>Search for:</h2>
<form action="worldcat_basicsearch_mashup.php" method="get">
<p>
    <label for="AddWorldCatSearch-SearchString">Search </label>
    <input type="text" id="AddWorldCatSearch-SearchString"
name="AddWorldCatSearch-SearchString" value="" />
</p>
<input type="submit" value="Search"/>
</form>
```

Explanation:

- Provides a search box
- Form passes search string as a variable

URL Request:

http://worldcat.org/webservices/catalog/search/opensearch?format=rss&q=javascript&maximumRecords=10&wskey=yourAPIkey

Explanation:

- Sends request to the WorldCat Basic API service
- Sets the request protocol as Opensearch (http://www.opensearch.org)
- Sets the response format to RSS

PHP used to build the URL request:

```
if ( strlen($_REQUEST['AddWorldCatSearch-SearchString']) > 0) {

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

$searchURL = $searchURL .  '&maximumRecords=10';
$searchURL .= '&wskey='. $WorldCatAPIKey;

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

Explanation:

- Checks to see if a search string was passed as a variable
- Adds the search query, the number of records, and your API key
- Sends request, loads the XML results as an object

PHP used to fetch the ISBNs

```
$lt_ids = "";
$isbns = $xml->xpath("//dc:identifier");

foreach ((array)$isbns as $isbn) {

      if (strlen($isbn) > 1) {
            if (strpos($isbn[0], " ") > 0) {
                  $lt_ids = $lt_ids . substr($isbn, 0, strpos($isbn, "
"));
            } else {
                  $lt_ids = $lt_ids = $lt_ids . ltrim($isbn[0],
"urn:ISBN:" );
            }
            if ($isbn != end($isbns)) {
                  $lt_ids = $lt_ids . ',';
            }
      }
}
```

Explanation:

- Specifies what element in the XML object contains the ISBN
- Sets up an array of the ISBNs from the records
- Trims the extra characters “urn:ISBN:” from each ISBN

Parse and display with PHP

```
foreach($xml->xpath('//item') as $book ) {
      $book['xmlns:dc'] = 'http://purl.org/dc/elements/1.1/';
      $book['xmlns:oclcterms'] = 'http://purl.org/oclc/terms/';
      $field = simplexml_load_string($book->asXML());
```

```
      $title = $field->xpath("title");
      $isbn = $field->xpath("dc:identifier");
      $isbn_1 = ltrim($isbn[0], "urn:ISBN:" );
      $author = $field->author->name;
      $oclcnumber = $field->xpath("oclcterms:recordIdentifier");

      echo '<div class="record">';
      if (strlen($isbn_1) > 0) {

// check Open Library for a cover
            $image_url = 'http://covers.openlibrary.org/b/isbn/' .
$isbn_1 . '-S.jpg';
            $image_size = getimagesize($image_url);
            if ($image_size[0] > 1 and $image_size[1] > 1) {
                  echo '<img src="' . $image_url . '"/>';
            } else {

// check LibraryThing for a cover
                  $image_url = 'http://covers.librarything.com/devkey/'
. $librarything_key . '/small/isbn/' . $isbn_1;
                  $image_size = getimagesize($image_url);
                  if ($image_size[0] > 1 and $image_size[1] > 1) {
                        echo '<img src="' . $image_url . '"/>';
                  }
            }
      }
```

Explanation:

- Specifies which XML element holds the records
- Loops through the results and extracts the title, author, ISBN, and OCLC number for each item
- Checks OpenLibrary and LibraryThing for a cover; if available, displays it

Create the HTML for record display:

```
      echo '<p><a href="http://www.worldcat.org/oclc/' . $oclcnumber[0]
. '"><span>' . $title[0] . '</span></a></p>';
      if ( strlen($isbn_1) > 1 ) {
            echo '<br/>';
            echo '<span id="LT_' . $isbn_1 . '"></span><br/>';
            echo '<noscript><a href="http://www.librarything.com/isbn/'
. $isbn_1 . '">View Book Information at LibraryThing</a></noscript>';
            echo '<br/>';
            echo '<script type="text/javascript">';
            echo 'GBS_insertPreviewButtonPopup("ISBN:' . $isbn_1 .'")';
            echo '</script>';
            echo '<noscript><a
href="http://books.google.com/books?vid=ISBN' . $isbn_1 . '">Book Info
at Google Books</a></noscript>';

      }
```

```
      echo '</p>';
      echo '<br clear="all"/>';
      echo '</div>';
}
```

Explanation:

- Uses the OCLC number to create a link to the WorldCat record
- Creates a placeholder for the LibraryThing rating
- Adds the link to the Google Books preview of available
- Creates <noscript></noscript> messages in case the user has javascript turned off

Javascript to fetch the ratings from LibraryThing:

```
<script type="text/javascript">
      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 src="' + book.rating_img +
'"/>' ;
                              $('#LT_' + book.id).append('<a href="' +
book.link + '">' + desc + '@ LibraryThing</a>' + ratimg );
                        }
                  }
            }
      }
</script>

<script type="text/javascript"
src='http://www.librarything.com/api/json/workinfo.js?ids=<?php echo
$lt_ids ?>&callback=LTpop'></script>
```

Explanation:

- The first part is the function that checks the retrieved records and displays the rating
- The second part is the URL query sent to LibraryThing to retrieve the book records as JSON
- If the item has a rating, the rating is displayed in the html placeholder

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