Using the OCLC WorldCat APIs

by by Mark A. Matienzo

Mashups combine information from several web services into a single web user experience — but what do they look like in practice? Here are the gritty details of connecting an industry leading online catalog with Google maps, and how Python made it easy.

Like other communities, the library world has begun providing APIs to allow developers to reuse data in various ways. OCLC, the world's largest library consortium, provides several APIs for developers, including APIs to search and retrieve bibliographic records and data about their participating institutions. This article describes the available APIs provided by OCLC and a sample application to map library holdings on a Google Map using `worldcat`, a Python module written to work with these APIs.

About OCLC, WorldCat, and WorldCat.org

The Online Computer Library Center, Inc. (OCLC), is an international, non-profit cooperative that provides a variety of services to libraries, archives and museums, and engages in research and programmatic work for them. OCLC was founded in 1967 as the Ohio College Library Center as an organization dedicated to developing and supporting a computerized regional library network for Ohio's academic libraries. In 1971, OCLC's bibliographic database, the Online Union Catalog, began operation, and in 1981, the organization changed its name to OCLC Online Computer Library Center, Inc. From 1991 to 1996, OCLC provided access to the Online Union Catalog on a subscription basis for research use under the name WorldCat. In 1997,

REQUIREMENTS

PYTHON: *2.4 - 2.7*

Useful/Related Links:

- WorldCat Facts and Statistics - *http://www.oclc.org/us/en/worldcat/statistics/default.htm*
- "Extending the OCLC cooperative" - *http://web.archive.org/web/20010615111317/www.oclc.org/strategy/strategy_document.pdf*
- OCLC Office of Research - *http://www.oclc.org/research/projects/frbr/algorithm.htm*
- Requesting a WorldCat API key - *http://worldcat.org/devnet/wiki/SearchAPIWhoCanUse*
- SIMILE Exhibit API - *http://simile-widgets.org/exhibit/*
- Babel for SIMILE - *http://simile.mit.edu/wiki/Babel*
- Exhibit's JSON format - *http://simile.mit.edu/wiki/Exhibit/Creating,_Importing,_and_Managing_Data*
- WorldCat Identities - *http://worldcat.org/identities/*
- Terminology Services - *http://tspilot.oclc.org/resources/*
- Metadata crosswalk service - *http://www.oclc.org/research/researchworks/xwalk/*
- Third OCLC Research Software Contest - *http://www.oclc.org/research/researchworks/contest/*

OCLC began referring to the Online Union Catalog in all forms as WorldCat.

WorldCat currently contains over 135 million bibliographic records, representing nearly 1.5 billion items held by over 71,000 libraries (see their Facts and Statistics Page for more details). Originally, WorldCat records were available exclusively on a subscription basis for OCLC member organizations. Beginning in 2000, however, OCLC began a three-year project to investigate the expansion of WorldCat into a "globally networked, and globally available information resource" (see their report "Extending the OCLC cooperative: A three-year strategy.") This project eventually became known as Open WorldCat. In 2001, OCLC began developing a prototype of a public web-based system to perform simple queries and that would provide holdings information for the nearest holding libraries if users also entered geographic information. Between 2001 and 2002, OCLC also began partnering with online booksellers such as Abebooks and Alibris to enable users to search WorldCat records automatically when their searches on the booksellers' sites returned no results.

OCLC signed an agreement with Google in 2003 to release a 2-million-record subset of its bibliographic data to provide similar functionality to the earlier pilot partnership with online booksellers. In 2004, Open WorldCat moved out of its pilot status and went live, with OCLC signing similar agreements with Yahoo!, Ask.com, and Microsoft. In 2005, OCLC began incorporating records for electronic journals and restricted participation in Open WorldCat to institutions that subscribed to its FirstSearch service. OCLC launched a beta version WorldCat.org in 2006, which let users search the entire WorldCat database from a freely available web interface for the first time.

The WorldCat Affiliate APIs

OCLC began making web services available to developers in early 2007. OCLC's first two offerings were the WorldCat Registry, which provided search and retrieval of information about OCLC member institutions, and xISBN, which retrieves ISBNs and other bibliographic metadata for related versions of an individual work held in the WorldCat database. In mid-2007, OCLC added the OpenURL Gateway, which assists Web-based applications in routing end users to full-text articles and other online resources provided by libraries. OCLC added xISSN in October 2007, which provides similar functionality to xISBN but for serial publications (e.g., journals and magazines). OCLC's most recent addition, as of August 2008, is the WorldCat Search API, which allows developers to build applications that can programmatically search and retrieve bibliographic metadata and holdings information from the entire WorldCat database.

xISBN and xISSN are two of OCLC's Identifier Services (also referred to as the "xID" or "xIdentifier" services). OCLC also provides the xOCLCNUM subservice under the banner of xISBN, which provides similar functionality to works identified by either their OCLC record number or their Library of Congress Catalog Number. All of the Identifier services are free for non-commercial use by all developers when usage does not exceed 500 requests per day. Additionally, all OCLC members with a cataloging subscription may use the services for free without restriction, and OCLC accommodates commercial use or non-commercial usage beyond 500 requests per day on a subscription basis.

The Identifier APIs rely on an algorithm developed by the OCLC Office of Research to group bibliographic records using the conceptual model for Group 1 entities specified in the Functional Requirements for Bibliographic Records. To use the service, a user submits an identifier as a URL parameter to the appropriate web service along with parameters specifying the appropriate method for the type of request to make and the desired format of the response. Responses can be returned as XML, HTML, CSV, or TSV, as well as dictionaries/associative arrays in JSON, Python, Ruby, or PHP syntax.

All the Identifier services have two common methods: getEditions, which retrieves a list of relevant identifiers and metadata for editions of a work specified a given identifier, and getMetadata, which retrieves metadata about item specified by the given identifier. xISBN and xISSN also provide fixChecksum, which regenerates the appropriate check digit for the specified identifier scheme. xISSN provides a few additional methods including getForms, which provides information of the physical form of the serial specified by the identifier, and getHistory, which provides information about preceding and succeeding serials. xOCLCNUM also makes the getVariants method available, which returns various identifier formats for a given OCLC record number or Library of Congress Catalog number.

For example, to construct an xOCLCNUM request for OCLC record number 34745932, using the getEditions method, and returning the results as a Python dictionary, we would issue an HTTP GET to the following URL:

```
http://xisbn.worldcat.org/webservices/xid/oclcnum
/34745932?method=getEditions&format=python
```

In response, we would get the following dictionary back from xOCLCNUM:

```
{'stat':'ok',
 'list':[
  {'oclcnum':['34745932']},
  {'oclcnum':['50174603']}]}
```

The WorldCat Search API and Registry

The WorldCat Search API is, more accurately, a set of APIs that allow developers to search and retrieve bibliographic and holdings data from the entire WorldCat database. As of the time of writing, OCLC is currently making the WorldCat Search API available only to developers affiliated with member institutions with a cataloging subscription. Developers can request an OCLC Web Services key ("wskey") from OCLC through the WorldCat web site. The Search API allows applications to search the WorldCat database programmatically by submitting queries in formatted as either OpenSearch or SRU (Search/Retrieve via URL) and CQL (Contextual Query Language).

Instead of requesting a key for yourself, you can also use the temporary key provided in the source code bundle that goes with this issue of the magazine. Visit the URL given on the title page of this issue of *Python Magazine* beneath the worlds "Download this month's code", and look inside of the directory of this article for a copy of `holdingsmap.py` (the demo program we will investigate below) that already includes a working key.

If you use the *Python Magazine* key when connecting, then please limit yourself to only a few queries per session! The key is limited to only about three hundred queries per day, and if you use more than your share (maybe just two or three, to demonstrate to yourself that the demo indeed works?) then other readers will not be able to run the demo.

> When my employer first gained access to the APIs provided by OCLC, I chose to write a Python module to make development with the APIs easier.

For the sake of brevity, this article will not discuss the syntax of SRU URL parameters and CQL queries. Detailed information on SRU and CQL as well as the various search indexes available for SRU queries can be found on the WorldCat Developers' Network wiki. OpenSearch queries return either responses in Atom or RSS, while SRU queries return results as SRU result sets containing either MARCXML or Dublin Core XML records. In addition, the WorldCat Registry can be searched using SRU and CQL, while individual records can be retrieved using a RESTful URL pattern. Individual records can be returned as formatted HTML or as XML for parsing.

In addition, the WorldCat Search API also provides a

LISTING 1

```
<searchRetrieveResponse ...>
 <version>1.1</version>
 <numberOfRecords>5</numberOfRecords>
 <records>
  ...
  <record>
   <recordSchema>info:srw/schema/1/dc</recordSchema>
   <recordPacking>xml</recordPacking>
   <recordData>
    <oclcdcs>
     <dc:creator>Sangameswaran, Ramaswamy P.
       (Ramaswamy Pulikoot Madhom)</dc:creator>
     <dc:date>1994</dc:date>
     <dc:description>"December 1994."</dc:description>
     <dc:description>Thesis (M.S.)--University of Arkansas,
       Fayetteville, 1994.</dc:description>
     <dc:format>ix, 60 leaves : ill. ; 28 cm.</dc:format>
     <dc:language ...>eng</dc:language>
     <dc:subject ...>Gases--Separation.</dc:subject>
     <dc:subject ...>Gas separation membranes.</dc:subject>
     <dc:subject ...>Polymers--Permeability.</dc:subject>
     <dc:title>Penetrant induced effects of polyimides for gas
       separation membranes </dc:title>
     ...
    </oclcdcs>
   </recordData>
  </record>
 </records>
 ...
</searchRetrieveResponse>
```

LISTING 2

```
<holdings ...>
 <holding>
  <institutionIdentifier><value>AFU</value>...</institutionIdentifier>
  <physicalLocation>University of Arkansas - Fayetteville
  </physicalLocation>
  <physicalAddress>
   <text>Fayetteville, AR 72701 United States</text>
  </physicalAddress>
  <electronicAddress>
   <text>
    http://www.worldcat.org/wcpa/oclc/32597190?page=frame&amp;url=http
    %3A%2F%2Flibrary.uark.edu%2Fsearch%2Fo32597190%26checksum%3D8df95d
    7d1a6a06656475b032a30f69a3&amp;title=University+of+Arkansas+-+Faye
    tteville&amp;linktype=opac&amp;detail=AFU%3AUniversity+of+Arkansas
    +-+Fayetteville%3AAcademic&amp;app=wcapi&amp;id=foobar
   </text>
  </electronicAddress>
  <holdingSimple>
   <copiesSummary><copiesCount>1</copiesCount></copiesSummary>
  </holdingSimple>
 </holding>
</holdings>
```

handler for "content requests" for bibliographic data, holdings information, and formatted citations for individual records specified by OCLC record number or ISBN. For holdings information, the content request must include some sort of geographically identifying information, such as a name or code for a state, province, or country, a postal code, latitude and longitude, or an IP address. Responses for holdings information are returned as an XML document conforming to the ISO 20775 schema for holdings data. Citation requests include an optional citation parameter, specifying the format of the citation, with APA, Chicago, Harvard, MLA, and Turabian formats available upon specification.

Constructing an example SRU request for the WorldCat Search API requires us to specify the search indexes we need for the query. OCLC provides information on the available indexes and a JavaScript-based URL constructor for SRU requests, located at:

```
http://worldcat.org/webservices/catalog
 /evaluator.html
```

For example, let's construct an SRU request for works in WorldCat with the words "Polyimides" in the title and containing the keyword phrase "gas separation." Let's also say the value wskey was "foobar", and that we wanted our result set to return records in Dublin Core. With these parameters, the request URL for this query would be:

```
http://worldcat.org/webservices/catalog/search/sru
?query=srw.ti%3dpolyimides%20and%20srw.kw%3d%22gas
%20separation%22&wskey=foobar&recordSchema=info%3A
srw%2Fschema%2F1%2Fdc
```

Issuing an HTTP GET request would then return results similar to those shown in Listing 1.

To get the holding information for this particular work, we would then generate a holdings request for the OCLC record number specified in the bibliographic data, 32597190. Accordingly, we would send an HTTP GET request to:

```
http://worldcat.org/webservices/catalog/content
/libraries/32597190?&wskey=foobar
```

which would return the response shown in Listing 2. In this response, we can see basic information about the institution, including its OCLC symbol and general address information. To retrieve detailed information about the institution in XML, we must therefore submit a request to the RESTful interface to the WorldCat Registry, passing the OCLC symbol ("AFU") as part of the URL request:

```
http://www.worldcat.org/webservices/registry/lookup
/Institutions/oclcSymbol/AFU?serviceLabel=content
```

The response, which I will not bother to include, contains detailed contact information, branch information, catalog links, and, if available, administrative statistics for the specified institution.

worldcat: A Python Module for OCLC APIs

When my employer first gained access to the APIs provided by OCLC, I chose to write a Python module to make development with the APIs easier. This was in part motivated by the fact that OCLC's Identifier APIs could return Python dictionaries as responses!

I made the initial public release of the worldcat module in August 2008, shortly after OCLC made the WorldCat Search API available to the public. worldcat is dual-licensed under the GNU General Public License version 3 and the Modified (three-clause) BSD License. worldcat has few dependencies, only requiring the pymarc module for some of the helper functions for processing data returned in MARCXML from the WorldCat Search API. However, having an ElementTree implementation installed (such as the one that comes with the Python Standard Library in Python 2.5 and higher) is strongly encouraged, as much of the data from OCLC's APIs is returned in highly structured XML.

worldcat provides a number of different classes to construct requests to the OCLC APIs. All of the request classes, SearchAPIRequest, xIDRequest, and RegistryRequest, are subclassed from a WorldCatRequest. Each of these subclasses is further subclassed to the specific types of requests that the APIs can handle. For example, the SearchAPIRequest has SRURequest, OpenSearchRequest, and ContentRequest subclasses, with the latter further subclassed to handle the different types of content requests. The WorldCatRequest class defines a number of methods, some of which are overridden by the individual subclasses. The two methods most developers

LISTING 3

```
$ wc/bin/python
>>> from worldcat.request.xid import xOCLCNUMRequest
>>> from pprint import PrettyPrinter
>>> pp = PrettyPrinter(indent=2)
>>> o = xOCLCNUMRequest(rec_num='34745932')
>>> o.validate()
>>> r = o.get_response()
>>> pp.pprint(r.__dict__)
{ 'data': { 'list': [ { 'isbn': ['9780824794668'],
                        'lccn': ['96018710'],
                        'oclcnum': ['34745932']},
                      { 'isbn': ['9780585399645'],
                        'oclcnum': ['50174603']}],
            'stat': 'ok'},
  'eval': True,
  'method': 'getEditions',
  'response_format': 'python',
  'response_type': 'xOCLCNUMRequest'}
```

will want to use are the validate() method and the get_response() method; the latter returns the appropriate subclass of a WorldCatResponse object.

Submitting Requests

To work through the following examples, we will install virtualenv and create a virtual environment for worldcat and the all other modules we will need for the examples. We'll also grab a copy of the latest copy sources for a later example using Mercurial, a distributed version control system. This walk-through assumes that you are using Python 2.6. To get started, run the following:

```
$ easy_install virtualenv
$ virtualenv wc
$ wc/bin/easy_install mercurial
$ wc/bin/easy_install pymarc
$ wc/bin/easy_install worldcat
$ wc/bin/hg clone \
      http://bitbucket.org/anarchivist/worldcat \
      wc/worldcat
```

We will first recreate the xOCLCNUM request that we issued earlier. By default, the xID requests in worldcat return the response as a Python dictionary that is then parsed using a "safe eval" function. Additionally, the default API method called for worldcat xID requests is the getEditions method. Any of this behavior may be overridden when you construct a request.

To construct the request, we'll start Python from within our virtual environment. Then we must import the xOCLCNUM request, pass the appropriate arguments, and run some basic validation on the request. Next, we submit the request to the xOCLCNUM service, and pretty-print the response object. Take a look at Listing 3 to see what the result looks like.

Let's also construct an SRURequest based upon the SRU query that we sent to the APIs earlier. Like with the xOCLCNUM request, we must import the appropriate subclass. Since the WorldCat Search API requires authentication, we must construct our request with our wskey. Next, we will run some basic validation on the request, make the request, and examine the results. Unlike requests to the xID services, worldcat does not automatically parse the actual response data in SearchAPIResponse objects. Instead, it returns the XML response as a string.

worldcat also provides a utility function called extract_elements() to parse responses using ElementTree if desired. Take a look at Listing 4 to see how these tools can be made to work together to retrieve a WorldCat result and then parse its XML.

LISTING 4

```
>>> from worldcat.request.search import SRURequest
>>> pp = PrettyPrinter(indent=2)
>>> s = SRURequest(wskey="foobar")
>>> s.args['query']='srw.ti=polyimides and srw.kw="gas separation"'
>>> s.args['recordSchema']='info:srw/schema/1/dc'
>>> s.validate()
>>> r = s.get_response()
>>> pp.pprint(r.__dict__)
{ 'data': ' (big block of XML, like that in Listing 1) ',
  'eval': False,
  'record_format': 'dc',
  'response_format': 'xml',
  'response_type': 'SRURequest'}
>>> from worldcat.util.extract import extract_elements
>>> x = extract_elements(r.data,
...     "{http://purl.org/dc/elements/1.1/}title")
>>> pp.pprint(x)
[ <Element {http://purl.org/dc/elements/1.1/}title at f230d0>,
  <Element {http://purl.org/dc/elements/1.1/}title at f233a0>,
  <Element {http://purl.org/dc/elements/1.1/}title at f239e0>,
  <Element {http://purl.org/dc/elements/1.1/}title at f23e40>,
  <Element {http://purl.org/dc/elements/1.1/}title at 1951418>]
>>> x[4].text
'Penetrant induced effects of polyimides for gas separation membranes '
```

Given that we've received the same record set as the earlier example, we can also construct requests for holdings information using the same record that we specified before. Again, you can either fill in a wskey that you yourself apply for, or use the key you'll find in the source code bundle on the *Python Magazine* web site for this article (in which case, please be gentle with it, and limit yourself to a few uses per day):

```
>>> from worldcat.request.search \
...     import LibrariesRequest
>>> l = LibrariesRequest(wskey=\"...\",
...                      rec_num=\"32597190\")
>>> l.validate()
>>> r = l.get_response()
>>> pp.pprint(r.__dict__)
{ 'data': ' (big block of XML, like in Listing 2) ',
  'eval': False,
  'record_format': 'iso20775',
  'response_format': 'xml',
  'response_type': 'LibrariesRequest'}
```

Finally, let's submit a request for the registry information for the institution with holdings for this item:

```
>>> from worldcat.request.registry \
...     import OCLCSymbolRequest
>>> o = OCLCSymbolRequest(symbol=\"AFU\")
>>> r = o.get_response()
```

If you examine the XML that this leaves in r.data, you will see a standard WorldCat institution description ready to be parsed and have useful information extracted by a Python program.

Putting it Together, Making a Map

Now that we've covered the basics of generating requests and receiving requests from OCLC's APIs using worldcat, we can start building complete applications to reuse data from the available APIs. worldcat is distributed with two sample applications, and the rest of this article will discuss the implementation of one of them. The application, holdingsmap.py, is a web application that allows you to find holding libraries for variants of a work, given its OCLC record number and a United States ZIP code. Beyond OCLC's APIs, the application uses two additional APIs.

First, it uses the Google Maps API to geocode and, indirectly, to render holding location information for a given work.

Second, it uses the SIMILE Exhibit API to parse and plot the data gathered from the OCLC APIs into a form the Google Maps API can consume. While the user's browser initially renders the page containing an "exhibit" with a map-based view, the browser also loads a JSON object with a particular structure used by Exhibit. Exhibit's developers also provide a hosted data translation service called Babel to assist in the translation from other data formats to Exhibit JSON. Exhibit JSON objects are then passed to Exhibit's JavaScript API, which parses the object into memory as an Exhibit database. The Exhibit API calls an externally hosted "painter service" to generate the map markers on the fly. This painter service, along with Exhibit API's map extension, act as an abstraction layer on top of the Google Maps API, allowing developers to focus on preparing their data for display. Exhibit's behavior therefore makes it well suited for prototyping interfaces for visualizing data using maps. In addition, since Exhibit's API is written in JavaScript, developers can extend or override behavior fairly easily.

In addition to worldcat, this application relies on a few additional third-party modules for its functionality: web.py, a lightweight, Pythonic web application framework, and geopy, a Python module for geocoding. In addition, if you not using Python 2.6 or later, you will need to install simplejson, a JSON parsing module, and processing, a module that provides process spawning using an API similar to the threading module included in the standard library.

To get started, we will install the additional modules in our virtual environment:

```
$ wc/bin/easy_install web
$ wc/bin/easy_install geopy
```

Note that, as of the time this magazine is going to press, the command above to install geopy fails because the host name exogen.case.edu is failing to resolve. In such an emergency, you can also try directly installing the current development version of geopy which is hosted somewhere a bit more robust:

```
$ easy_install \
  http://geopy.googlecode.com/svn/tags/release-0.93/
```

To use holdingsmap.py, we will need to modify the script to store our OCLC wskey and our Google Maps API key. To make these changes, edit lines 45 and 46

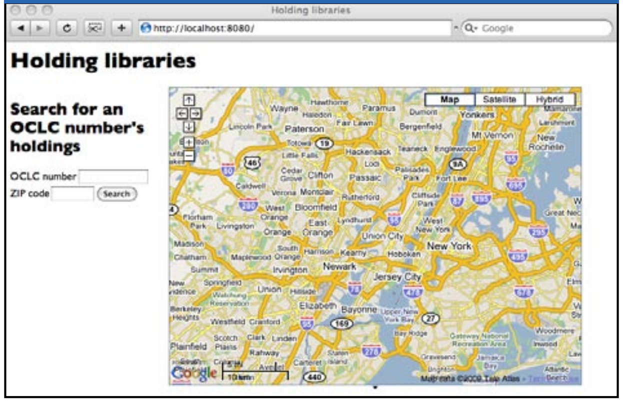
LISTING 5

```
<!-- excerpts from "holdingsmap/templates/index.html" -->

<link href="/json?oclcnum=$rdata['oclcnum']&amp;zip=$rdata['zip']"
      type="application/json" rel="exhibit/data" />

<script
 src="http://static.simile.mit.edu/exhibit/api-2.0/exhibit-api.js"
 type="text/javascript"></script>
<script
 src="http://static.simile.mit.edu/exhibit/extensions-2.0/map/map-
extension.js?gmapkey=$rdata['key']"
 type="text/javascript"></script>
```

FIGURE 1



Holding libraries

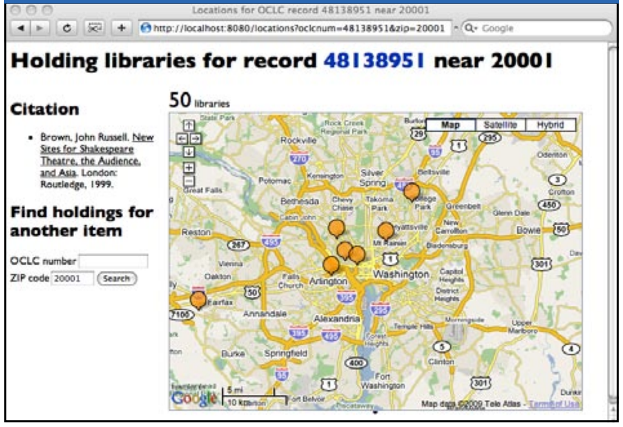
Search for an OCLC number's holdings

OCLC number

ZIP code

Search

FIGURE 2


Locations for OCLC record 48138951 near 20001
http://localhost:8080/locations?oclcnum=48138951&zip=20001
Google
Map
Satellite
Hybrid

Holding libraries for record 48138951 near 20001

Citation

- Brown, John Russell. New Sites for Shakespeare Theatre, the Audience, and Asia. London: Routledge, 1999.

Find holdings for another item

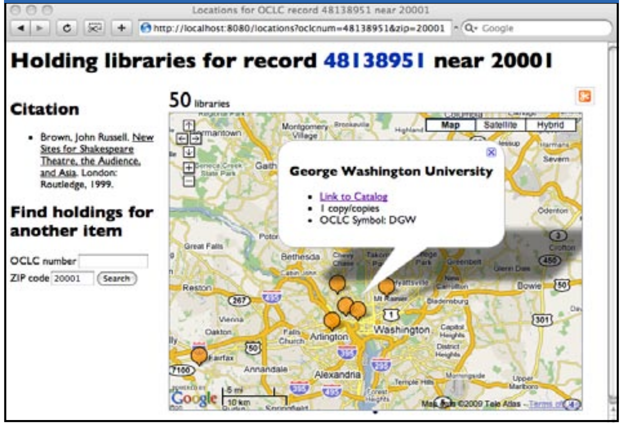
OCLC number

ZIP code 20001

Search

50 libraries

FIGURE 3


Locations for OCLC record 48138951 near 20001
http://localhost:8080/locations?oclcnum=48138951&zip=20001
Google
Map
Satellite
Hybrid

Holding libraries for record 48138951 near 20001

Citation

- Brown, John Russell. New Sites for Shakespeare Theatre, the Audience, and Asia. London: Routledge, 1999.

Find holdings for another item

OCLC number

ZIP code 20001

Search

50 libraries

George Washington University

- Link to Catalog
- 1 copy/copies
- OCLC Symbol: DGW

of holdingsmap.py. (As mentioned above, the copy of holdingsmap.py that you can download with this issue from the *Python Magazine* web site already has a wskey set, but be careful to use it only a few times per day so as not to lock out other readers who want to try the demo.) Now that we've made these changes, we can launch the application and begin trying to find holdings for individual works:

```
$ cd wc/worldcat/examples/holdingsmap
$ ../../../bin/python holdingsmap.py \
  http://0.0.0.0:8080/
```

near the ZIP code 20001. Once we submit the request, we will get a result something like what is shown in Figure 2.

If we click on one of the map markers, a call-out box appears, containing brief information about the institution and a link to the institution's online library catalog, as shown in Figure 3.

How It Works

To see how the application works, look at the holdingsmap.py source code, either in the source package you can download for this issue of *Python*

OCLC launched a beta version WorldCat.org in 2006, which let users search the entire WorldCat database from a freely available web interface for the first time.

And, yes, it is necessary to actually cd to the directory as shown here, because otherwise the web application will be unable to find the accompanying templates directory that contains its HTML files.

Now that the application is running, you can open a web browser on the same machine as the Python script is running and begin looking up locations for items:

```
http://localhost:8080/
```

You can see what the interface looks like in Figure 1. To try this out, let's search for holdings for the OCLC record number 48138951 (*New Sites for Shakespeare Theatre, the Audience, and Asia* by John Russell Brown)

LISTING 6

```
# Sample JSON response from the holdingsmap.py back to the browser

{
  "items": [
    {
      "numberOfCopies": "1",
      "oclcid": "AFU",
      "label": "University of Arkansas - Fayetteville",
      "addressLatLng": "36.065475,-94.15567",
      "link": "http://www.worldcat.org/wcpa/oclc/32597190?page=frame&\
url=http%3A%2F%2Flibrary.uark.edu%2Fsearch%2Fo32597190%26checksum%3D8\
df95d7d1a6a06656475b032a30f69a3&title=University+of+Arkansas+-+Fayett\
eville&linktype=opac&detail=AFU%3AUniversity+of+Arkansas+-+Fayettevil\
le%3AAcademic&app=wcapi&id=foobar",
      "address": "Fayetteville, AR 72701 United States",
      "type": "library"
    }
  ],
  "types": {
    "library": {
      "pluralLabel": "libraries"
    }
  }
}
```

Magazine, or in the example code packaged with the worldcat package itself. This 130-line listing is an *entire* web application that interfaces with both the OCLC and with Google Maps! To run, it needs only the two template files index.html and locations.html that accompany it, in the adjacent templates directory, and that contain the actual HTML of its web pages.

The form rendered by holdingsmap.py sends an HTTP GET request to:

```
http://localhost:8080/locations
```

The form submission includes an oclcnum and zip sent as HTTP query parameters. This then calls the GET() method from the locations class (see line 104 of holdingsmap.py). This method first sends a geocoding request to the Google Maps API for the ZIP code submitted by the user. Next, it sends a CitationRequest to the WorldCat Search API to return a formatted citation for the work, and an xOCLCNUMRequest to the xISBN service to retrieve OCLC record numbers for related works. The method gathers this data into a dictionary that is passed as an argument to a rendering function provided by web.py. The user's browser receives the response using the template in index.html, and the browser sends a second request to http://localhost:8080/json with the same query parameters as the initial request.

How do we tell the browser to load the JSON URL? By providing it as a "link" element up in the header, as you can see in line 3 of Listing 5.

holdingsmap.py responds to this JSON request using the GET() method of the json class (line 92). It first creates a dictionary containing the structure of information required by Exhibit's JSON format. Next, it gathers data from a LibrariesRequest, which returns holdings data for the item specified by the OCLC record number. The XML response is parsed using ElementTree, which provides a list of Element objects. A pool of worker processes applies the process_libraries() over the list of objects. This function extracts data about each institution that holds the work, such as the OCLC organization symbol, the address of the institution, and, if applicable, a link to the work's record in the institution's online catalog.

The function then issues an OCLCSymbolRequest to attempt retrieval of latitude and longitude data from the WorldCat Registry. If this data is not found, it then sends a geocoding request to the Google Maps API for the address specified in the holdings information. Once all this data is gathered, json.GET() calls a web.py function to set the content type of the response, and renders the Python dictionary as a JSON object using simplejson. A sample Exhibit JSON object for the holdings information for OCLC record number 32597190 can be seen in Listing 6.

Conclusion and Future Developments

By now we have covered the basics of using worldcat to interact with several of the APIs that OCLC makes available to assist developers in reusing bibliographic and holdings data from the WorldCat database. As time goes on, I certainly hope more developers find innovative ways to reuse this data. So far, the worldcat module leverages only about a half of the available APIs. In addition to the WorldCat Search API, the xID Services, and the WorldCat Registry, there are three other web services provided by OCLC that are under active development.

WorldCat Identities provides an "identity" or profile page of a particular person or organization based on relationships within the bibliographic data held in WorldCat. The human-readable view of the pages in WorldCat Identities provides an overview of that person or organization's published works, a timeline that displays information about publications by or about the profilee, a list of commonly held titles for that profilee, and links to other identities associated with that person or organization. In addition to the human readable view of the Identities data, OCLC also provides an SRU search interface that returns the data in XML.

Terminology Services is an experimental web service for working with controlled vocabularies. Terminology Services allow users to search descriptions of, as well as terms within, controlled vocabularies; retrieve an individual term within a controlled vocabulary using its identifier; and view relationships between terms such as equivalence, hierarchical relationships, and associative relationships. Like WorldCat Identities and the WorldCat Search API, Terminology Services provides an SRU interface to search the vocabulary descriptions and terms. Available vocabularies include Library of Congress Subject Headings, Medical Subject Headings, and several other vocabularies containing form and genre terms. Users can retrieve information about controlled vocabulary terms in a number of representations, including HTML, MARCXML, SKOS, and Zthes. Furthermore, OCLC also intends to expand Terminology Services to return data in other representations such as BS 8723-5, MARC (ISO 2709), MADS, and MODS.

OCLC also provides an experimental metadata crosswalk service. The Crosswalk Web Service allows developers to translate a group of metadata records from one format to another. For this service, OCLC defines a "metadata format" as a triple of a standard (like MARC or the Dublin Core), a structure (like XML or ISO 2709), and an encoding (MARC-8 or UTF-8 or whatever). This service is a SOAP-based service, and OCLC has provided a WSDL file to assist in the automatic generation of client code for this experimental service.

Finally, OCLC is also encouraging developers to use their APIs through participation in the Third OCLC Research Software Contest. The contest challenge is "to use an OCLC resource to create a Web service that does something interesting, innovative, and useful." The winner of the award will receive a US$2,500 prize and have the opportunity to visit OCLC Research in Dublin, Ohio to discuss their contest entry with OCLC staff.

Given the other available APIs and the Software Contest, I anticipate that I will continue developing worldcat for quite some time. Potential changes include creating a separate SRU client Python module and developing further sample applications for developers to explore. I hope you enjoyed learning about the APIs and the module — let's show OCLC what we can do with Python!

MARK A. MATIENZO is an Applications Developer in the Digital Experience Group of the New York Public Library, where he works on digital repositories; digital projects with archives and manuscript collections; search, indexing, and information retrieval applications; linked data and semantic web projects; and other architecturally interesting problems.