Web API- Application Programming Interface

* Set of protocols for building tools and applications that use a specific web service
* Often a structure for URL/URI formulation to make a query on a web database, such as Neotoma, iDigBio, or the Paleobiology Database.
  + URL = Uniform Resource Locator, or web address.
  + URI = Uniform Resource Identifier, or web address of a service, like an API.
* Calls are usually sent as URL “get” statements: values appended to a URL after a “?” that are processed as queries by a SQL or other web database.

For example, Google has APIs for most of its products. You could write an R (or Python, etc.) script that would connect to the Google Calendar API to allow you to automatically change events on your calendar or report agenda items back to you in your own custom environment.

APIs exist for many important biological and paleobiological databases. We will work through examples from GeoLocate, the georeferencing service, iDigBio, the Paleobiology Database, and Neotoma. Different databases/services produce different kinds of API returns. The simplest kinds are comma delimited text files, but many APIs are now returning JSON documents. We will begin with text and move on to JSON.

First, the Paleobiology database. The full documentation for the Paelobiology Database API is located at <https://paleobiodb.org/data1.2/>

The PaleioBioDB API uses calls to different URLs to return different kinds of data. We’ll try some calls to the occurrence APIs.

In your web browser, try typing: <https://paleobiodb.org/data1.2/occs/list.txt?base_name=Camelidae&interval=Pleistocene&show=loc,class>

Note that this call is telling the API to search the base\_name, “Camelidae”; the interval, “Pleistocene”; and to return the attributes “loc,class”. In this API, “base\_name” searches for a taxonomic name at any level of the Linnaean hierarchy and returns any occurrences with that name, its synonyms, or its subtaxa. You should try changing to a name of interest to you and viewing the results.

Similarly, interval returns occurrences that fall within the specific geologic interval. These intervals include North American Land Mammal Ages as well as the conventional Geologic Timescale. Try “Blancan” and “Burdigalian” to see what happens. Try an interval of interest to you.

You can search by taxonomy, age, environment, even by the specific locality or occurrence ID in the database.

The list of potential calls is here: <https://paleobiodb.org/data1.2/occs/list_doc.html>

You can narrow your results geographically by invoking “lngmin” and “lngmax” as well as “latmin” and “latmax”. Try for a few minutes to explore the data with different calls, using the call list.

Now we will try iDigBio, the hub of the NSF’s Advancing the Digitization of Biological Collections (ADBC) program.

iDigBio (Integrated Digitized Biocollections) is the central resource for searching digital specimen data liberated by the ADBC program. This program funds networks of collections centered on research themes (Thematic Collection Networks, TCNs) to enter specimen data into databases, to georeference collection locations, and to photograph those specimens. Most importantly, these data have to be made available online and connected to the iDigBio portal. iDigBio cleans up the data and provides several ways to access them, including the API we will be investigating.

First, try running this query:

<https://search.idigbio.org/v2/search/records/?rq=%7B%22scientificname%22%3A+%22camelus+bactrianus%22%2C+%22hasImage%22%3A+true%7D&limit=5>

There are several URL encoding tags in this statement. The %7B means {, %22 means “, the %3A means :, the %2C is a comma, and the %7D is }. The + means a space. So this statement translates as ?rq={“scientificname”: “camelus bactrianus”, “hasImage”: true}&limit=5. Try changing some of the values and seeing what you get.

This encoding formats the URL into what is called JSON, or JavaScript Object Notation. The output from this call is also in JSON. It is an important way to transmit the sort of complicated, nested data saved in relational databases when you have to send a flat file over the web.

The documentation for the iDigBio API is here: <https://www.idigbio.org/wiki/index.php/IDigBio_API>

More about JSON

At its most basic level, JSON transmits data objects in attribute-value pairs. It has come to replace XML, which was the previous standard for this sort of data transmission.

JSON is composed of objects, enclosed by curly brackets, which may have any number of attributes named in quotes, with values after a colon, separated by commas. You may also present an array, or an ordered collection of values, enclosed in square brackets.

As an example, you could present an occurrence like this example from the Neotoma API:

{

"SiteLongitudeWest": -103.31666666666666,

"SiteLatitudeSouth": 34.283333333333339,

"TaxonName": "Smilodon fatalis",

"VariableElement": "bone/tooth",

"Value": 1.0,

"VariableContext": null,

"TaxaGroup": "MAM",

"SampleAgeYounger": 15332.0,

"SampleAgeOlder": 30041.0,

"SiteLongitudeEast": -103.31666666666666,

"SiteAltitude": 1280.0,

"VariableUnits": "present/absent",

"DatasetID": 4564,

"SampleAge": null,

"SiteLatitudeNorth": 34.283333333333339

}

Can you tell what kind of occurrence this JSON object is describing? How old is it? Where is it located?

This record comes from searching the Neotoma DB API. Here is the example API call:

[http://api.neotomadb.org/v1/data/sampledata?taxonname=Smilodon\*](http://api.neotomadb.org/v1/data/sampledata?taxonname=Smilodon*)

Notice when you make this call that the JSON is computer-friendly but not human-friendly. Try it again with the ‘pretty’ format tag:

[http://api.neotomadb.org/v1/data/sampledata?taxonname=Smilodon\*&format=pretty](http://api.neotomadb.org/v1/data/sampledata?taxonname=Smilodon*&format=pretty)

You should be able to see the nested set of JSON objects, including the occurrences returned as comma-separated objects within the array “data”.

Try experimenting with the search, substituting different names. We’ll do more with the Neotoma API in a bit.

JSON is becoming the standard for data transfer in web services. R has several packages for dealing with JSON-formatted data. We will use some examples from the package RJSONIO. We will also use the package RCurl, which has functions to let you query APIs from within the R environment.

Go ahead and install and load these two packages in your preferred R environment.

You will also want to make sure you have the latest version of R installed, or else the secure connection (hhtp***s***) won’t work in the following queries.

Now create a query for the PaleoBioDB API using the example from above:

library(RCurl)

library(RJSONIO)

q <- "https://paleobiodb.org/data1.2/occs/list.txt?base\_name=Camelidae&interval=Pleistocene&show=loc,class"

Create an object to receive the results:

a <- basicTextGatherer()

And execute that query:

curlPerform(url = q, writefunction = a$update)

Finally, view the data:

a$value()

You can see that the data have come in as a character vector; a long list of text strings with no clear structure. Luckily, the PaleoBioDB API also has a JSON interface.

Change your query to refer to list.json:

q <- "https://paleobiodb.org/data1.2/occs/list.json?base\_name=Camelidae&interval=Pleistocene&show=loc,class"

Rerun your query and look at a$value() again. How does it look now?

You can convert it to a data frame:

tmp <- fromJSON(a$value())

records <- tmp$records

results <- data.frame(records[1], stringsAsFactors = FALSE)

for(x in records[-1]){

x<-data.frame(x, stringsAsFactors = FALSE)

results <- merge(results, x, by = intersect(names(results),names(x)), all = TRUE)

}

When you have gotten this code to work, go through and make comments to describe what each section is doing. Remember, you can make comments in your R code by placing a hashtag (#) at the beginning of a line.

Try playing with your queries to see what you can pull from the PaleoBioDB, or even extending to the other two APIs we have explored.

**More on the Neotoma DB API**

If you think back to the Neotoma API example, you can see that the example API call is reporting data from only part of the distributed database schema of Neotoma. In fact, the Neotoma API is designed around a set of different URLs, each of which allows a user to search a portion of the database. So, if you want the full Site information for locations with *Smilodon* present, you would have to search on *Smilodon* in the SampleData URI (as we did in the example), pull the DatasetID values from those returns, then search on the Dataset URI (api.neotomadb.org/v1/data/datasets) for those DatasetIDs, which would, in turn, produce the SiteIDs, which you would then search on the Sites URI (api.neotomadb.org/v1/data/sites). This sort of searching would be cumbersome if you were to do it by hand, but fortunately you can script a computer to do it for you. In fact, you don’t have to write the scripts to do it, because they have already been constructed and provided to the community as the R neotoma package.

Using this package will the the subject of our next module, but bear in mind that the other APIs also have wrapper packages to simplify data calls in R. The PaleoBioDB has a package, paleiobioDB, and iDigBio has a package, ridigbio. Currently, a large group of collaborators is working on a single API and wrapping R package to access both PaleoBioDB and Neotoma at the same time, as well as linking to many online museum databases and iDigBio. We wanted to introduce you to the underlying architecture here so you would understand what these packages are doing, and would know that you can crack them open and hack your own solutions if you cannot get them to give you the data or format of data that you need for your work.