Web APIs for non-programmers

November 18, 2013 in HowTo

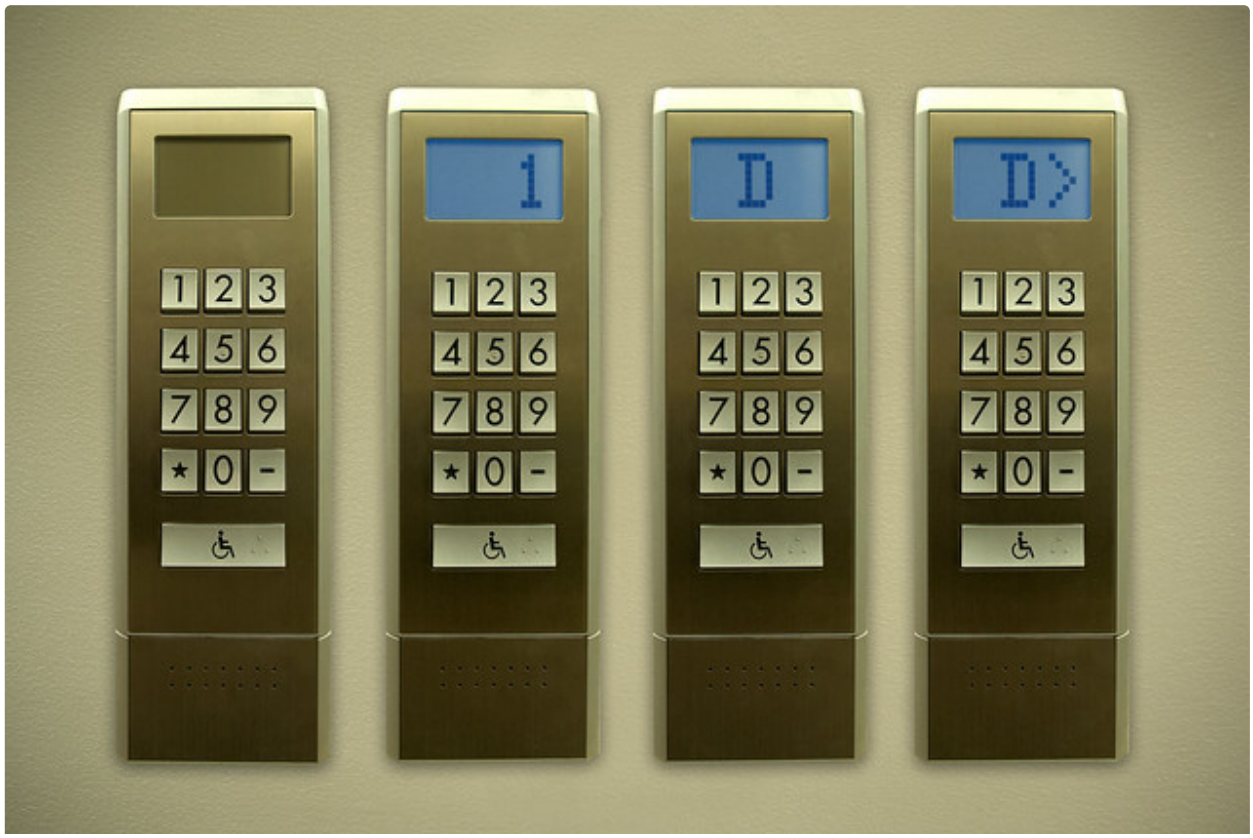

Photo credit: Jason Kuffer

School of Data is re-publishing Noah Veltman's Learning Lunches, a series of tutorials that demystify technical subjects relevant to the data journalism newsroom.

This Learning Lunch is about web APIs, a powerful way to gain direct access to data.

What is an API?

API stands for **A**pplication **P**rogramming **I**nterface. Don't worry about the AP, just focus on the I. **An API is an interface**. You use interfaces all the time. A computer operating system

is an interface. Buttons in an elevator are an interface. A gas pedal in a car is an interface.

An interface sits on top of a complicated system and simplifies certain tasks, a middleman that saves you from needing to know all the details of what's happening under the hood. A web API is the same sort of thing. It sits on top of a web service, like Twitter or YouTube, and simplifies certain tasks for you. It translates your actions into the technical details for the computer system on the other end.

What is a web API?

There are lots of different flavors of web API. One of the most common, and most accessible to non-programmers, is called a REST, or RESTful, API. From now on, when I say "web API" I mean a REST API. If you want to nitpick, there are other kinds of web API. But why would you want to do that?

A web API is an interface with URLs as the controls. In that respect, the entire web is a sort of API. You try to access a URL in your browser (also known as a **request**), and a web server somewhere makes a bunch of complicated decisions based on that and sends you back some content (also known as a **response**). A standard web API works the same way.

The key difference between an ordinary URL and a URL that's part of a web API is that an ordinary URL sends back something pretty designed to look good in your browser, whereas **a web API URL sends back something ugly designed to be useful to a computer**.

When you request the URL `http://twitter.com/` in a browser you get back a nice-looking webpage with a bunch of colors and pictures and buttons. It's designed for a human to look at and for a browser to draw on a screen. But it sucks if what you want is to gather and analyzing data. The structure of the underlying document is very confusing and inconsistent. It's hard to extract the bits you care about, and it doesn't provide you information in bulk.

When you request this web API URL instead, you get back an ugly-looking chunk of plain text with no decorations:

```
https://api.twitter.com/1.1/statuses/home_timeline.json
```

It's designed for a computer to read. It sucks if what you want is to look at some tweets while you drink your morning coffee, but it's great if you want to extract and analyze tweets and their metadata, especially if you want to extract a lot of them at one time.

Web APIs are a way to strip away all the extraneous visual interface that you don't care about and get at the data. In the same vein, you can often think of them as a limited shortcut into a web service's database. Twitter won't just let you log in to their

internal database and poke around (unless you work there), but they will give you an easier way to access it in certain limited ways using an API.

Why would I want to use a web API?

1. To **get information that would be time-consuming to get otherwise**. Most often, you *could* get some data from a website manually. You could sit there and click through 100 pages and copy/paste it into a spreadsheet (or make your intern do it). If you're a programmer, you could spend a lot of time figuring out how a page is structured and write a custom scraper to get the data out. Either way, it would take a long time and be really aggravating. In cases like this, an API is a time-saver that lets you get what you want quickly instead.
   Example: you want every public photo on Flickr that has the tag 'maui,' was taken with a Nikon camera in 2012, and is wider than it is tall. You could sit there and run the search on Flickr, and then click through all the results one at a time, finding the right ones and copying them to a list. It would take hours. If you use the Flickr API, you can get much more specific about what you want, and instead of getting back a gallery you have to click through one-by-one, you get back a big text list with all the details.
2. To **get information that you can't get otherwise**. Sometimes web APIs give you a way to access information that doesn't exist on the normal web, stuff that's in a database attic somewhere or that's hidden from normal users because normal users wouldn't care about it (as a journalist, there are probably lots of things you care about that a casual web user doesn't).
   Examples: Twitter might not display tweets before a certain date on its website, but still allow you to download them via the API. The Rotten Tomatoes API might give you extra metadata on release dates that's not displayed on the website.
3. To **automate a news app that needs live data** from other sources.
   Example: you want your Super Bowl coverage to include a live stream of every Instagram photo taken near the stadium.
4. To use it as a **more direct interface for reading and writing data to a service**.
   Example: you use MailChimp to manage all of your mailing lists. You want to be able to automatically add and remove people from lists when certain things happen instead of someone having to manually manage those lists.

Reasons #1 and #2 are both ways in which a non-coding journalist can still make their life easier by retrieving data in bulk or with great specificity. Reasons #3 and #4 both generally require some programming savvy.

OK, so how does it work?

The basics

First, you need to figure out whether the data you're interested in is available through an API. The easiest way to do this is by Googling, but you can also look for an 'API' or 'Developers' link on a web service's homepage (it's often buried at the bottom of the page). Sometimes there is an "unofficial" or "undocumented" API for a site that was built by someone else; the reliability of a service like that is a question mark, so caveat usor.

Usually you'll find a set of documentation, sometimes called an API **specification**. This is the API's instruction manual. It tells you what all the controls do. Here are a few examples:

The Twitter API: https://dev.twitter.com/docs/api/1.1
The New York Times Best Sellers API: http://developer.nytimes.com/docs/best_sellers_api/

These instruction manuals are very dense with information, but what they mainly describe is a list of URLs you can use to retrieve data. These URLs can called many things, like *resources* or *methods*. Using one of them is also sometimes called an *API request* or *API call*. A "request" is a good way to think about it, because that's exactly what you're doing. You're going up to the friendly librarian at Twitter or YouTube or whatever other popular service, and asking them to give you some information.

For each URL, you will usually find a page or a section explaining more details about how to use it. Typically this consists of two things: **parameters** and the **response**.

Parameters

Parameters are information you can supply as part of the URL in order to define what you want. For example, if you're running a search with the Twitter API, the URL might be:

```
https://api.twitter.com/1.1/search/tweets.json
```

but this doesn't tell Twitter anything about what you want to search for. You have to add in parameters, like so:

```
https://api.twitter.com/1.1/search/tweets.json?q=burritos
```

in order to specify that you are searching for 'burritos'.

Parameters might also be called *variables*, because they vary – they are the parts of a URL that you change in order to change what you're asking for. Think of each web API resource/method/URL as a gadget, and each parameter as a specific knob on that gadget.

For any given URL, some parameters might be required, some might be optional. Let's go back to the Twitter search example:

`https://api.twitter.com/1.1/search/tweets.json?q=burritos`

The `q` parameter that gives it a word to search for is a required parameter. If you don't supply one, the API won't accept your request because you're searching without telling it what to search for. There are lots of other optional parameters, though. For example, if you add a `lang` parameter, you're telling it you only want results that are in a certain language. This would do the same search, but only bring back results in English:

```
https://api.twitter.com/1.1/search/tweets.json?q=burritos&lang=en
```

That's just the tip of the iceberg. For any given URL, there are usually lots of optional parameters that will further refine what you're doing. The instructions will probably list what the *default* is for every parameter you don't specify. This is important. For example, the default number of results for the search above is 15. If you want more than 15 at a time, you have to specify that:

```
https://api.twitter.com/1.1/search/tweets.json?q=burritos&lang=en&count=100
```

Parameters specify things like:

- How many results do you want back at once?
- How do you want the results to be sorted?
- What date range do you want to search?
- What location do you want to search?
- What format do you want the results in?

You may be wondering about how those parameters end up slapped on to the end of the URL in these examples. We'll talk about that more in a second, in the **GET and POST** section.

The Response

When you use a web API, you supply a URL (possibly with some extra parameters), and you most likely get back a response. The response can be one of two things:

- Some data – this is what you want.
- An explanation of why your request failed – this might tell you that you failed to supply a required parameter, you're over your rate limit (see below), you need to supply authentication to make that particular request (see below), or there was some other problem with the request.

The documentation should give you some information about sort of response gets sent back for each type of request. Here's an example of a response you would get if you used the Rotten Tomatoes API in order to search for 'skyfall':


```
{"total":1,"movies":[{"id":"770680844","title":"Skyfall","year":2012,"mpaa_ratin
g":"PG-13","runtime":145,"critics_consensus":"Sam Mendes brings Bond surging back w
ith a smart, sexy, riveting action thriller that qualifies as one of the best 007 f
```

ilms to date.","release_dates":{"theater":"2012-11-09","dvd":"2013-03-11"},"rating
s":{"critics_rating":"Certified Fresh","critics_score":92,"audience_rating":"Upright","audience_score":88},"synopsis":"In Skyfall, Bond's loyalty to M is tested as her past comes back to haunt her. As MI6 comes under attack, 007 must track down and destroy the threat, no matter how personal the cost. -- (C) Official Site","posters":{"thumbnail":"http://content9.flixster.com/movie/11/16/82/11168295_mob.jpg","profile":"http://content9.flixster.com/movie/11/16/82/11168295_pro.jpg","detailed":"http://content9.flixster.com/movie/11/16/82/11168295_det.jpg","original":"http://content9.flixster.com/movie/11/16/82/11168295_ori.jpg"},"abridged_cast":[{"name":"Daniel Craig","id":"162687443","characters":["James Bond"]},{"name":"Judi Dench","id":"162652435","characters":["M"]},{"name":"Javier Bardem","id":"162661456","characters":["Silva"]},{"name":"Ralph Fiennes","id":"162653681","characters":["Gareth Mallory","Mallory"]},{"name":"Naomie Harris","id":"162705781","characters":["Eve"]}],"alternate_ids":{"imdb":"1074638"},"links":{"self":"http://api.rottentomatoes.com/api/public/v1.0/movies/770680844.json","alternate":"http://www.rottentomatoes.com/m/skyfall/","cast":"http://api.rottentomatoes.com/api/public/v1.0/movies/770680844/cast.json","clips":"http://api.rottentomatoes.com/api/public/v1.0/movies/770680844/clips.json","reviews":"http://api.rottentomatoes.com/api/public/v1.0/movies/770680844/reviews.json","similar":"http://api.rottentomatoes.com/api/public/v1.0/movies/770680844/similar.json"}}],"links":{"self":"http://api.rottentomatoes.com/api/public/v1.0/movies.json?q=Skyfall&page_limit=1&page=1"},"link_template":"http://api.rottentomatoes.com/api/public/v1.0/movies.json?q={search-term}&page_limit={results-per-page}&page={page-number}"}

That's pretty unhelpful, right? It's a dense mess, the data equivalent of meatloaf. Let's unpack it a little by adding indentation:


```
{
   "total":1,
   "movies":[
      {
         "id":"770680844",
         "title":"Skyfall",
         "year":2012,
         "mpaa_rating":"PG-13",
         "runtime":145,
         "critics_consensus":"Sam Mendes brings Bond surging back with a smart, sexy, riveting action thriller that qualifies as one of the best 007 films to date.",
         "release_dates":{
            "theater":"2012-11-09",
            "dvd":"2013-03-11"
         },
         "ratings":{
            "critics_rating":"Certified Fresh",
            "critics_score":92,
            "audience_rating":"Upright",
```

```
            "audience_score":88
         },
         "synopsis":"In Skyfall, Bond's loyalty to M is tested as her past comes ba
ck to haunt her. As MI6 comes under attack, 007 must track down and destroy the thr
eat, no matter how personal the cost. -- (C) Official Site",
         "posters":{
            "thumbnail":"http://content9.flixster.com/movie/11/16/82/11168295_mob.j
pg",
            "profile":"http://content9.flixster.com/movie/11/16/82/11168295_pro.jp
g",
            "detailed":"http://content9.flixster.com/movie/11/16/82/11168295_det.jp
g",
            "original":"http://content9.flixster.com/movie/11/16/82/11168295_ori.jp
g"
         },
         "abridged_cast":[
            {
               "name":"Daniel Craig",
               "id":"162687443",
               "characters":[
                  "James Bond"
               ]
            },
            {
               "name":"Judi Dench",
               "id":"162652435",
               "characters":[
                  "M"
               ]
            },
            {
               "name":"Javier Bardem",
               "id":"162661456",
               "characters":[
                  "Silva"
               ]
            },
            {
               "name":"Ralph Fiennes",
               "id":"162653681",
               "characters":[
                  "Gareth Mallory",
                  "Mallory"
               ]
            },
            {
```

```
               "name":"Naomie Harris",
               "id":"162705781",
               "characters":[
                  "Eve"
               ]
            }
         ],
         "alternate_ids":{
            "imdb":"1074638"
         },
         "links":{
            "self":"http://api.rottentomatoes.com/api/public/v1.0/movies/770680844.
json",
            "alternate":"http://www.rottentomatoes.com/m/skyfall/",
            "cast":"http://api.rottentomatoes.com/api/public/v1.0/movies/770680844/
cast.json",
            "clips":"http://api.rottentomatoes.com/api/public/v1.0/movies/77068084
4/clips.json",
            "reviews":"http://api.rottentomatoes.com/api/public/v1.0/movies/7706808
44/reviews.json",
            "similar":"http://api.rottentomatoes.com/api/public/v1.0/movies/7706808
44/similar.json"
         }
      }
   ],
   "links":{
      "self":"http://api.rottentomatoes.com/api/public/v1.0/movies.json?q=Skyfall&p
age_limit=1&page=1"
   },
   "link_template":"http://api.rottentomatoes.com/api/public/v1.0/movies.json?q={se
arch-term}&page_limit={results-per-page}&page={page-number}"
}
```


Now it starts to make a little more sense. Don't worry about all the brackets and commas just yet. You can see broadly that you're getting back some basic info about the movie *Skyfall*: its ratings, its release date, its cast, etc.

When you get a response back with data, it will almost always be in one of two formats: XML or JSON. In order to get much use out of this data, you'll need to wrap your head around how these two formats express data. To learn more, read XML and JSON.

GET vs. POST

For any API method/resource/URL that involves parameters, you will typically supply them in one of three ways:

Modifying the URL itself

Often times the URL for an API includes parts of the URL that can be changed themselves. For example, a URL might be listed as:

```
http://www.api.com/list/[page number]/[sort by]/[format]/
```

In this case, to get different variations, you would modify the URL accordingly:

```
http://api.ebay.com/list/1/price/xml/ (Get page 1, sorted by price, in XML format)
http://api.ebay.com/list/5/date/json/ (Get page 5, sorted by date, in JSON format)
```

Adding GET parameters onto the end of a URL

This is a slight variation on the version above. When you look at a URL with a question mark followed by a bunch of junk, everything after the question mark is a list of `GET` parameters. Each one has a name and a value, like so:

```
?name=value&name=value&name=value...
```

For example, in this URL:

```
http://www.google.com/search?query=ducktales&language=english
```

You are supplying two extra parameters:

- A parameter called `query` with the value `ducktales`
- A parameter called `language` with the value `english`

Most popular web APIs use this mechanism as a way of supplying parameters for looking something up or getting a list. They give you a base URL and then you add on extra `GET` parameters. For example:

```
http://api.yelp.com/v2/search?term=burritos&location=Chicago
https://api.foursquare.com/v2/venues/search?query=burritos&near=Chicago
```

Sending POST variables

If the documentation tells you that you have to use the `POST` method of making a request, you've got some problems. `POST` is a way to send *hidden* data as part of a request, data that doesn't get listed as part of the URL. You use `POST` all the time when you use the web. When you fill out a form to buy something online, it submits your payment information using `POST`, because it would be bad if you loaded a URL that put that information out in the open, like:

```
http://www.amazon.com/checkout?creditcardnumber=1234567812345678
```

`POST` is generally used for ephemeral or sensitive things that you don't want to have a permanent URL for, like things that modify data in a database.

Let's imagine you have a page where someone submits a form with their email address to be added to a list. If that data gets passed via GET:

```
http://mailinglist.com/add?q=john@smith.com
```

and John Smith accidentally shares that URL with others, they will all re-initiate that action when they load the URL. John will get added to the list over and over. If you instead send it as `POST` parameters to the URL:

```
http://mailinglist.com/add
```

John won't be able to reproduce that action without filling out the form again.

If an API method you need requires to send parameters via `POST`, you can still do it without writing any code, but it's a little more complicated, because you can't just do it in the browser address bar. You can use a program like Fiddler or browser extensions like Postman or REST Client. These will allow you to add `POST` parameters to a request.

API keys

Some APIs are genuinely public, open to all. When you find an API like this, you can just take an API URL, add some parameters, put it into your browser, and you'll get back a useful response. An example of this is the old Twitter API. You can just put a URL like this into your browser and get results:

```
https://search.twitter.com/search.json?q=burritos
```

Go ahead, try it!

These days, most APIs are not quite that freewheeling. Instead, most of them require some sort of authentication, often in the form of an **API key**, a long string of letters and numbers that functions like a password. The bad news is that in order to use these APIs, you have to do a little extra work, filling out a form to request access first. The good news is, this process is usually quite quick, and plenty of APIs are still free even though they require an API key.

For example, if you try to search for 'skyfall' in the Rotten Tomatoes API without an API key:

```
http://api.rottentomatoes.com/api/public/v1.0/movies.json?q=Skyfall
```

You'll get back this response:

```
{"error":"Account Inactive"}
```

If you supply an API key as a parameter with your request, like this:

```
http://api.rottentomatoes.com/api/public/v1.0/movies.json?q=Skyfall&apikey=krw98sa2
```

You'll get what you want.

As long as you can supply the API key as a parameter with the URL, you can still make requests. If you have to authenticate in some other way, you might be out of luck without a little bit of programming.

Rate limiting

Many APIs include limits on how many requests you can make per hour or per day in order to prevent you from overloading their servers. If you aren't a programmer, these limits probably won't be an issue. They are there to prevent you from automating a flurry of thousands of requests. For information on rate limits, see a particular API's documentation. Note also that lots of APIs are free up to a certain rate limit and then charge you money if you want to exceed it.

An example API workflow for a non-programmer

1. I want some data from service X. Does service X have an API? Hooray, it does, and it's free!
2. Look at the API documentation. Figure out if there is a URL that retrieves the kind of data you're looking for. There is? Great!
3. Sign up for an API key if one is required.
4. Figure out what parameters you need to include in the URL in order to get the exact data you want.
5. Load the URL, parameters included, into your browser. Get back a response! If it didn't work, go back to step 4.
6. Take that data and unpack it. For more on this, read XML and JSON.

Disclaimer

APIs are very useful for getting data on a one-off basis, especially if you can master XML and JSON, but be wary about building ongoing processes on them. There is no guarantee that an API won't change. APIs or specific parts of them are discontinued all the time, making it harder to get data that used to be easy to get. Or, the service starts charging a lot of money to use the API now that everyone relies on it. Keep this in mind when thinking about how you want to use a particular API.

Some popular free web APIs

New York Times
Yelp
Twitter
Flickr
Foursquare
Instagram
LinkedIn
Vimeo
Tumblr
Google Books
Facebook
Google+
YouTube
Rotten Tomatoes

A long list of REST APIs: http://www.programmableweb.com/apis/directory/1?protocol=REST

← An Introduction to Mapping Company Networks Using Gephi and OpenCorporates, via OpenRefine

We're hiring: Open Development Toolkit Lead →