

# Creating a Python Script for Twitter Search

Joe Bob Hester  
Associate Professor  
University of North Carolina  
School of Journalism & Mass Communication  
@joebobhester

February 12, 2014

## Abstract

With a little help from Google, you can probably find a preexisting Python script that you can use to search Twitter. However, in my experience, you can rarely find one that does exactly what you want. That's why you might want to build your own. And, since we're trying to learn a little programming along the way, it's a great exercise. This document takes you through the step-by-step process of creating the program.

Note: This document was produced using L<sup>A</sup>T<sub>E</sub>X.

## 1 The Big Picture

The first step is to think about exactly what you want your program to do. We know we want to search Twitter, but we also need to think about the specifics of that type of search. For example:

- Twitter search only returns 100 statuses (tweets) per call to the API. Do we need to be able to retrieve more than that number?
- What different parameters do we need to include in the search? The only parameter that is required is the query itself (`q`), but there are optional parameters that may be added.
- How do we want to store the resulting data? Do we want to store it using JSON or convert it to some other format?

### 1.1 This search program

In order to create the most useful and flexible program for everyone who is participating in this workshop, we'd probably answer those questions like this:

- For this particular exercise, we want to be able to search back as far as the API will let us (currently 6-9 days).
- We want to be able to easily include additional search parameters in order to make the program as flexible as possible.
- And, we want to store all the data that is returned, so JSON makes the most sense.

## 1.2 Programming Implications

Those answers have some very specific implications for our program, such as:

- If we want to search back more than 100 statuses, we'll need to use some type of loop to page back through the search results.
- We'll need to include all possible parameters in the program yet make it easy for the user to pick and choose from the parameters s/he wants to use.
- We've already learned how to store data as a JSON file. Personally, I'm a big fan of creating an individual file for each set of 100 results. It tends to make it easier to deal with computer crashes, Twitter API errors, etc.

## 2 Creating the Authentication Request

### 2.1 Libraries

We know we'll need the `oauth2` library for authentication. Authentication also requires a timestamp, so we'll also need the `time` library, which will also come in handy for dealing with the Twitter API rate limits (for search, Twitter limits individual users to 180 API calls per 15-minute period).

Even though they aren't used for authentication, later we'll need two other libraries we're already familiar with, `urllib2` and `json`, so let's go ahead and add them now.

---

```
import oauth2
import time
import urllib2
import json
```

---

### 2.2 Fixed Authentication Parameters

There are certain base authentication parameters that will not change regardless of the user or the particular search. The "Resource URL" (aka API endpoint) for all searches is the same: `https://api.twitter.com/1.1/search/tweets.json`, which we'll assign to the variable `url1`.

Other requirements include the `version`, `nonce`, and `timestamp`. We'll include these parameters in a dictionary named `params`. By using a dictionary, we can allow the user to add more parameters easily.

---

```
url1 = "https://api.twitter.com/1.1/search/tweets.json"
params = {
    "oauth_version": "1.0",
    "oauth_nonce": oauth2.generate_nonce(),
    "oauth_timestamp": int(time.time())
}
```

---

## 2.3 Variable Authentication Parameters

Other authentication parameters will be unique to each user of the program. Once you've created a Twitter app, you will be provided with four specific parameters:

- API key (aka consumer key)
- API secret (aka consumer secret)
- Access token
- Access token secret

We'll set these up as two variables: `consumer` and `token`. Be sure to insert your own unique keys and secrets in your code as strings.

Since the authorization header will need the two keys, we'll add them to the `params` dictionary.

---

```
consumer = oauth2.Consumer(key="INSERT API KEY", secret="INSERT API SECRET")
token = oauth2.Token(key="INSERT ACCESS TOKEN", secret="INSERT ACCESS TOKEN SECRET")

params["oauth_consumer_key"] = consumer.key
params["oauth_token"] = token.key
```

---

## 2.4 Creating and Signing the Request

Now we're ready to set up the actual authentication request. Recall from before that since Twitter search only returns 100 results at a time, we want to set up the request as a loop. For now, we only want the loop to execute one time.

Twitter currently encrypts using a 160-bit (SHA-1) hash-based message authentication code (HMAC), which we specify in the `signature_method`. The `print headers` statement will print the authorization header to the console. The `print url` statement will print the actual URL produced by the program.

Once you've added this code to your program, it should run without errors. Give it a try.

---

```
for i in range(1):
    url = url1
    req = oauth2.Request(method="GET", url=url, parameters=params)
    signature_method = oauth2.SignatureMethod_HMAC_SHA1()
    req.sign_request(signature_method, consumer, token)
    headers = req.to_header()
    url = req.to_url()
    print headers
    print url
```

---

Your output should look something like this (keys and tokens are simulated):

```
{'Authorization': u'OAuth realm="", oauth_body_hash="2jnj7l5rSw0yVb2jnj7l5rSw0", oauth_nonce="43928464", oauth_timestamp="1392145106", oauth_consumer_key="bByA2jnM9IhbByAnna", oauth_signature_method="HMAC-SHA1", oauth_version="1.0", oauth_token="12345-EMBqjfSxvGiPlvT7CqjfSxvG2fnq9c", oauth_signature="r5S8qjfSxvGSUKWD1HWkXivqjfSxvG"}'}
```

[https://api.twitter.com/1.1/search/tweets.json?oauth\\_body\\_hash=2jnj7l5rSw0yVb2jnj7l5rSw0&oauth\\_nonce=43928464&oauth\\_timestamp=1392145106&oauth\\_consumer\\_key=bByA2jnM9IhbByAnna&oauth\\_signature\\_method=HMAC-SHA1&oauth\\_version=1.0&oauth\\_token=12345EMBqjfSxvGiPlvT7CqjfSxvG2fnq9c&oauth\\_signature=r5S8qjfSxvGSUKWD1HWkXivqjfSxvG](https://api.twitter.com/1.1/search/tweets.json?oauth_body_hash=2jnj7l5rSw0yVb2jnj7l5rSw0&oauth_nonce=43928464&oauth_timestamp=1392145106&oauth_consumer_key=bByA2jnM9IhbByAnna&oauth_signature_method=HMAC-SHA1&oauth_version=1.0&oauth_token=12345EMBqjfSxvGiPlvT7CqjfSxvG2fnq9c&oauth_signature=r5S8qjfSxvGSUKWD1HWkXivqjfSxvG)

### 3 Creating a Simple Search

So far, we haven't actually searched for anything. We can create a search by adding the required query (q) parameter and query term at the beginning of our loop. In this example, we'll search for "pizza." This would also be a good time to add the count parameter, which is used to specify the number of results to return. For now, we'll set it at two (100 is the maximum; 15 is the default).

We'll delete the `print headers` and `print url` statements and replace them with two lines to format the search results as JSON (these statements should look very familiar to you; we've used them before).

Finally, we'll add a print statement to print out the results. The loop should now look like this:

---

```
for i in range(1):
    url = url1
    params["q"] = "pizza"
    params["count"] = 2
    req = oauth2.Request(method="GET", url=url, parameters=params)
    signature_method = oauth2.SignatureMethod_HMAC_SHA1()
    req.sign_request(signature_method, consumer, token)
    headers = req.to_header()
    url = req.to_url()
    response = urllib2.Request(url)
```

---

```
data = json.load(urllib2.urlopen(response))
print data
```

---

The result of this search is a dictionary containing two keys: `search_metadata` and `statuses`. The value associated with the `statuses` key is a list of statuses, and each status is a dictionary containing a variety of metadata.

### 3.1 Null Search Results

If the search query does not find any appropriate results, the `statuses` key will have an empty list (`[]`) as its value. While null search results may occur because the API cannot find any results matching the query, it can also occur when the time range being searched is outside the 6-to-9-day window that can be searched. Later, we will see how to page back through the search results in order to search back as far as the API will let us. In doing that, we will eventually reach a point where a null search result is returned. So, now would be a good time to go ahead and add some code to test for that. We'll do that using Python's `if` statement, which will test to see if the `statuses` value is an empty list. If it is, the program will print an "end of data" message and stop the `for` loop. Otherwise, the program will print the data.

---

```
for i in range(1):
    .
    .
    .
    data = json.load(urllib2.urlopen(response))
    if data["statuses"] == []:
        print "end of data"
        break
    else:
        print data
```

---

### 3.2 More Search Parameters

Let's go ahead and add the other possible search parameters with empty strings as values. Remember that all the search parameters must come before the `req` assignment in the loop.

---

```
for i in range(1):
    url = url1
    params["q"] = "pizza"
    params["count"] = 2
    params["geocode"] = ""
    params["lang"] = ""
    params["locale"] = ""
    params["result_type"] = "" # Example Values: mixed, recent, popular
    params["until"] = ""
    params["since_id"] = ""
```

---

```

params["max_id"] = ""
params["include_entities"] = ""
req = oauth2.Request(method="GET", url=url, parameters=params)
.
.
.

```

---

### 3.3 Searching During a Particular Time Range

So far, our search is performed starting with the most recent status that fulfills the search query and then searches back in reverse-chronological order. What if we want to search during a specific time period such as 8:00-9:00 p.m. on Monday, February 10, 2014?

All Twitter statuses have a unique ID number, and those numbers are assigned in numeric order. If we specify an ID number in the `max_id` parameter, the search will begin with (and include) that ID. An ID number in the `since_id` parameter will limit our search to statuses after that particular status.

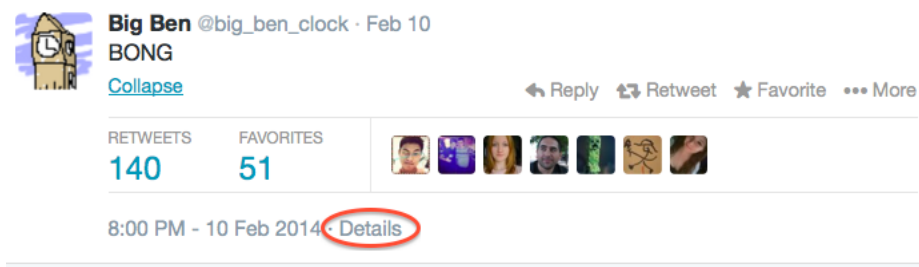


Figure 1: @big\_ben\_clock on 2/10/14 at 8 p.m. Click on "Details" link to find ID number in URL.

There are a number of Twitter "clock" accounts that tweet on the hour, so it is easy to find an ID number that represents a particular hour. Using the @big\_ben\_clock account, we find that on 2/10/14, 8 p.m. = 433042711546191872 and 9 p.m. = 433057813859155968.

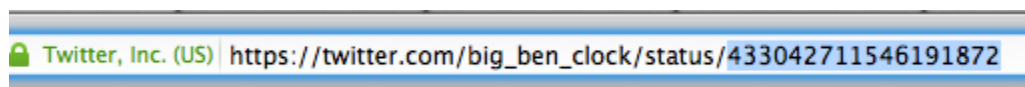


Figure 2: @big\_ben\_clock URL with status ID number on 2/10/14 at 8 p.m.

Plugging those numbers into our parameters gives us a search during that time period, starting with 9 p.m. and working back in time. Of course, since you are probably reading this outside of the 6-to-9-day window for Twitter search, you'll need to use more recent ID numbers. (Note that the times are precise only to the second.)

---

```
params["since_id"] = "433042711546191872"
params["max_id"] = "433057813859155968"
```

---

If you just want to search back from a particular status or point in time, only the `max_id` parameter is necessary. However, to continue to page back through the results, we'll need to continually adjust the `max_id` parameter.

### 3.4 Paging Back Through Results

In order to page back through the results, we need to change the `max_id` parameter value to a variable rather than a static value. Then, on each pass through the `for` loop, we can change the value of the `max_id` parameter so that the next API call will search back from the end of the previous search result.

Let's name the new variable `prev_id` and initialize it before the `for` loop. Note that the variable is converted from a string to an integer so that we can manipulate it mathematically. Then, set the value of the `max_id` parameter to a string of `prev_id`.

---

```
prev_id = int("435458631669415936")

for i in range(1):
    url = url1
    params["q"] = "pizza"
    params["count"] = 2
    params["geocode"] = ""
    params["lang"] = ""
    params["locale"] = ""
    params["result_type"] = "" # Example Values: mixed, recent, popular
    params["until"] = ""
    params["since_id"] = ""
    params["max_id"] = str(prev_id)
```

---

If you run the program now, the `print data` command will print a dictionary containing two keys: `search_metadata` and `statuses`. We want to find the ID number of the last (oldest) status. Since the value of `statuses` is a list, we can use our knowledge of indexing lists to find the last status: `data["statuses"][-1]`. Within that status, which is a dictionary, the ID number is identified by the key `id`. So, to find and print the value of the last status:

---

```
if data["statuses"] == []:
    print "end of data"
    break
else:
    print data["statuses"][-1]["id"]
```

---

We need to convert this value to an integer and subtract one before assigning it to the `prev_id` variable. Subtracting one will prevent the next API call from including the last status from the previous API call. We will also print the value of `prev_id` along with the

count from the loop (i). This allows us to observe the progress of the search.

---

```
if data["statuses"] == []:
    print "end of data"
    break
else:
    prev_id = int(data["statuses"][-1]["id"]) - 1
    print prev_id, i
```

---

Now we can save the results of each API call to a file. While we could append the results of each API call to the same file, I prefer to save each set of results separately, especially when returning large sets of results. This makes it much easier to deal with computer crashes, Twitter API errors, etc. Files will be numbered sequentially using the loop value (i), which we need to convert to a string in order to concatenate it with the file name.

Since Twitter limits searches by individual users to 180 API calls per 15-minute period, we'll also add a 5-second delay before the next iteration of the loop.

---

```
f = open("outfile_" + str(i) + ".txt", "w")
json.dump(data["statuses"], f)
f.close()
time.sleep(5)
```

---

In order to use the program, just edit it to include the appropriate API KEY, API SECRET, ACCESS TOKEN, ACCESS TOKEN SECRET, BEGINNING TWITTER ID, SEARCH QUERY, and count value.

You may edit other parameters as well, such as `params["lang"] = "en"` to limit results to statuses in English, `params["result_type"] = "recent"`, etc.

You may also want to rename the output file(s) to better reflect content. For example, if you conducted a search for pizza, adding the term “pizza” to the file name lets you know what search results are contained in the file.

---

```
f = open("out_pizza_" + str(i) + ".txt", "w")
```

---



## 4 The Completed Program

---

```
import oauth2
import time
import urllib2
import json

url1 = "https://api.twitter.com/1.1/search/tweets.json"
params = {
    "oauth_version": "1.0",
    "oauth_nonce": oauth2.generate_nonce(),
    "oauth_timestamp": int(time.time())
}

consumer = oauth2.Consumer(key="INSERT API KEY", secret="INSERT API SECRET")
token = oauth2.Token(key="INSERT ACCESS TOKEN", secret="INSERT ACCESS TOKEN
    SECRET")

params["oauth_consumer_key"] = consumer.key
params["oauth_token"] = token.key

prev_id = int("INSERT BEGINNING TWITTER ID")

for i in range(1):
    url = url1
    params["q"] = "INSERT SEARCH QUERY"
    params["count"] = INSERT INTEGER FROM 1 TO 100
    params["geocode"] = ""
    params["lang"] = ""
    params["locale"] = ""
    params["result_type"] = "" # Example Values: mixed, recent, popular
    params["until"] = ""
    params["since_id"] = ""
    params["max_id"] = str(prev_id)
    if data["statuses"] == []:
        print "end of data"
        break
    else:
        prev_id = int(data["statuses"][-1]["id"]) - 1
        print prev_id, i
    f = open("outfile_" + str(i) + ".txt", "w")
    json.dump(data["statuses"], f)
    f.close()
    time.sleep(5)
```

---