Paper SAS379-2014

Three Different Ways to Import JSON from the Facebook Graph API

Philihp Busby, SAS Institute

ABSTRACT

HTML5 has become the de facto standard for web applications. As a result, the lingua franca object notation of the web services that the web applications call has switched from XML to JSON. JSON is remarkably easy to parse in JavaScript, but so far SAS® doesn't have any native JSON parsers. The Facebook Graph API dropped XML support a few years ago. This paper shows how we can parse the JSON in SAS® by calling an external script, using PROC GROOVY to parse it inside of SAS®, or by parsing the JSON manually with a DATA step. We'll extract the data from the Facebook Graph API and import it into an OLAP data mart to report and analyze a marketing campaign's effectiveness.

INTRODUCTION

The Facebook Graph API offers a very rich interface for extracting marketing data, however pulling this data into SAS® requires a little bit of work. We've got go over SSL, obtain an OAuth access token, and use that to query JSON data from REST URLs.

SSL SECURITY

SSL encrypts communication between the client (your SAS® session) and the server (Facebook's servers). This is a Facebook requirement, which prevents session tokens from being hijacked when used on public Wifi networks, which was common before it was implemented (http://www.nytimes.com/2011/02/17/technology/personaltech/17basics.html). SAS® requires SAS/Secure® in order to use SSL, and once installed and configured, should appear no different from plain HTTP.

SSL makes use of public/private key cryptography in order to validate the source of transmissions. Server SSL certificates are signed by certificate authorities (CAs). Trusted root CA certificates are usually on your machine, and are used to validate these signatures and build a web of trust between the client and server; Usually this is setup by default, but if SAS® can't find the specific certificate your server is signed with, you'll need to tell it where these exist. You can do this by adding this line to your sasv9.cfg file, with the appropriate path:

```
-SSLCALISTLOC="/etc/ssl/certs/ca-certificates.crt"
```

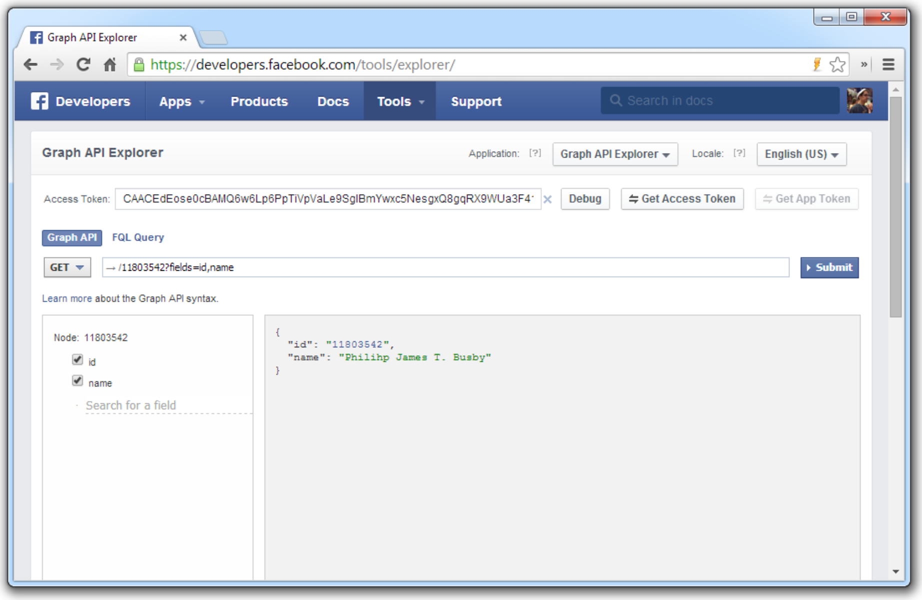
OAUTH TOKEN

Once we can talk to the Facebook server, we'll need an auth token. This token will be passed in to the server with every request that tells Facebook that

A. We are the app

B. We have access to the data we are querying

Getting a token is normally an interactive process requiring the user to actively authorize our app. It is described on the Facebook API's documentation[0], and is beyond the scope of this paper. Facebook offers a Graph API Explorer tool (https://developers.facebook.com/tools/explorer/), which we can use to get an auth_token for us.



Access Token: CAACEdEose0cBAMQ6w6Lp6PpTiVpVaLe9SglBmYwxc5NesgxQ8gqRX9WUa3F4

GET → /11803542?fields=id,name

Learn more about the Graph API syntax.

Node: 11803542

✓ id

✓ name

```
{
  "id": "11803542",
  "name": "Philihp James T. Busby"
}
```

Figure 1 Facebook Graph Explorer

Unfortunately, without being in a browser, getting a user-level access token from Facebook is not supported. It may be possible to get an app-level access token if the user has already authorized our app to their data, however the data these tokens have much more limited access to data and special care still needs to be taken to make sure that our app's secret key is not shared.

QUERYING FACEBOOK

The Facebook Graph API is a REST interface, so querying it is as simple as making HTTPS GET requests, and reading JSON data from it. The format of the page is typically limited to only a window of the available records in a collection, however the next page of the data is available at the end of the request.

NOTE ABOUT PATHS

The paths listed are intended to work on Linux, and on Ubuntu 13.10, installing groovy with sudo apt-get install groovy will install 1.8.6 into the stated path. If you are on Windows, put the `groovy-all.1.8.6.jar` file at the path `C:\usr\share\groovy\lib\` and SAS® on Windows will find it. Similarly, you will need to have a folder `C:\tmp\` in order to store intermediate files. As long as these two things exist, paths will resolve and the code will work cross-platform.

METHOD 1: PROC HTTP AND PROC GROOVY

The first method, which parses the JSON using Groovy, was explored on Jiangtang Hu's blog[1]. First, we will need an `access_token`, as described above. It should look something like this, all on one line

```
%let
token=CAACEdEosGOATaltLbHZA5Le2afZAnWDyg1letNLQ7ITsCi5nvzgxSIenOMtr6GJOrYu0ZCNms0rl
kFdA6QYx20QfQslFR0C3OjKDI5Iexxx6YuZBZBbjgQ47vqEDBniTM1NozAIb2F43G15VVtXzuXQ8p3ZB1ZB
5vIHAzWwERWcTt0LkZANss1O4LbueWgnOdftXWWnZCAq6QmAZDZD;
```

File 1. token.sas

The following groovy code could be kept inline with SAS® code, but this tends to confuse text editor code highlighters. Keeping it in a separate file alleviates this. Notice how it stores the next page in the "exports" map. This is a reserved map for SAS®, and all values saved to it will be accessible as macro variables by SAS® after the code executes.

```
import groovy.json.*
def input=new File('/tmp/http.json').text
def json=new JsonSlurper().parseText(input)
if(json.data.size > 0) {
  def output=new File('/tmp/groovy.csv')
  json.data.each {
    def row = [
               it.id,
               it.updated_time,
               '"'+it.message.replaceAll('"','""').replaceAll('\n',' ')+'"'
              ]
    output.append row.join(',') + '\n'
  }
  next = json.paging.next.replaceAll('&','%str(&)');
  exports.put('topull',next)
}
else {
  exports.put('topull','')
}
```

File 2. json2csv.groovy

Then, in the following file we include that and make an HTTP call to Facebook's API. With every response, we parse it and check to see if there's an additional page of data. If there is, we loop back around and pull that in as well. All of this is saved to a CSV file, which in the final step is imported into a SAS® dataset.

```
%include token;
data _null_;
  fname='fh';
  rc=filename(fname,'/tmp/groovy.csv');
  if(rc=0 and fexist(fname)) then
    rc=fdelete(fname);
  rc=filename(fname);
run;
filename csv '/tmp/groovy.csv';
data _null_;
  file csv;
  put 'id,updated_time,message';
run;
filename headers '/tmp/headers.txt';
filename out "/tmp/http.json";
data _null_;
  file headers;
  put "Authorization: OAuth &token";
run;

%macro pullpage();
%put TOPULL=&topull;
%if %length(&topull) ^= 0 %then %do;
proc http
  method="get"
  headerin=headers
  url="&topull."
  out=out;
```

```
run;
proc groovy;
  /* http://repository.codehaus.org/org/codehaus/groovy/groovy-all/1.8.6/ */
  add classpath="/usr/share/groovy/lib/groovy-1.8.6.jar";
  exec "json2csv.groovy";
quit;
%end;
%mend;
%macro puller();
%let topull=https://graph.facebook.com/11803542/statuses;
%do %while(%length(&topull) ^= 0);
  %pullpage;
%end;
%mend;
%puller;

data statuses;
  infile csv missover dsd firstobs=2 ;
  informat id $50. updated_str $24. message $400.;
  input id $ updated_str $ message $;
  updated_time = input(substr(updated_str,1,22)||':'||substr(updated_str,23),
                       e8601dz.);
  format updated_time datetime20.;
  drop updated_str;
run;
```

File 3. groovy.sas

When we run groovy.sas, it will create a CSV file, then make repeated calls to the Facebook API for a given REST query, in bold above. The Facebook API will *page* results, so it's necessary to check the result object for the next page of data. In the remainder of this paper when I pull code in python, I won't bother to do this, however I wanted to demonstrate it here.

There are two strategies for including the access_token. The most common is by appending it to the REST URL as a GET parameter. The lesser known way is to include it as an HTTP header. I prefer doing it this way here, since the URL for the next page of data won't have our access_token param included, and we would have to alter it to have that.

METHOD 2: PYTHON SCRIPT TO CSV

Another option is to push all of the HTTP and parsing completely out of SAS®. The following example does this using Python and the pyFaceGraph package[3].

```
import csv
from pprint import pprint
from facegraph import Graph
g = Graph("CAACEdltLbHZA5Le2a... access token here...LbudftXWWnZCAq6QmAZDZD")

with open('/tmp/python.csv', 'wb') as csvfile:
    writer = csv.writer(csvfile, delimiter=',',
                        quotechar="'", quoting=csv.QUOTE_MINIMAL)
    for friend in g.me.friends().data:
          writer.writerow([friend[k].encode('utf-8') for k in friend])
```

File 4. callfacebook.py

```
x 'python callfacebook.py';
filename csv '/tmp/python.csv';
data friends;
    infile csv missover dsd;
    informat name $200. id $25.;
    input name $ id $;
run;
```

File 5. python.sas

In the first line of python.sas, Python will be called and will run callfacebook.py and a CSV file generated. Then SAS® will import it.

METHOD 3: PIPE DIRECTLY FROM A COMMAND

This can be shortened further, to where our Python directly outputs CSV to SAS® using a filename pipe. While this is similar to the second method, it eliminates an intermediate step of putting a file on the system.

```
import csv
from pprint import pprint
from facegraph import Graph
g = Graph("CAACEdEose0cBACwgvcBpIBgKGbS...  access token  ...ZBzuhKy37dpNswZDZD")

for friend in g.me.friends().data:
    print ','.join([friend[k].encode('utf-8') for k in friend])
```

File 4. callfacebook.py

```
filename csv pipe 'python pullcsv.py';
data friends;
    infile csv missover dsd;
    informat name $200. id $25.;
    input name $ id $;
run;
```

File 5. python.sas

CONCLUSION

Presented here are three distinct methods of pulling in data from the Facebook Graph API. Unfortunately this can't be completely automated because of Facebook's OAuth implementation, and an access token needs to be generated before the code will run. Natively, SAS® doesn't currently include anything to parse JSON, so the parsing needs to be done in another language, either in Groovy or Python or the language of your choice.

REFERENCES

[0] https://developers.facebook.com/docs/facebook-login/access-tokens

[1] http://www.jiangtanghu.com/blog/2012/10/28/hello-groovy-in-sas-9-3/

[2] http://blogs.sas.com/content/sascom/2013/12/12/how-to-import-twitter-tweets-in-sas-data-step-using-oauth-2-authentication-style/

[3] https://github.com/colinhowe/pyFaceGraph

CONTACT INFORMATION

Your questions and comments are valued and encouraged. Please don't hesitate to contact the author at philihp.busby@sas.com, or visit his blog at http://philihp.com/

Code presented here is available at https://github.com/philihp/SGFPaper2014