Paper AD13

Using SAS® for Application Programming Interface Requests

Michael Jadoo

Economist in the Bureau of Labor Statistics

ABSTRACT

Application Programming Interface (API) is a method of requesting data and has been around for some time. Their primary uses are for front-end webs developers who wish to use these respective data sets for charts, tables, and maps. However, this method of data request can also be useful for data processing and analysis as fewer steps in the data products process can be achieved.

In the paper we will discover how an API request can remove steps in the data production process and how to make a request for data from a major statistical accounts producer. Moreover, some useful tips will be revealed when using this method.

INTRODUCTION

Secondary data users have a new option when collecting data sets for their analysis or production efforts. It is the application programming interface (API), which gives web developers and data analyst programmatic access to statistical agencies and company data products. This technology is not new but is a manifestation of similar methods of requesting and receiving information from other online sources. For the novice user, APIs can seem complicated to utilize and implement in support of their office mission. However, this paper should be able to remove barriers to understanding and access about this technology by providing information on three key points. These key points are; first, becoming knowledgeable about the data products you need to collect from an agency's API; second, constructing an API request using a URL; and third, turning the data table into a manageable format using SAS.

HISTORY OF THE API

APIs have been used with procedural languages since well before personal computing and were typically delivered as libraries (first used in the 1960's). The current API is also called Web API (used for data repository and tracking). According to some the modern Web API was officially branded with Roy Fieldings dissertation "Architectural Styles and the Design of Network-based Software Architectures" in 2000. The Web API's first appearance was found in the introduction to Salesforce.com on February 7th, 2000. A year later eBay launched their own API in 2001. Then in 2002 Amazon launched their own API (Apievangelist). Web API has been rapidly gaining popularity since 2010.[1]

[1] Frye, Alan. https://www.benefitfocus.com/blogs/design-engineering/bit-api-history, April 15, 2015 "A Bit of API History".

WHAT IS AN API?

Application Programming Interface is defined as a set of programming instructions for accessing information or services from a web-based software application. Companies make an API available to the public so that software developers can design products that can access its services.

The Windows API, for example, provides developers with user interface controls and elements, such as windows, scroll bars, and dialog boxes. It also provides commands for accessing the file system and performing file operations, such as creating and deleting files. Additionally, the Windows API includes networking commands that can be used to send and receive data over a local network or the Internet. While there are different types of APIs and some that are really advanced, the more basic example is when a website provides an API for developers to access information from their site. A website API just needs a set of commands for retrieving the desired data set (Christensson).

The information that is received usually comes in either through a JavaScript object notation (JSON) or extensible markup language (XML) format. Both JSON and XML file formats make it easier for humans and computers to read, separate and create data from the client driven requested information. In the diagram below a simple flow chart of an API request can be seen. The client or developer makes the request using their computer software interface. The request is then goes to the websites internal database. After the API obtains the data, it is sent back to the client or developer in either a JSON or XML file.

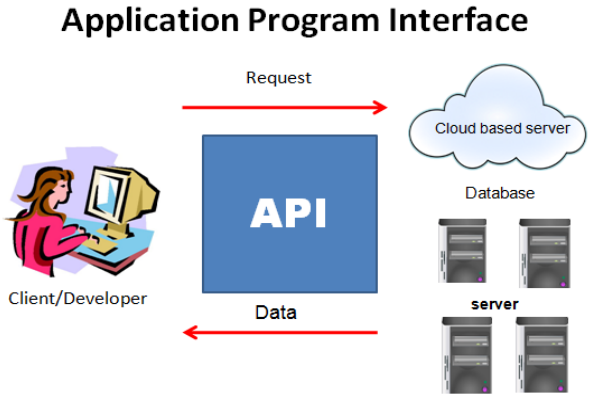

Figure 1. Flowchart of API request

HOW CAN USING API HELP DATA ANALYSTS?

Several steps must be undertaken when collecting and preparing data tables before the data can be used in charts, graphs, statistical models, or processing. Usually, when collecting data tables the analysts would need to go to a specific website or contact a statistical agency staff to retrieve the information required for their specific need. Once the data were collected they would be placed in a spreadsheet or database. These files would normally be in a CSV or Excel file and the charts and tables are then created in a desired software choice or used as input files to be used in a statistical computing software to be process, such as SAS. If that is the case, the data table would be stored in a directory so that it could be accessed by a SAS program when the time comes for data processing. The above process can be time consuming, also loading data into CSV or Excel files could result in human errors in processing.

When using SAS to access a website API the data is transmitted directly into the programmer's SAS session. After some initial data manipulation during the SAS session it can be stored as a permanent SAS data set or exported into another format for further processing or analysis. With this method of data

retrieval the programmer/analyst can focus more time on analyzing the data while minimizing data preparation time or costly mistakes.

BEST PRACTICES IN USING AN API

When the time comes for an analyst to decide to start using a website API to collect data there are some good approaches one should take before beginning.

The first best practice for using an API is to have a detailed understanding what data series the analyst is collecting. For example, the time frame of the data, the units and scale in which the data are measured, and if there is any unique identifier associated with the data that can easily be found. For example, specific lines out of a table can sometimes be manipulated and directly pulled into the SAS program if the analyst has the table and line number.

The second best practice is to obtain the API procedure user guides for developers. Some API data producers have manuals already prepared and have posted online tutorials either on their website or YouTube on how to access their data using API. These resources can help users tremendously.

The third best practice is to obtain a key code if the data provider requires one. An API key code is a unique identifying code that acts as a form of authentication about the user, about the data collected, and about any other important information concerning the request. There are public and private API keys. Public keys are usually free to obtain while private API keys normally has a standard fee. However, there are some websites APIs that do not require a key code at all. The key system is used as a protection for the data provider against a cyber-attack. By identifying unique users by their key code the data provider can disable the user account and restore web site functionality quickly. Most API applications will never bump into the data provider upper limit of web requests; however, some may restrict the number of queries.

Having to obtain an API key code could be a nuisance for some; however, on the data providers end a key code gives more security whenever a request is made for their data. Additionally, they can identify a specific request and the information that was collected. Also, key codes ensure that usernames and passwords are secure. Once the analyst has completed the three steps the API request can be constructed.

MAJOR DATA REPOSITORIES

There are a hundreds of data repositories that currently use API's. Some examples include Healthdata.gov, Census, NASA, FRED, the Bureau of Labor Statistics (BLS), the Bureau of Economic Analysis (BEA), the World Bank, and the Center for Disease Control and Prevention. These data repositories have all or most of their data in an API database. However, there are some series that may be in different increments (i.e., available in billions and not in millions of units) or may only be available through the web and not the API, so the values in each data series collected must be checked accordingly.

COLLECTING THE DATA

The rest of the paper will provide a how-to SAS demonstration on collecting and processing data using API request. The data set that was used was the BEA's energy quantity index data series for the

manufacturing sectors from the GDP by Industry Division. The SAS version used for this demonstration was the 9.4 TS1 maintenance 3 (M3).

It is important to note that the maintenance 4 (M4) version has an important update to SAS which allows users to easily import JSON files from an API request. It uses the SAS JSON libname engine. Once the data is retrieved into your SAS session the libname JSON engine will turn it into a usable SAS data set for further processing or analysis (Hemedinger).

To start, once completing the initial three steps above, create a FILENAME statement which will create a file that will store the information from the API request. The extension of the file should be a JSON for this exercise. Next, use the LRECL option to create the maximum recorder length for the file created- this is where knowing the data you expect to see is important to your code:

```
filename eqty "C:\PHARMSUG\EnergyQuantityIndex.json" lrecl=32000;
```

Next, place the URL request in the PROC HTTP. The HTTP procedure issues hypertext transfer protocol requests. You can submit input and receive output from a fileref (SAS). Also if a user name and password is given PROC HTTP can support basic authentication. For our example, the URL request has different parts separated by the main URL of the website, the API key code, and predicates. A predicate has guidelines for the URL regarding the type of data requested so that it can be easily found. Predicates always start with the ampersand, "&", symbol. Below is a short example of constructing an API request using a URL for the BEA GDP by industry accounts data:

- http://www.bea.gov/api/data/ - all API requests for BEA data must go through this main site first
- &UserID= - individual API key code obtained from the BEA site
- &method=GetData - gives the user the choice of what to display-the data or list of the data series
- &DataSetName=GDPbyIndustry - the program office that has the data
- &ParameterName=TableID -the way to identify how the data is named
- &ResultFormat=json -the output format of the request (either XML or JSON)

Once the URL is constructed place it in the URL option of the PROC HTTP statement. Next, place the file in the FILENAME statement at the OUT= option. Afterwards, specify the HTTP content-type (CT) to be application/json so that it can read as a JSON.

```
proc http
url='http://www.bea.gov/api/data/?&UserID=YOUR-USER-ID-
CODE&method=GetData&DataSetName=GDPbyIndustry&Frequency=A&Year=ALL&Industry=A
LL&tableID=31&ResultFormat=json'

 method='get' out=eqty
```

```
 ct="application/json";
run;
```

When you launch PROC HTTP the log file should give you a message saying “OK”. This is informing you whether the procedure step processed the information stored in the URL to the JSON file successfully.

```
1
2    filename eqty "C:\PHARMSUG\EnergyQuantityIndex.json" lrecl=32000;
3
4
5    proc http
6    url='http://www.bea.gov/api/data/?&UserID=01DB5A76-B431-45EA-8394-0A62
6  ! 47C7C3B5&method=GetData&DataSetName=GDPbyIndustry&Frequency=A&Year=ALL
6  ! &Industry=ALL&tableID=31&ResultFormat=json'
7
8    method='get' out=eqty
9    ct="application/json";
10   run;

NOTE: PROCEDURE HTTP used (Total process time):
      real time           0.87 seconds
      cpu time            0.03 seconds

NOTE: 200 OK
```

Display 1.Log file of PROC HTTP with BEA API request

After making the API request and the data the process to convert the JSON file into a usable SAS data set for either analyzing or processing begins. The JSON structure is read into SAS as one long observations and it must be parsed.

To begin the parsing process, read the JSON file into a SAS data step by using the INFILE statement with the DSD option, setting the record length to its maximum, and identifying the delimiters using the DLM option. In this example, the delimiters are the “[]”, “{}”, “,”, and “:”symbols.

Next, create a variable that contains the text (note that all the information starting in these tables will be text format) and identifies the common variables that are in the file (i.e. Data, notes, footnotes, etc.).

Afterwards, one needs to find a method of separating out the important information from the unnecessary items from the file. These types of information usually come after a common word or phrase. The IF /THEN conditional statements can identify the important information and remove unnecessary items from the data table by putting them into groups. In the example below the word “Data” comes before the important information within the file and the word “Notes” comes before the unnecessary items.

```
data temp;
 infile "C:\PHARMSUG\EnergyQuantityIndex.json" dsd lrecl=3000000
 dlm='[]{},:';
 input x : $2000. @@;
 retain flag;
 if x ='Data' then flag=1;
 if x='Notes' then group+1;
 if flag and x not in (' ' 'Data' 'Notes');
run;
```

Next, separate these groups while keeping only the important information.

```
data a b;
 set temp;
 if group=0 then output a;
  else output b;
run;

data a;
 set a;
 if mod(_n_,2)=1 then n+1;
 drop group flag;
run;
```

Now, begin to shape the data table by transposing it using PROC TRANSPOSE:

```
proc transpose data=a out=temp_a;
 by n;
 var x;
run;

data temp_a;
 set temp_a;
 if col1='TableID' then group+1;
 drop n _name_;
run;

proc transpose data=temp_a out=want_a(drop=group _name_);
 by group;
 id col1;
 var col2;
run;
```

Once the data set is transposed, clean the file of unnecessary variables from your new data table. Re-transpose to obtain the desired format for your data set.

After this is done convert the data values from character to numeric using the INPUT () function and use the DROP statement to remove the old variable:

```
/*change datavalue variable from character type to numeric */

data chgtype;
 set want a;
 data=input(datavalue,10.4);
 drop datavalue;
run;
```

Next, a DATA step is used to rename the industry names to its proper nomenclature for presentation.

```
proc sort data=chgtype;
 by  TableID industry IndustrYDescription   ;
run;

proc transpose data=chgtype out=want_a2;
 by TableID industry IndustrYDescription;
 id year;
 var data;
run;

data qexi ;
 set want_a2;
 where substr(industry,1,1)='3';
 prefix="qex";

/*change industry names to proper nomenclature*/

 drop _name_  TableID IndustrYDescription;
 if Industry in('31G','31ND','33DG') then delete;
 if industry='311FT' then industry='i311';
 if industry='313TT' then industry='i313';
 if industry='315AL' then industry='i315';
 if industry='321' then industry='i321';
 if industry='322' then industry='i322';
 if industry='323' then industry='i323';
 if industry='324' then industry='i324';
 if industry='325' then industry='i325';
 if industry='326' then industry='i326';
 if industry='327' then industry='i327';
 if industry='331' then industry='i331';
 if industry='332' then industry='i332';
 if industry='333' then industry='i333';
 if industry='334' then industry='i334';
 if industry='335' then industry='i335';
 if industry='3361MV' then industry='mve';
 if industry='3364OT' then industry='ote';
 if industry='337' then industry='i337';
 if industry='339' then industry='i339';
 var=cats(prefix,industry);
run;

/*rename cells and create a new variable that is the concatenation of the
PREFIX and INDUSTRY variables */

proc sort data=qexi out=qexi2(drop=industry);
 by industry;
run;

proc transpose data=qexi2 out=want_a4;
 id var;
 var _: ;
run;
```

```
data energy quantity;
 length year $ 4;
 set want a4;
  year=substr(_name_,2,4);
 drop _name_;
run;
```

At this point we have a data set that is ready to be stored for processing at a later time which is the ENERGY_QUANTITY data set.

CHALLENGES TO USING API

One major challenge to using API is that the user must be aware of the date when your data series is due for an update. If you are constructing a program to estimate values using inputs from an API source, in real time, then you don't have to worry about this situation.

However, if you are processing data tables for estimates on a periodic basis and you need input tables from a certain time frame or before the next data release it is important to remember the API pulls the data real time. Make sure to make the API request before the data tables are updated or you will lose the data set. Also, if you run your program after the website API has updated its table then you will save over your old data table and your estimates will be inaccurate.

To remedy this situation one can create a comment macro variable to be activated during a certain time period. Place the comment macro variable around the PROC HTTP step so that after the date when the new data has been uploaded to the API database this PROC step won't run along with the rest of the program and overwrite the old data that was collected previously.

The comment macro variable is created using a conditional statement based on date values. For example, if you are tasked to collect a data set within January 27, 2017 and February 28, 2017. Then use the built-in SYSDATE macro in a conditional statement. Whereas, when today's date is between January 27, 2017 and February 28, 2017 the macro variable CMT will be blank and won't create a comment create a comment (*) so that the PROC HTTP can run. Conversely, when the condition is not met then the macro variable CMT will create a comment (*) so that the PROC HTTP won't run and save over the old data that was collected. Some data providers also provide release dates in an API format from which the client can request the date of the release data are consistent.

```
data _null_;
  if "27jan17"d <= "&sysdate"d <= "28feb17"d
        then do; call symput('cmt',' '); end;

           else do; call symput('cmt','*'); end;
run;
```

The comment should be placed around the PROC HTTP step. By doing this one removes the chance of saving over your old data set.

```
filename invt536
"M:\SAS_Production\SAS_Production_&last_year\Preliminary\Input\API_INV
EST536.json" lrecl=32000;

/*using the CMT macro variable to remove this PROC step before the BEA
```

```
Gross Domestic Product, 4th quarter and annual (second estimate) comes
out, as not to over write the advance estimate collection */

&cmt proc http  url='http://www.bea.gov/api/data/?&UserID=YOUR-USER-ID-
CODE&method=getdata&year=X&frequency=A&DataSetName=NIPA&TableID=146&Res
ultFormat=json'
       method='get' out=invt536
       ct="application/json";
&cmt run;
```

CONCLUSION

In conclusion, SAS can be used to effectively obtain data tables from a website API. However, understanding how to construct an API request is the major part of what a SAS programmer needs to do. As well as being able to parse and wrangle the data set into a usable format. It is important that users become knowledgeable about the information being requested as this information is needed to identify the data from the API source, to construct the API request, and turn the data table into a manageable format to be used in the office using SAS.

REFERENCES

Apievangelist. “History of APIs .” 20 December 2012. Web. 29 March 2017<http://history.apievangelist.com/ >

Christensson, Per. "API Definition." *TechTerms*. Sharpened Productions, 20 June 2016. Web. 03 March 2017. <https://techterms.com/definition/api>.

Hemedinger, Chris. “Reading data with the SAS JSON libname engine.” The SAS Dummy. 06 December 2016. Web. 03 March 2017. <http://blogs.sas.com/content/sasdummy/2016/12/02/json-libname-engine-sas/>

SAS. “HTTP Procedure.” Base SAS(R) 9.4 Procedures Guide, Sixth Edition. Web. 03 March 2017. <http://support.sas.com/documentation/cdl/en/proc/69850/HTML/default/viewer.htm#n0t7v16eitluu2n15ffpfeafqszs.htm>

ACKNOWLEDGMENTS

Special thanks to Ksharp, KurtBremser, and mkeintzz, from the SAScommunity.org site for their support.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Michael Jadoo
Mike_jadoo@yahoo.com