Paper SAS2120-2018

Web Metrics at Scale: Using Base SAS® to Access Google Analytics APIs

Chris Hemedinger, SAS Institute Inc., Cary, NC

ABSTRACT

With SAS® 9.4M4 and later, it's finally easy (relatively speaking) to connect to complicated APIs like those supported by Google, and to gather information with an unattended batch process.
The task is made possible by recent enhancements in the HTTP procedure and the new JSON library engine in SAS®. The PROC HTTP enhancements make it easier to negotiate multi-step authentication schemes like OAuth 2. And the JSON engine makes it easier to parse JSON results into SAS data sets. Previous approaches relied on complicated techniques such as the GROOVY procedure or other tricks to call outside of SAS to drive these OAuth2 APIs and parse JSON. The recent enhancements make such tricks unnecessary and thus provide a cleaner approach with fewer moving parts.
In this paper, I describe the four main steps needed to use Google APIs to gather web metrics data. The paper includes SAS code that you can adapt and use immediately with your own Google Analytics environment. I also present techniques that you can generalize to access any REST API that relies on OAuth2 as its authentication mechanism.

INTRODUCTION

Google Analytics is the de facto standard tool for monitoring and reporting on website activity. In exchange for allowing Google to collect visit and visitor information about your website, you can use Google Analytics to track almost any type of website metrics. Counts of page views and site visits, sometimes called *vanity metrics*, are the most basic levels of data that you can collect. However, Google Analytics offers access to more sophisticated insights, including time-on-site, referring pages or sites, geographic and demographic data for visitors, tracked progress against goals for registering or purchase (called *conversions*), and much more.

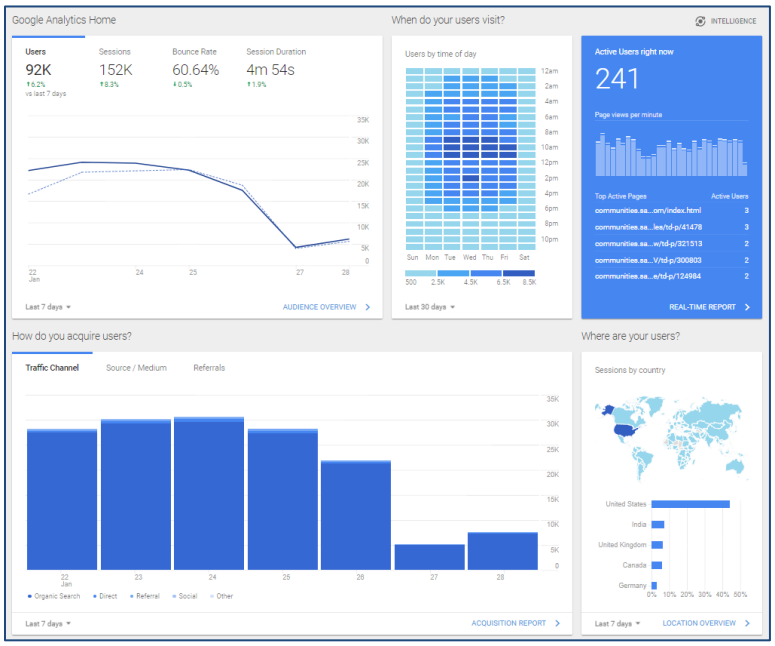

Figure 1. Example of a Google Analytics Dashboard

Google Analytics offers a rich dashboard and reporting tool, as shown in Figure 1. You can view the metrics that matter to you, and drill into activity across many dimensions including time, content, and search keywords. The tool includes useful charts such as trendlines, geographic maps, categorical

charts, and many more. Once you navigate to the view that you want, you can export the data into Microsoft Excel or comma-separated-value (CSV) format for easy use elsewhere.

WHY USE SAS TO ACCESS GOOGLE ANALYTICS DATA

If Google Analytics includes such a rich reporting tool, why use SAS at all? Here are a few advantages:

- Google Analytics is a "webmaster" tool, intended for the website owner or for a small team of experts who manage the site. It requires special knowledge and skills to use it effectively. And access must be controlled by a site administrator, a task that doesn't scale to more than a handful of users.
- You can extract the data from Google Analytics and combine it with in-house data to enrich the reporting you already do. For example, at SAS we produce operational reports about our blogs, which contain thousands of articles written by hundreds of authors. These authors can now see page views, time-on-page, and other accessible measures about the posts that they've written.
- Automation! By using application programming interfaces (APIs), you can extract Google Analytics data regularly with an unattended process, add it to your own data mart, and track the changes over time. At SAS, we use the Google Analytics APIs to extract the daily metrics about blogs.sas.com. We have blog performance data going back nearly four years (Figure 2), which helps us to create reports that inform our blogging strategies and successes.

OLDER (>1 year) blogs.sas.com posts with >300 views in past 30 days (as of 29JAN2018)

Obs	Published	Views (30 days)	Article	Blog
1	2011-09-07	5,483	Loops in SAS	iml
2	2015-05-01	4,675	Converting variable types—use PUT() or INPUT()?	sgf
3	2012-01-25	3,805	Export to Excel 2010 just got a little bit easier	sasdummy
4	2011-10-03	3,228	Rounding up, rounding down	iml
5	2011-04-27	3,005	Log transformations: How to handle negative data values?	iml
6	2013-10-29	2,629	5 keyboard shortcuts in SAS that will change your life	sasdummy
7	2012-02-11	2,148	How do I export from SAS to Excel files: Let me count the ways	sasdummy
8	2015-07-17	2,102	Customizing output from PROC MEANS	sgf
9	2017-01-18	2,009	Where does Girl Scout cookie money go?	sastraining
10	2011-09-19	1,948	Count the number of missing values for each variable	iml

Figure 2. Example Report about Blog Articles with Historical Data

In short, the Google Analytics website offers good tools for ad hoc reporting and analysis, meant to be accessed by only a few experts. Use SAS to ingest the most relevant measures into your reports for broader communication to your stakeholders.

SMOKE TEST: HTTP PROCEDURE AND THE JSON LIBRARY ENGINE

The SAS code shared in this paper uses only Base SAS procedures and statements. No additional products are necessary. There are two special requirements that you should verify before proceeding:

- You can access internet resources using PROC HTTP, including sites that are secured with Secured Sockets Layer (SSL), the encrypted protocol used for HTTPS sites.
- You can access the JSON library engine, added in SAS 9.4M4.

If you can run the following SAS program with no errors, then you have the basic requirements necessary to use the Google APIs (and many other REST-based APIs):

```
filename resp "%sysfunc(getoption(WORK))/now.json";
proc http
 url="https://now.httpbin.org/"
 method="GET"
 out=resp;
run;

/* Tell SAS to parse the JSON response */
libname time JSON fileref=resp;

title "Raw values from the JSON response";
proc print data=time.now (drop=ord:);
run;
```

The program uses a free HTTP echo service to verify that you can access secure HTTP sites, retrieve a JSON response, and then parse the JSON content with the new library engine.

Many companies route their internet traffic through proxy servers. This is usually handled behind the scenes for web browsers, but scripting tools require more explicit configuration. If PROC HTTP causes an error when you test this program, you might need to specify the PROXYHOST options, the PROXYPORT options, or both to tell SAS how to direct the traffic. If you don't know these values, ask your corporate network administrator for help.

If you don't have SAS 9.4M4 or later, you can prototype the techniques in this paper using SAS University Edition (beginning with the December 2017 release). It is free to download and use for learning purposes.

HOW TO USE THE GOOGLE APIS

There is much to learn about how Google APIs work and how to provision your Google account with access to resources like Google Analytics. In this paper, I describe the general steps for creating an API project and obtaining credentials. Your exact steps might be different from this, as it depends on how Google Analytics is administered for your site. For the detailed steps within this paper, I'm going to assume that you already have an API project created, and that you have a *client ID* and *client secret* value that is unique to your project. The Google API documentation is very good on this topic, and there are many internet resources to help you learn this part of the process. And, since it is software-as-a-service and not a static process, the exact steps are subject to change. My hope is that the general steps described in this paper are serviceable for years to come.

Some Google APIs are free to use by any user, while others cost money. And some are rate-limited (meaning that you can call the APIs only so many times within a given period) or have capped daily quotas. Your specific limits depend on your level of service with Google. If you have a corporate relationship with Google, you might need an administrator to provision your specific Google account to get access to the APIs.

ABOUT GOOGLE API PROJECTS AND CREDENTIALS

To use Google APIs, you first need a Google account and a developer project. If you're already working with Google Analytics, you must have already established a Google account. So that's one item we can mark done.

A project is a named set of capabilities and permissions that you establish before you can access the APIs that you need. To create a new project, visit Google's developer console site at https://console.developers.google.com.

If you're working with a larger team, including someone who manages your Google Analytics access, you might need another person with the proper access to create the project for you and invite you as a member.

With a project established, you can then create the credentials that you need to access the APIs from code. The Google API framework offers different types of credentials for different types of applications. For an application driven by SAS code, you need an OAuth client ID (Figure 3).

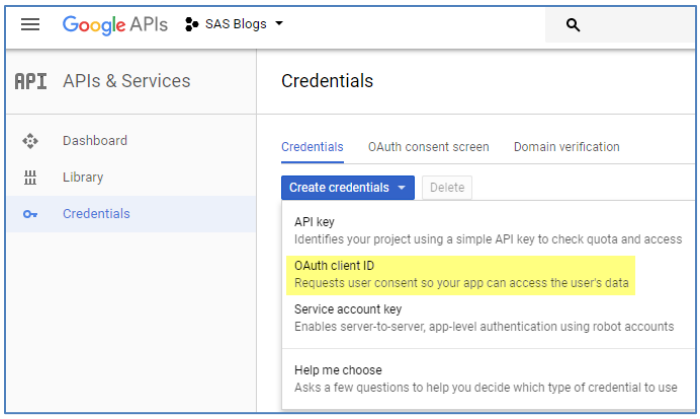

Figure 3. Create Credentials in Google Developer Console

The OAuth client ID provides you with two important keys for accessing the APIs: a *client ID* and a *client secret* (Figure 4). Copy these values to another file for reference, but keep them secure! These two items are private to your application. Any person or application that has these keys can access the APIs, and your data, as if they were *your application,* working against your quotas and privileges.

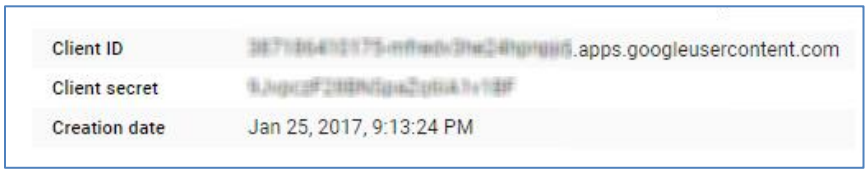

Figure 4. OAuth Client ID and Client Secret in the Developer Console

USING THE GOOGLE APIS IN FOUR STEPS

After you obtain your OAuth client ID (and secret), there are four main steps to use the Google APIs. The first two of these steps are performed as project setup, and they need to be performed just once. The last two steps are performed each time you access a Google API from a program, and they can be scripted. The steps are as follows:

1. Grant permission and obtain an authorization token for your app (a SAS program, in our case). This must be performed in a browser while logged on to your Google account.
2. Obtain an access token and refresh token. You can accomplish this with a SAS program that you run just once. You then save the refresh token (in a secure location!) for subsequent runs.
3. Convert your saved refresh token into an access token. Do this at the start of every SAS job that needs to get Google Analytics data.
4. And finally, use the Google Analytics API to get actual data!

The details of each step are described in the following sections.

STEP 1. GRANT PERMISSION AND RETRIEVE AUTH CODE

Do this step only once per account. You perform this step in the browser while logged into your Google account.

1. Enter the following URL, substituting your client ID as indicated. The URL needs to be all on one line, but I've broken it up here for readability.

```
https://accounts.google.com/o/oauth2/v2/auth?
        scope=https://www.googleapis.com/auth/analytics.readonly
        &redirect_uri=urn:ietf:wg:oauth:2.0:oob
        &response_type=code
        &client_id=<your-client-id>.apps.googleusercontent.com
```

2. You will be prompted to allow the app access to your Google Analytics data (Read only). This is like allowing another app to access your Facebook or Twitter profile. You've probably seen similar prompts during your own internet citizenship. Permissions are an important component of allowing apps to act on your behalf via REST APIs.
3. Then, you'll be redirected to a web page with an auth code that you should copy and save. We'll use it in the next step.

Note that setting the proper value for **redirect_uri** in this URL is very important: **redirect_uri=urn:ietf:wg:oauth:2.0:oob**. Otherwise, the API won't generate a code that you can copy and use in a tool-based app like SAS.

STEP 2. EXCHANGE THE AUTH CODE FOR AN ACCESS TOKEN

1. **Run this PROC HTTP step in SAS with the POST method to exchange that auth code from Step 1 for an access token:**

```
/* file to store your result */
filename token "c:\temp\token.json";
%let code_given =<code-returned-from-step-1> ;
%let oauth2=https://www.googleapis.com/oauth2/v4/token;
%let client_id=<your-client-id>.apps.googleusercontent.com;
%let client_secret=<your-client-secret>;
proc http
/* put this all on one line! */
 url="&oauth2.?client_id=&client_id.%str(&)code=&code_given.
 %str(&)client_secret=&client_secret.%str(&)redirect_uri=urn:ietf:wg:oauth:2.0:oob
      %str(&)grant_type=authorization_code%str(&)response_type=token"
 method="POST"
 out=token
;
run;
```

It will return a JSON response with a valid Bearer access token. That token expires in 3600 seconds (1 hour). It also returns a *refresh token*, which you can exchange again for a new access token after the first one expires. The refresh token never expires (although it can be revoked via the developer console or API). Thus, you usually need to perform this step just once, unless your token is revoked for some reason.

2. **Ensure that the refresh token and *client-id* and *secret* values are protected! Anyone who has access to these can access your Google API data as if they were you. Consider storing them in a file that only you have Read access to, and programmatically pull them in when running your SAS program under your host account.**

STEP 3. EXCHANGE THE REFRESH TOKEN FOR A VALID ACCESS TOKEN

Typically, you'll include this step just once at the beginning of your SAS job. This takes your saved refresh token value and asks Google to grant an access token for use in the rest of your program. The program code here parses the response with the JSON library engine, and stores the access token in a SAS macro variable (using CALL SYMPUTX):

```
/* STEP 3. Do this every time you want to use the GA API */
/* Turn in a refresh-token for a valid access-token      */
/* Should be good for 60 minutes                         */
/* So typically run once at beginning of the job.        */
%let oauth2=https://www.googleapis.com/oauth2/v4/token;
%let client_id=<your-client-id>.apps.googleusercontent.com;
%let client_secret=<your-client-secret>;
%let refresh_token=<refresh-token-from-step-2>;

filename rtoken temp;
proc http
 method="POST"
 /* Again, put this all on one line */
 /* broken here for readability */
 url="&oauth2.?client_id=&client_id.
     %str(&)client_secret=&client_secret.
     %str(&)grant_type=refresh_token%str(&)refresh_token=&refresh_token."
 out=rtoken;
run;

/* Read the access token out of the refresh response  */
/* Relies on the JSON libname engine (9.4m4 or later) */
libname rtok json fileref=rtoken;
data _null_;
 set rtok.root;
 call symputx('access_token',access_token);
run;
```

The Google APIs run very fast, so you should not need to renew the token again within the same job. You should be able to rely on this access token for all of your API queries in this session.

STEP 4. USE THE GOOGLE ANALYTICS API TO GATHER DATA

Finally, we're to the point where we can retrieve data from this service! Each call to the API uses the HTTP GET method to retrieve data. The SAS code has the following format:

```
filename garesp temp;
proc http
  url="<REST-api-url-call>"
  method="GET" out=garesp;
  /* Headers statement makes this easy */
  headers
    "Authorization"="Bearer &access_token."
    "client-id:"="&client_id.";
 run;
```

The PROC HTTP HEADERS statement was enhanced in SAS 9.4M3 to allow for inline name-value pairs. For this application, we need to inject the “Authorization: Bearer *access_token*”, which is a standard for OAuth2-compliant APIs. We’re also passing in the *client-id* value that identifies our application.

Designing your Query in the Google Developer Console

In this style of REST API, all details of the request are encoded as parameters on the URL value. This makes for very long URL strings that include values for the following:

- dimensions (or categories) of data
- metrics requested
- date range
- other API directives

You must specify an identifier on the URL for each dimension and measure. Here are a few examples:

Description	GA identifier
Number of sessions	ga:sessions
Number of page views	ga:pageViews
Page title	ga:pageTitle
Time on page	ga:timeOnPage
Audience gender	ga:userGender
Audience age range	ga:userAgeBracket

Google Analytics can report on only specific combinations of measures and dimensions, so it is important to understand how the possible values relate to each other.

> ***Caution***: You can use Google Analytics to request summarized data values, including averages and percentages (for example, "average time on page"). If your goal is to collect this data for further reporting in SAS, including further aggregation, be careful. You don't want to accidentally calculate an average of averages without also including a proper weight factor. For our project at SAS, we collect the data at the smallest detail level that we need, and then use SAS to perform all aggregated summaries.

You can explore, build, and test the requests by using the Google Developer Console and API Explorer. The developer console presents a form that allows you to build a request by supplying values for the required and optional fields (Figure 5). Each API method has different parameters, and the developer console presents a useful approach for designing your query. It's important to get your query working properly in the console before you try to script it; this can save you time in debugging API syntax problems later.

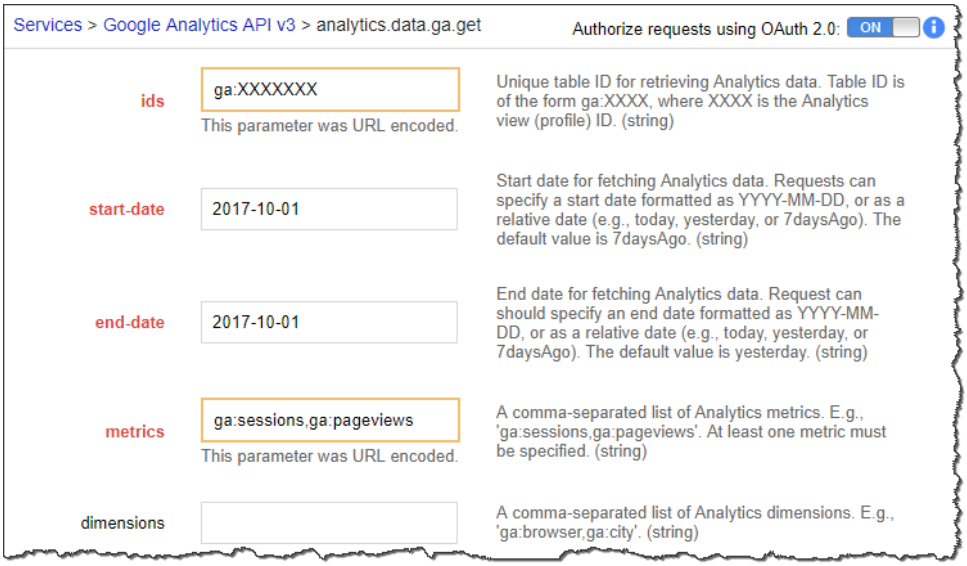

Figure 5. Google Developer Console Form using the analytics.data.ga.get Method

Figure 5 shows an example of the API Explorer for the GET method. When the GET method is executed, the console creates this HTTP request:

```
GET https://www.googleapis.com/analytics/v3/data/ga?ids=ga%3AXXXXXX&start-
date=2017-10-01&end-date=2017-10-01&metrics=ga%3Asessions%2Cga%3Apageview
```

(I've masked out the actual Google Analytics "view ID" as ga:XXXXXX; a valid view ID is usually a number, and represents your defined data view within Google Analytics.)

Transcribing the API Query into Your SAS Program

Notice that because these query details are transmitted on the URL, we need to encode (that is, replace with hexadecimal values) any characters that are not valid for URL values. These include commas, colons, spaces, and more. Thus, "ga:pageViews" becomes "ga%3Apageviews". We can use the URLENCODE function to handle this for us.

The following code snippet shows how to build the URL piecemeal, using SAS macro variables and functions:

```
%let workdate='01Oct2017'd;
%let urldate=%sysfunc(putn(&workdate.,yymmdd10.));
%let metrics=%sysfunc(urlencode(%str(ga:pageviews,ga:sessions)));
%let id=%sysfunc(urlencode(%str(ga:XXXXXX)));
filename garesp temp;
proc http
  url=" url="https://www.googleapis.com/analytics/v3/data/ga?ids=&id.%str(&)start-
date=&urldate.%str(&)end-date=&urldate.%str(&)metrics=&metrics.%str(&)max-
results=20000"
  method="GET" out=garesp;
  headers
    "Authorization"="Bearer &access_token."
    "client-id:"="&client_id.";
 run;
```

After this program successfully completes, the **garesp** fileref contains the JSON response. Now that we have what we need from the Google Analytics API, we can turn our attention to parsing the response into a useful data set.

WORKING WITH THE GOOGLE ANALYTICS RESPONSE DATA

JSON is simply a text representation of data: a collection of name-value pairs in a hierarchical structure. The schema of that structure is defined by the application. While it's possible to use basic SAS functions to parse JSON, the JSON library engine makes the task much easier. To get started, assign a new SAS library and reference the JSON fileref:

```
libname gadata json fileref=garesp;
```

The JSON library engine dissects the data in the file into several tables. If you're just getting started with the JSON structure from a new API, it's a good idea to explore these tables to learn which tables contain the values you need, and how the different tables relate to each other. By default, the JSON engine creates highly normalized tables and generates ordinal keys that allow you to join the rows together for analysis and reporting.

For the Google Analytics response data, we have just two tables that are important to us: COLUMNHEADERS and ROWS.

The COLUMNHEADERS table is like an ordered directory of the columns that you can expect to find in the detailed data portion of the response (the ROWS table). Here's the contents of a COLUMNHEADERS table for a request that asked for 5 metrics across 2 dimensions.

ordinal_columnHeaders	name	columnType	dataType
1	ga:pagePath	DIMENSION	STRING
2	ga:pageTitle	DIMENSION	STRING
3	ga:pageviews	METRIC	INTEGER
4	ga:uniquePageviews	METRIC	INTEGER
5	ga:timeOnPage	METRIC	TIME
6	ga:entrances	METRIC	INTEGER
7	ga:exits	METRIC	INTEGER

That columns directory is important, because you will need to reference it as you process the detail data in the ROWS table. The column headers in ROWS are labeled "element1", "element2", "element3", and so on, and the data values are all represented as character values. By using the schema information in COLUMNHEADERS, which includes the data type and column name, you'll be able to apply DATA step logic to convert the data into values that can be used for analysis. See Figure 6 for an example of the ROWS data annotated with the column names from COLUMNHEADERS:

			uniquePageViews				
pagePath	pageTitle	pageViews		timeOnPage	entrances	exits	
element1	element2	element3	element4	element5	element6	element7	
blogs.sas.com/content/academic/2012/11/30/student-...	Student resear...	1	1	0.0	1	1	
blogs.sas.com/content/academic/2013/06/14/learn-ho...	Learn how to u...	2	2	0.0	2	2	
blogs.sas.com/content/academic/2013/07/10/next-sto...	Next stop, grad...	1	1	0.0	1	1	
blogs.sas.com/content/academic/2013/09/20/congratu...	Congratulation...	1	1	0.0	1	1	
blogs.sas.com/content/academic/2015/04/24/congratu...	Congratulation...	1	1	0.0	1	1	
blogs.sas.com/content/academic/2016/04/04/mind-the...	Mind the gap:...	1	1	0.0	1	1	
blogs.sas.com/content/academic/2016/09/01/the-well-...	The well-equip...	1	1	49.0	0	0	
blogs.sas.com/content/academic/2016/10/19/working-...	Working with th...	2	2	0.0	2	2	

Figure 6. Detail Data in Generic Character Form, Annotated with Column Names

Using the information from these two tables, we can create a DATA step that converts the character data into proper values that we can analyze, assigning meaningful column names along the way. Here's an example DATA step, followed by the final data set that is ready for reporting (Figure 7):

```
data work.ga_daily (drop=element:);
    set gadata.rows;
    drop ordinal_root ordinal_rows;
    length  url $ 300 title $ 250
           views 8 unique_views 8
           time_on_page 8 entrances 8 exits 8;
    url = element1;
    title = element2;
    views = input(element3, 5.);
    unique_views = input(element4, 6.);
    time_on_page=input(element5, 7.2);
    entrances = input(element6, 6.);
    exits = input(element7, 6.);
run;
```

url	title	views	unique_views	time_on_page	entrances	exits
blogs.sas.com/content/academi...	Student researche...	1	1	0	1	1
blogs.sas.com/content/academi...	Learn how to use...	2	2	0	2	2
blogs.sas.com/content/academi...	Next stop, graduat...	1	1	0	1	1
blogs.sas.com/content/academi...	Congratulations A...	1	1	0	1	1
blogs.sas.com/content/academi...	Congratulations S...	1	1	0	1	1
blogs.sas.com/content/academi...	Mind the gap: Add...	1	1	0	1	1
blogs.sas.com/content/academi...	The well-equipped...	1	1	49	0	0
blogs.sas.com/content/academi...	Working with the...	2	2	0	2	2
blogs.sas.com/content/academi...	Learn SAS on you...	12	11	163	11	11
blogs.sas.com/content/academi...	Teachers can brin...	2	1	143	1	1
blogs.sas.com/content/academi...	Tips for internship...	1	1	42	0	0
blogs.sas.com/content/academi...	Interning at SAS:...	4	3	620	3	2
blogs.sas.com/content/academi...	Interning at SAS:...	2	2	658	0	1
blogs.sas.com/content/academi...	Inspiring students...	4	3	540	3	3
blogs.sas.com/content/all-posts/	All Posts - SAS Bl...	24	16	1226	6	8

Figure 7. Google Analytics Data with Proper Formatting

CONCLUSION

The Google Analytics API is complex, but powerful. It's also well designed and well documented, and Google provides the essential tools to prototype your queries and preview your results. PROC HTTP provides all of the function you need to navigate the API successfully. And the JSON library engine provides a natural method for SAS programmers to work with the results.

If you can master working with SAS and the Google Analytics API, it should be an easy task to transfer these skills to using other REST APIs with SAS. In my work, I use SAS processes with several different cloud-based services. Each is a little different, but all require multiple rounds of authentication and the processing of JSON responses.

REFERENCES

Hemedinger, Chris. 2015. "How SAS Uses SAS to Analyze SAS Blogs." *Proceedings of the SAS Global Forum 2015 Conference*. Cary, NC: SAS Institute Inc. Available at https://support.sas.com/resources/papers/proceedings15/SAS1708-2015.pdf.

Henry, Joseph. 2017. "Show Off Your OAuth." *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. Available at http://support.sas.com/resources/papers/proceedings17/SAS0224-2017.pdf.

Hemedinger, Chris. "Using SAS to access Google Analytics APIs." *The SAS Dummy blog*. https://blogs.sas.com/content/sasdummy/using-sas-to-access-google-analytics-apis/. Last modified April 14, 2017.

Hemedinger, Chris. "How to test PROC HTTP and the JSON library engine." *The SAS Dummy blog*. https://blogs.sas.com/content/sasdummy/check-json-and-http/. Last modified January 23, 2018.

Hemedinger, Chris. "How to secure your REST API credentials in SAS programs." *The SAS Dummy blog*. https://blogs.sas.com/content/sasdummy/hide-rest-api-tokens/. Last modified January 16, 2018.

ACKNOWLEDGMENTS

Thank you to Mike Tormey at SAS. As our Google services administrator, Mike trusted me enough to grant access to the Google Analytics APIs for our corporate sites. I appreciate the opportunity to experiment, learn, and deploy the Google Analytics data into our blog reporting operations. Thank you also to Brandy Mann, who operates the SAS blogging program. Brandy has integrated the Google Analytics data into many of her stakeholder communications. Her feedback and requests have helped me to make this as useful as it can be.

Thank you to the many people who read my blog post about this topic and attempted to follow my instructions. Your validation that “it all worked” provided the confidence I needed to refine those details and develop this paper.

CONTACT INFORMATION

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

Chris Hemedinger
SAS Institute Inc.
Chris.Hemedinger@sas.com
The SAS Dummy blog (https://blogs.sas.com/content/sasdummy)