Paper 3156-2019

SAS® Viya® reportImages Service: The Report Optimization Speedometer

Michael Drutar, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® Viya® offers several techniques that can maximize the speed of SAS® Visual Analytics reporting: data partitioning, user-defined formats, and the use of aggregated data. However, every SAS Visual Analytics report can be different: different data, different graphs, and other differences in terms of filters, interactive widgets, and more. Testing how changes to individual reports affect speed can be laborious and might involve manually opening reports in the SAS® Report Viewer several times and meticulously reviewing each report's diagnostics or microservice logs. Even with this information, external factors such as network performance can confound the diagnostics. This paper presents a programmatic way to call a SAS Visual Analytics report to quickly determine how long it takes the report to render using the reportImages service, available via the SAS Viya REST API. This paper provides all of the code for an automated, end-to-end process that leverages the SAS Viya REST API to retrieve the server-side render time of a SAS Visual Analytics report. Code is provided for testing an individual report on demand. This process can be repeated automatically while the report designer tests several versions of the report. Macro code demonstrates how to test a suite of reports for comprehensive A/B comparisons. Data gathered from these repeated API calls enables designers to quickly determinate the best performance techniques to meet their specific reporting needs.

INTRODUCTION

SAS Viya includes many REST APIs that enable developers to interact with the SAS® platform in new and exciting ways. Before using these APIs, developers should become familiar with basic REST concepts, such as endpoints, methods, headers, and content that can be contained within the body of a request. This paper assumes that you have a general understanding of using PROC HTTP to make REST API calls using these principles.

ACCESSING SAS VIYA SERVICES USING SAS® STUDIO 5.1

Before clients can leverage the many assets that SAS Viya REST APIs provide, they must first be authenticated using an access token. Methods for how to acquire access tokens vary, based on the language and method that clients use to interact with the SAS Viya REST APIs.

When logged on to the SAS Viya platform and submitting SAS code using SAS Studio 5.1, developers can connect to the SAS Viya services by placing the keyword *sas_services* in the PROC HTTP option *oauth_bearer*. This ensures that the access token for the current authenticated user is automatically obtained and attached to the request, thus enabling SAS programmers who are already logged on to the SAS Viya environment to seamlessly connect with the resources that are available in SAS Viya services. Details on the *sas_services* keyword and *oauth_bearer* option are beyond the scope of this paper; however, more information can be found in the SAS documentation in the References section of this paper.

The example SAS code for achieving this access is below:

```
* Base URI for the service call;
%let BASE_URI=%sysfunc(getoption(servicesbaseurl));
```

```
/*Assign filename for output*/
filename reports temp;

/*Call reports Service*/
proc http url="&BASE_URI/reports/"
     method='get'
     oauth_bearer=sas_services
     out=reports;
run;
```

This paper assumes that all code is submitted in a SAS Studio 5.1 session within a SAS Viya 3.4 (and beyond) environment that contains the SAS Viya services being called.

REPORTIMAGES SERVICE OVERVIEW

The *reportImages* service can be used by clients to create an SVG image of a SAS® Visual Analytics report. The process flow for using this service consists of two steps. Each step involves the client making a request to the service and the service sending a response. The requests and responses for these two steps are outlined below:

- Step 1
  - The client requests that a new job be created, which generates an SVG image of a specific SAS Visual Analytics report.
  - The service initiates the job and sends a response to the client containing information about the job (such as the job's ID).
- Step 2
  - The client requests the status of the job using the job ID returned from Step 1.
  - The service sends a response to the client containing the current status of the job specified in the request.

When Step 2 returns a state of *completed* for the specified job, additional information is also contained in the response. Specifically, the response contains the duration the job took to generate the SVG image. After the service returns a *completed* status, the client can use other features of the *reportImages* service to retrieve the generated SVG image file itself.

MAKING REPORTIMAGES SERVICE REQUESTS USING SAS

STEP 1: CREATING A JOB

As described in the previous section, the client begins the process of using the *reportImages* service by first making a request to create a job to render a specific SAS Visual Analytics report as an SVG image. This request has the following components:

- Endpoint: https://www.example.com/reportImages/jobs
- Method: POST

The parameters of the API call are contained within the body of the request. For the purposes of leveraging the *reportImages* service as a speedometer, specific parameters and values must be included. The body should be composed of the following components:

- reportUri: /reports/reports/<report id>
- layoutType: entireSection

- selectionType: report
- refresh: true
- size: <Desired SVG Image Size>
- sectionIndex: <section number>

The *reportURI* parameter tells the service which SAS Visual Analytics report is to be rendered as an SVG. The *layoutType* parameter tells the service to create an SVG image of the entire report section that is specified in the *sectionIndex* parameter, which defines which report tab the service should render. The first tab in a report has an index value of zero, the second tab in the report has an index of 1, and so on. Similar to the *layoutType* parameter, the *selectionType* parameter tells the service to generate a single image, representing the entire report. The *size* parameter defines the desired width and height dimensions of the SVG image. Typically, these dimensions should be set so that they are similar to the size of the screen on which the SAS Visual Analytics Report is shown, when opened in the SAS® Report Viewer (for example, 1680x1050).

To retrieve the most accurate image generation data, it is important to set the *refresh* parameter to *true*. Ordinarily, when a request is made to generate an SVG of a specific report, the *reportImages* service first checks to see whether the requested image already exists within its cache. If it does, the service instantly returns the cached image, rather than rebuilding the image. However, this behavior is not desirable when using the *reportImages* service to gain insight into how long it takes the SAS Report Viewer to render a report in real time. Setting the *refresh* parameter to *true* forces the *reportImages* service to regenerate the SVG from scratch for each request. Therefore, the same report can be called several times and each resulting duration metric is much more accurate.

The following SAS code requests that a job be created to render a SAS Visual Analytics report:

```
* Base URI for the service call;
%let BASE_URI=%sysfunc(getoption(servicesbaseurl));

/* create filenames to hold responses*/
filename startjob temp;
filename resp_hdr temp;

/* Make request */
proc http
     method="POST"
      oauth_bearer=sas_services
     url="&BASE_URI/reportImages/jobs"
     ct="application/vnd.sas.report.images.job.request+json"
     in='{
               "reportUri" : "/reports/reports/<- report id ->",
               "layoutType" : "entireSection",
               "selectionType" : "report",
               "refresh":true,
               "size" : "<Desired Image Size>",
               "sectionIndex" : 0
          }'
    /* place response in filenames */
          out=startjob
          headerout=resp_hdr
          headerout_overwrite;
```

```
run;
```

Submitting the code above results in successfully sending a request and the *reportImages* service sending a response. This response is captured in the *filename startjob.* The JSON LIBNAME engine can be used to read the contents of *startjob filename* as a SAS library. After this library has been assigned, the response from the *reportImages* service can be viewed as SAS tables. A subset of the contents of the *startjob.root* table is shown in Figure 1.

Subset of table: startjob.root

id	state	creationTimeStamp
9200415b-7706-44b1-a440-7c0e12ecc080	running	2019-02-21T14:33:07.742Z

Figure 1. Subset of Table: startjob.root

Several pieces of important information are contained within *table startjob.root*. Most importantly, it contains the unique ID and current state of the job that has been initiated to create the SVG image. Because the job has just begun, the job's state is returned as *running*; or rather, it hasn't completed yet.

In order to move on to the next step of using the *reportImages* service, the ID of the job must be captured for use later in the SAS program. To achieve this, use the SYMPUTX function to save the value of the ID variable as the SAS macro variable *id*:

```
/* Use JSON LIBNAME engine to read in response */
libname startjob json;
data _NULL_;
    set startjob.root;
       call symputx('job_id',id);
run;
```

STEP 2: RETRIEVING JOB DURATION

After the job has successfully started and the job ID has been stored in the macro variable *id*, the developer can move on to the second step of using the *reportImages* service: retrieving how long it takes to render the SAS Visual Analytics report as an SVG image. This is achieved by making a second request to the *reportImages* service for the status of a job.

This API request has the following components:

- Endpoint: https://www.example.com/reportImages/jobs/<- job id ->
- Method: GET

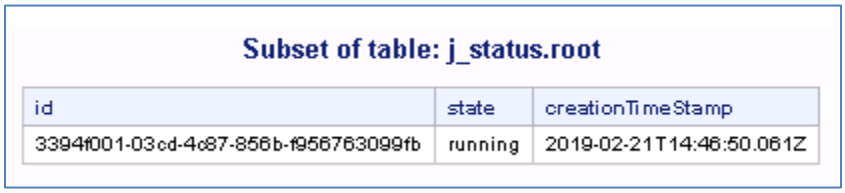
It is important to note the value at the end of the endpoint: *job id*. This must be the unique job ID that was stored in the macro variable *id* in Step 1. With a valid job ID value in the endpoint, the *reportImages* service successfully returns the status of the specified job. As in Step 1, the JSON LIBNAME engine is used to read the response in the form of a SAS table. The SAS code that requests the status of a specific job ID and reads in the response via the JSON LIBNAME engine is below:

```
* Base URI for the service call;
%let BASE_URI=%sysfunc(getoption(servicesbaseurl));
```

```
/* create filenames to hold responses*/
filename j_status temp;
filename res_hdr temp;
/*Make the request    */
    proc http
     method="GET"
        oauth_bearer=sas_services
     url="&BASE_URI/reportImages/jobs/&job_id"
     out=j_status
     headerout=res_hdr
     headerout_overwrite;
    run;

/* Use JSON LIBNAME engine to read in response */
    libname j_status json;
```

The current status of the job specified in the request is contained as the variable *state* in the *j_status.root* table. If at the time of the request the *reportImages* service has not completed generating the SVG, the response has a value of *running*, as shown in Figure 2.

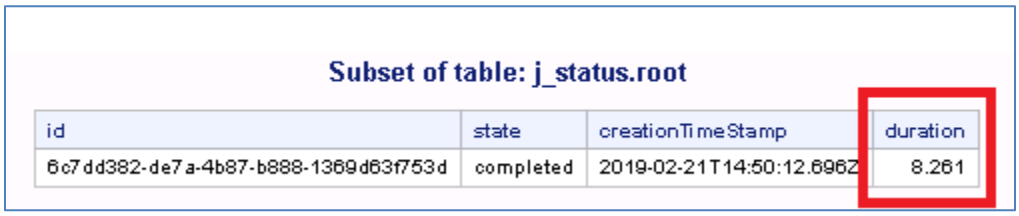


Subset of table: j_status.root

id	state	creationTimeStamp
3394f001-03cd-4c87-856b-f956763099fb	running	2019-02-21T14:46:50.061Z

Figure 2. Subset of Table: j_status.root

If the service has completed generating the SVG image, the response has a value of *completed* and there is a new variable named *duration* in the *status.root* data set. (See Figure 3.) The *duration* variable contains the total time it has taken to successfully render the report as an SVG image. This value is the key to using the *reportImages* service as a speedometer for report rendering time.



Subset of table: j_status.root

id	state	creationTimeStamp	duration
6c7dd382-de7a-4b87-b888-1369d63f753d	completed	2019-02-21T14:50:12.696Z	8.261

Figure 3. Subset of Table: j_status.root Including the Variable *duration*

Because it might take several seconds for the SVG to be completely generated, developers might have to repeat the request in Step 2 several times before successfully retrieving a status of *completed*. To avoid having to manually re-run the SAS code that performs the *get status* request, developers can invoke a SAS macro to automatically make repeated requests until the variable state returns a value of *completed*.

An example of such a macro is below:

```
* Base URI for the service call;
%let BASE_URI=%sysfunc(getoption(servicesbaseurl));

/* initially set status */
%let status=0;
%macro jobstatus;
```

```
    %do %until (&status ne 0);
           /* clear filenames and librefs */
                 filename res_hdr clear;
                 filename j_status clear;
                 libname j_status clear;
           /* assign filenames */
                 filename j_status temp;
                 filename res_hdr temp;
         /* Make request */
                 proc http
                  method="GET"
                   oauth_bearer=sas_services
                  url="&BASE_URI/reportImages/jobs/&job_id"
                  out=j_status
                  headerout=res_hdr
                  headerout_overwrite;
                 run;

           /* Read response */
                 libname j_status json;
           /* Determine state and reset status */
                 data job_status;
                       set j_status.root;
                             if state = 'running' then status = 0;
                             else if state = 'completed' then status = 1;
                             call symputx('status',status);
                 run;

    %end;
  %mend jobstatus;
  /*call the macro*/
  %jobstatus;
```

The code above begins by creating a macro variable named *status* and setting its value to zero. After this, the definition for the *jobstatus* macro begins with creating a DO UNTIL loop that runs until the value of the macro variable *status* is not zero. Because this loop is expected to run several times, the *libref* and *filename j_status* are cleared of values that might have been assigned in the loop's previous iteration.

After this setup, the request is made to get the status of the job that is specified in the *job_id* macro variable. Its response value is captured in the *j_status filename* and *libref*. A DATA step is then run to create a numeric variable named *status*, based on value of the *state* variable in the *j_status.root* table. If the value of *state* is *running*, the value of the *status* variable is created and set to zero. Alternatively, if the value of state is *completed*, the value is set to a quantity other than zero. The SYMPUTX function then reads the value of the *status* variable and uses it to reset the value of the *status* macro variable. Finally, the DO UNTIL loop is ended and macro definition is completed.

End to end, the macro *jobstatus* simplifies the process of retrieving the duration of a *reportImages* job by offering an automated way to make repeated requests until a status of *completed* is returned.

COMPARING MULTIPLE REPORT RENDER DURATIONS

The following example demonstrates how the *reportImages* service can be used as a speedometer for server-side report rendering time. This scenario consists of a SAS Visual

Analytics report that visualizes data containing information about the names of roads within the United States. Specifically, the report shows which states have roads that contain the word "Parkway" in their names. The data source of this report is a SAS data set named "us_roads" that contains 10 variables and 20 million observations. Figure 4 and Figure 5 show screenshots of the data source and report, respectively.

SAS Dataset: "us_roads"
10 Variables 20 Million Observations

DEGX	DEGY	SEGMENT	FULLNAME	MTFCC	ROADFLG	TLID	STATE	COUNTY	ID	STATECODE	STATENAME
-86.8346	34.1571	1	2nd Ave SW	S1200	Y	130244269	01	043	0001043	AL	Alabama
-86.2659	31.6699	1	Brantley Hwy	S1200	Y	10441314	01	041	0001041	AL	Alabama
-87.5234	34.6509	1	State Rte 157	S1200	Y	40780764	01	033	0001033	AL	Alabama
-87.4025	32.2562	1	Jefferson Davis Hwy	S1200	Y	1298420	01	047	0001047	AL	Alabama
-86.5791	31.1766	1	State Rte 137	S1200	Y	1856842	01	039	0001039	AL	Alabama
-85.8311	31.2916	1	Plaza Dr	S1200	Y	99554415	01	031	0001031	AL	Alabama
-86.8789	34.2116	1	I- 65	S1100	Y	130219522	01	043	0001043	AL	Alabama
-85.4693	33.7292	1	US Hwy 78	S1200	Y	612932371	01	029	0001029	AL	Alabama
-85.7858	31.4261	1	State Rte 27	S1200	Y	99596542	01	045	0001045	AL	Alabama
-87.0594	31.6871	1	State Rte 83	S1200	Y	2002051	01	035	0001035	AL	Alabama

Figure 4. Subset of the Source Table: us_roads

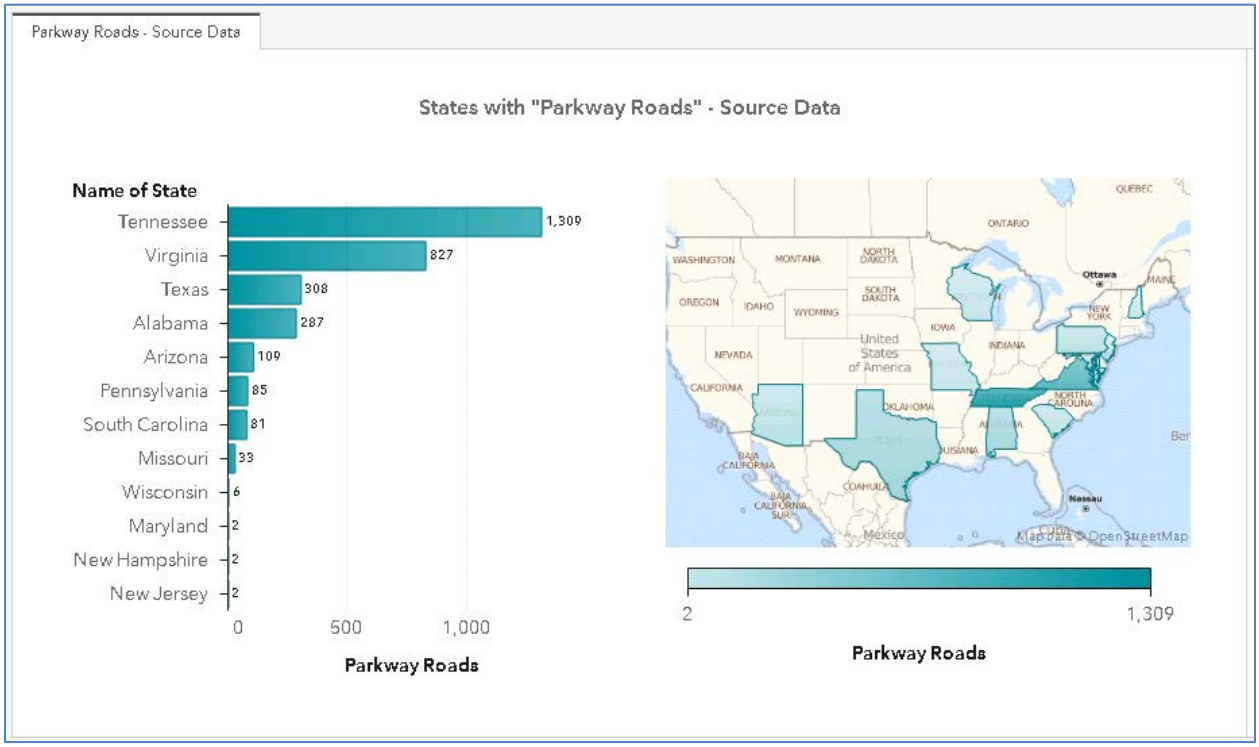


Figure 5. Visual Analytics Report Using the Source Table: us_roads

The data for this report is refreshed nightly. To ensure that a fast report load time occurs when the first user opens the report after the nightly data refresh, the developer is exploring various methods to maximize report speed.

One method is to create an aggregated version of the data source named *us_roads_agg* that contains only 3 variables and 12 observations. The report developer then creates a new version of the "Parkway Roads" report that uses *us_roads_agg* as its data source.

Screenshots of the *us_roads_agg* aggregated data source and its associated report appear in Figure 6 and Figure 7, respectively.

SAS Dataset: "us_roads_agg"
3 Variables 12 Observations

STATENAME	STATECODE	Parkway_Roads
Alabama	AL	287
Arizona	AZ	109
Maryland	MD	2
Missouri	MO	33
New Hampshire	NH	2
New Jersey	NJ	2
Pennsylvania	PA	85
South Carolina	SC	81
Tennessee	TN	1,309
Texas	TX	308
Virginia	VA	827
Wisconsin	WI	6

Figure 6. Aggregated Table: us_roads_agg

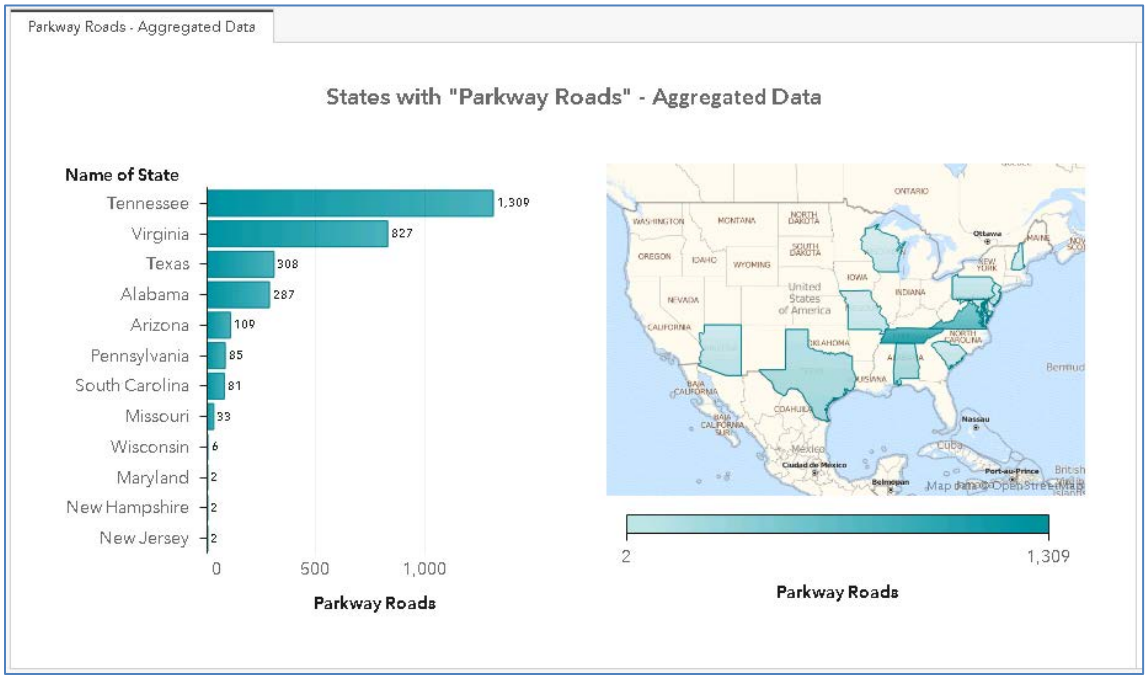


Figure 7. Visual Analytics Report Built Using the Source Table: us_roads_agg

Now that the report developer has created two different versions of the report, the *reportImages* service can be used to test the server-side rendering time for each of them.

Leveraging the previously discussed concepts of using the *reportImages* service to retrieve a job's duration, the report developer writes one comprehensive macro to test these two examples reports and return their respective server-side rendering times. The macro is

called by using the parameters *datasource*, *report_name*, and *report_URI*. This macro's definition is displayed below:

```
* Base URI for the service call;
%let BASE_URI=%sysfunc(getoption(servicesbaseurl));

%macro report_generation_duration(sourcedata,report_name,report_uri);

/* Refresh Data in CAS */
cas myses;
proc casutil;
    droptable incaslib="PUBLIC" casdata="&sourcedata" quiet;
    load incaslib="PUBLIC" outcaslib="PUBLIC"
casdata="&sourcedata..sashdat" casout="&sourcedata"  promote;
run;

/* create dynamic proc http 'in=' statement  */
data create_params;
    request_params = "'" || trim('{"reportUri" : "') || "&report_uri" ||
trim('","layoutType" : "entireSection","refresh":true,"selectionType" :
"report","size" : "1680x1050","version" : 1}' || "'");
    call symput('request_params',request_params);
run;

/* create job and get response */
filename resp_hdr clear;
filename startjob clear;
libname startjob clear;
filename startjob temp;
filename resp_hdr temp;
proc http
     method="POST"
     oauth_bearer=sas_services
     url="&BASE_URI/reportImages/jobs"
     ct="application/vnd.sas.report.images.job.request+json"
     in=&request_params.
     out=startjob
     headerout=resp_hdr
      headerout_overwrite;
run;
libname startjob json;

/* capture job id into macro variable job_id */
data _NULL_;
    set startjob.root;
      call symputx('job_id',id);
run;

/* Set initial &status to be zero */
%let status=0;

/* macro to check status until job is completed */
%macro jobstatus;
       %do %until (&status ne 0);
          filename res_hdr clear;
          filename j_status clear;
```

```
            libname j_status clear;
            filename j_status temp;
            filename res_hdr temp;
        /* Make API Call   */
            proc http
                   method="GET"
                   oauth_bearer=sas_services
                   url="&BASE_URI/reportImages/jobs/&job_id"
                   out=j_status
                   headerout=res_hdr
                   headerout_overwrite;
            run;

            libname j_status json;

            /* create &status macro variable */
            data job_status;
                  set j_status.root;
                        if state = 'running' then status = 0;
                        else if state = 'completed' then status = 1;
                        call symputx('status',status);
            run;

        %end;
    %mend jobstatus;

    /* call macro %jobstatus */
    %jobstatus;

    /* create and print final data set */
    data report;
        set j_status.root;
            reportName = "&report_name";
            reportURI = "&report_uri";
            label id = "reportImages Job ID"
            duration = "Job Duration"
            label state="Job Status";
    run;

    /* Print output */
    title "reportImages Duration - Report: '&report_name'";
    proc print data= report noobs label;
        var reportName reportURI id state duration;
    run;

    %mend report_generation_duration;
```

The macro begins by reloading the report data sources from sashdat files to simulate a nightly refresh of the data. This is achieved by using the PROC CASUTIL statement in conjunction with the *sourcedata* parameter. The next step is to dynamically generate the value for PROC HTTP's "in=" argument using the value contained the *report_uri* parameter. This value is placed in a macro variable named *request_params* via the SYMPUT function. The macro then uses the *request_params* value to make the needed requests to create a *reportImages* service job and repeatedly check its status until a value of *completed* is

returned. Finally, a data set that contains the final desired variables from the macro's results is created and printed.

This macro can now be called with the values from the two reports in this example scenario:

```
/* call reports */
%report_generation_duration(us_roads,Parkway Roads - Source
Data,/reports/reports/c199d225-a536-44c9-a9c3-5d9e19aac6cc);
%report_generation_duration(us_roads_agg,Parkway Roads - Aggregated
Data,/reports/reports/e3704e78-2316-48b3-9f25-a056a2bccc3f);
```

The result of submitting these two macro calls (and their associated parameters) is the following PROC PRINT outputs of the *report* data set for each of the two example reports shown in Figure 8.

reportImages Duration - Report: 'Parkway Roads - Source Data'

reportName	reportURI	reportImages Job ID	Job Status	Job Duration label
Parkway Roads - Source Data	/reports/reports/c199d225-a536-44c9-a9c3-5d9e19aac6cc	c3f799be-6e0a-4120-8af2-871a7fe5dbdc	completed	17.269

reportImages Duration - Report: 'Parkway Roads - Aggregated Data'

reportName	reportURI	reportImages Job ID	Job Status	Job Duration label
Parkway Roads - Aggregated Data	/reports/reports/e3704e78-2316-48b3-9f25-a056a2bccc3f	d23e52ac-e0ba-41c5-86d1-ccb2138a2291	completed	7.082

Figure 8. Printed Output from the Macro *report_generation_duration*

It is obvious from the results above that the report that uses the aggregated version of the data source renders much faster than the report that uses the larger source data.

CONCLUSION

Creating reports that render quickly is a key component in ensuring that report viewers are satisfied with their reporting systems. Though the true report rendering time for different users might be slowed by external factors, such as network performance, developers should ensure that their reports are as efficient as possible before the first customer opens the report using SAS Report Viewer. Using the *reportImages* service job duration time as a report optimization measuring stick greatly assists in this task.

As mentioned at the onset of this paper, there are several techniques that can maximize the speed of SAS Visual Analytics reporting: data partitioning, user-defined formats, and the use of aggregated data. Programmatically retrieving a report's server-side rendering time empowers the report developer to quickly explore all options for ensuing that a customer's data visualizations are returned in the shortest time possible. Armed with this new method of determining which report optimization is best for each specific report and data source, developers are ready to create their most optimized, fastest reports yet.

REFERENCES

Hemedinger, Chris. "How to secure your REST API credentials in SAS programs." Available https://blogs.sas.com/content/sasdummy/2018/01/16/hide-rest-api-tokens/. Accessed on September 12, 2018.

Henry, Joseph. 2016. "REST at Ease with SAS®: How to Use SAS to Get Your REST." *Proceedings of the SAS Global 2016 Conference*. Cary, NC: SAS Institute Inc. Available http://support.sas.com/resources/papers/proceedings16/SAS6363-2016.pdf.

Roda, Mike. 2018. "OpenID Connect Opens the Door to SAS® Viya® APIs." *Proceedings of the SAS Global 2018 Conference*. Cary, NC: SAS Institute Inc. Available https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/1737-2018.pdf.

SAS Institute Inc. 2018. "SAS Viya REST APIs." Available https://developer.sas.com/apis/rest/Topics/.

SAS Institute Inc. 2018. "Authentication and Access Tokens." Available https://developer.sas.com/apis/rest/Topics/#authentication-and-access-tokens.

SAS Institute Inc. 2018. "Report Images." Available https://developer.sas.com/apis/rest/Visualization/#report-images.

SAS Institute Inc. 2018. "SAS Maps Online." Available http://support.sas.com/rnd/datavisualization/mapsonline/html/usroads.html.

SAS Institute Inc. 2018. *SAS® Job Execution Web Application 2.1: User's Guide*. Available https://go.documentation.sas.com/api/docsets/jobexecug/2.0/content/jobexecug.pdf.

SAS Institute Inc. 2018. "PROC HTTP Statement." In *Base SAS 9.4 Procedures Guide*. 7th edition. Cary, NC: SAS Institute Inc. Available https://go.documentation.sas.com/?docsetId=proc&docsetVersion=9.4&docsetTarget=n197g47i7j66x9n15xi0gaha8ov6.htm&locale=en.

CONTACT INFORMATION

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

Michael Drutar
SAS Institute Inc.
Michael.Drutar@sas.com
http://www.sas.com