Paper SAS3472-2019

Enabling SAS® Software to Communicate RESTfully with Cloud Services

Tom Caswell and Fred Burke, SAS Institute Inc.

ABSTRACT

Representational State Transfer (REST) architecture has become the standard for cloud communication; it defines an abstraction among service providers that enables each to evolve independently from the others. Because REST architecture hides the underlying implementation, SAS® software does not need to know how any required function (such as authorization or persistence) is provided and it is not affected by changes in the cloud service. For example, authorization can change from LDAP to Kerberos and persistence can change from PostgreSQL to Oracle. SAS software can access functions by invoking the HTTP procedure to send cloud service requests and by using the LIBNAME statement with the JSON (JavaScript Object Notation) engine to parse cloud service responses. REST also requires stateless communication, which allows work to be automatically load-balanced to meet increasing demands. This paper shows how SAS software can become a RESTful client of cloud services.

INTRODUCTION

Since SAS was incorporated in 1976, its software has constantly evolved and today is a leader in the analytics market. Representational State Transfer (REST) was published in 2000 and would later become a cloud architecture standard. Although the SAS programming language is almost 25 years older than REST, it can still take advantage of the cloud architectural style to become a cloud service itself.

REPRESENTATIONAL STATE TRANSFER ARCHITECTURE

Fielding (2000) recognized the World Wide Web's explosive growth and what would be needed to continue that growth. One requirement is that network architectures must be scalable and independent so that they can evolve over time. REST is an architectural style that provides this ability. Fielding (2008) says, "REST is software design on the scale of decades: every detail is intended to promote software longevity and independent evolution."

Chapter 5 of Fielding (2000) covers REST in detail. It begins by defining a null style and building on it until all the following constraints have been constructed:

Client-server	This constraint provides separation of duties for a network architecture. The client is not responsible for data persistence, and the server is not responsible for rendering what the user will see. The separation of duties enables the client and the server to evolve independently of each other.
Stateless	This constraint requires the client to provide everything the server needs to perform the work. The server cannot maintain any context that is specific to a client. This constraint provides reliability and scalability for the network architecture. If a server fails, the request is simply routed to another server that can perform the work.

Cache	This constraint allows the client to keep responses from the server that are suitable for caching. This can improve performance.
Uniform interface	This constraint is especially critical to the REST architectural style. It requires the server not to be coupled to the work it provides. A typical example of this constraint pertains to persistence. A server needs to store information in its original state, but the persistence mechanism should not be exposed. A uniform interface allows one database vendor to be exchanged with another database vendor such that the client cannot see the change. Thus, the system can evolve without disruption. This constraint also identifies hypermedia as the engine of application state (HATEOAS).
Layered system	This constraint provides a single view of the system to clients. This constraint promotes performance and reliability through business logic encapsulation in a single interface and through network failover. Gateway APIs are a typical implementation.
Code-on-demand	This constraint is optional and allows the client to download code from the server and run it.

Hypermedia as the Engine of Application State (HATEOAS)

Hypermedia allows the server to provide functions for the client without coupling itself to the client. For example, the HTTP protocol uses links as hypermedia.Figure 1 shows a typical workflow that provides a data source, performs analytics on the data source, and publishes the results from the analytics.

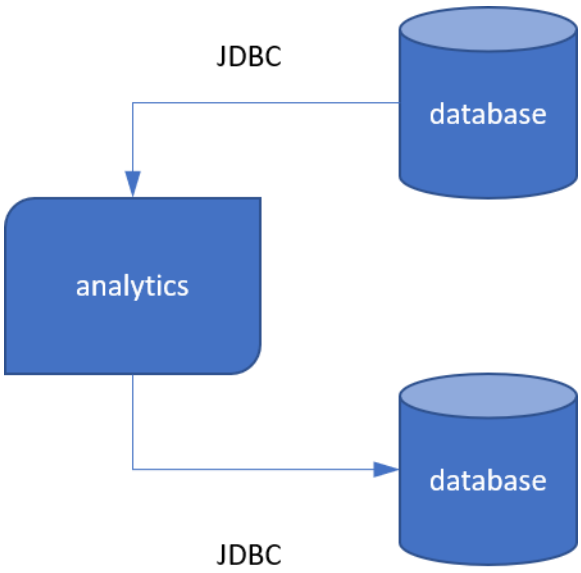

Figure 1 Analytics with Direct Connections

Each connection is a direct coupling between two components. A component in the system cannot be replaced without affecting the other components. For example, if a database vendor changes, a new Java Database Connectivity (JDBC) driver needs to be recompiled, tested, and deployed.

Figure 2 shows how HTTP methods and uniform resource identifiers (URIs) can help uncouple the components from each other.

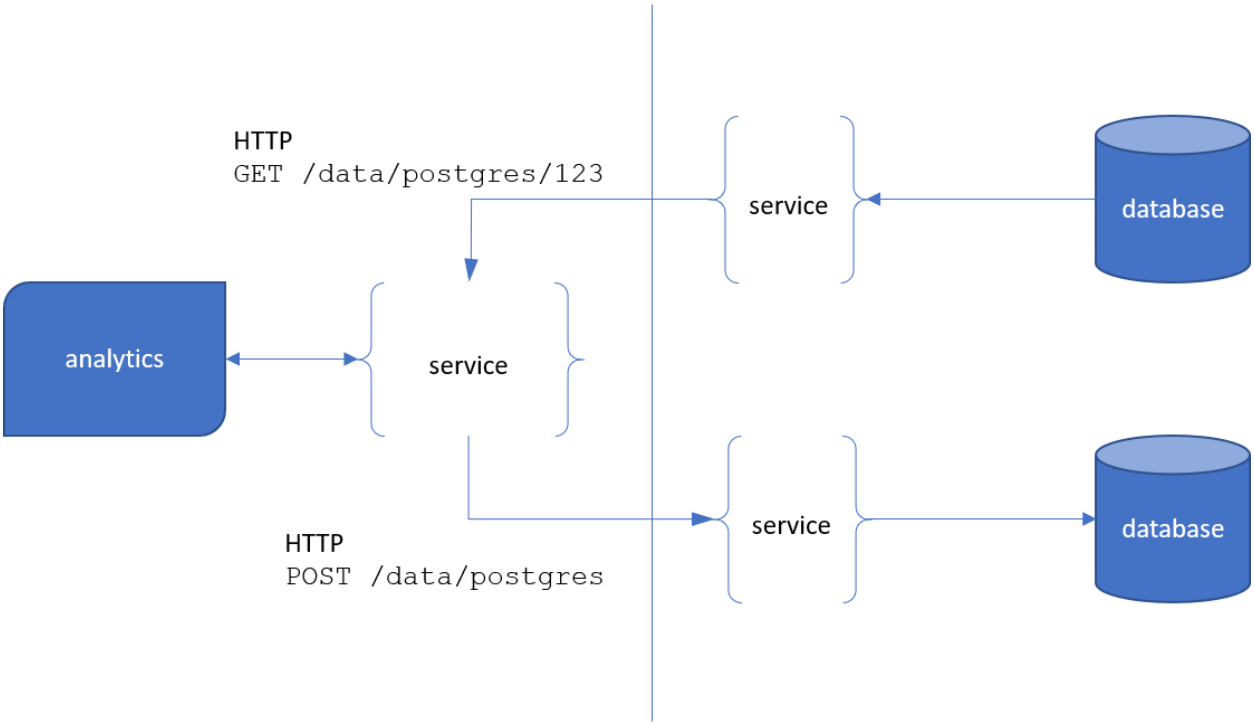

Figure 2 Analytics with HTTP Connections

The service implementation is hidden by the HTTP URI, such as `/data/postgres`, which is considered an identity. A service can be replaced, but the replacement must have its own URI. For example, if the results database vendor changes and its identity is now `/data/mysql`, each instance of `/data/postgres` will need to be changed to `/data/mysql`.

Figure 3 shows how HATEOAS allows the components to be replaced without affecting the other components.

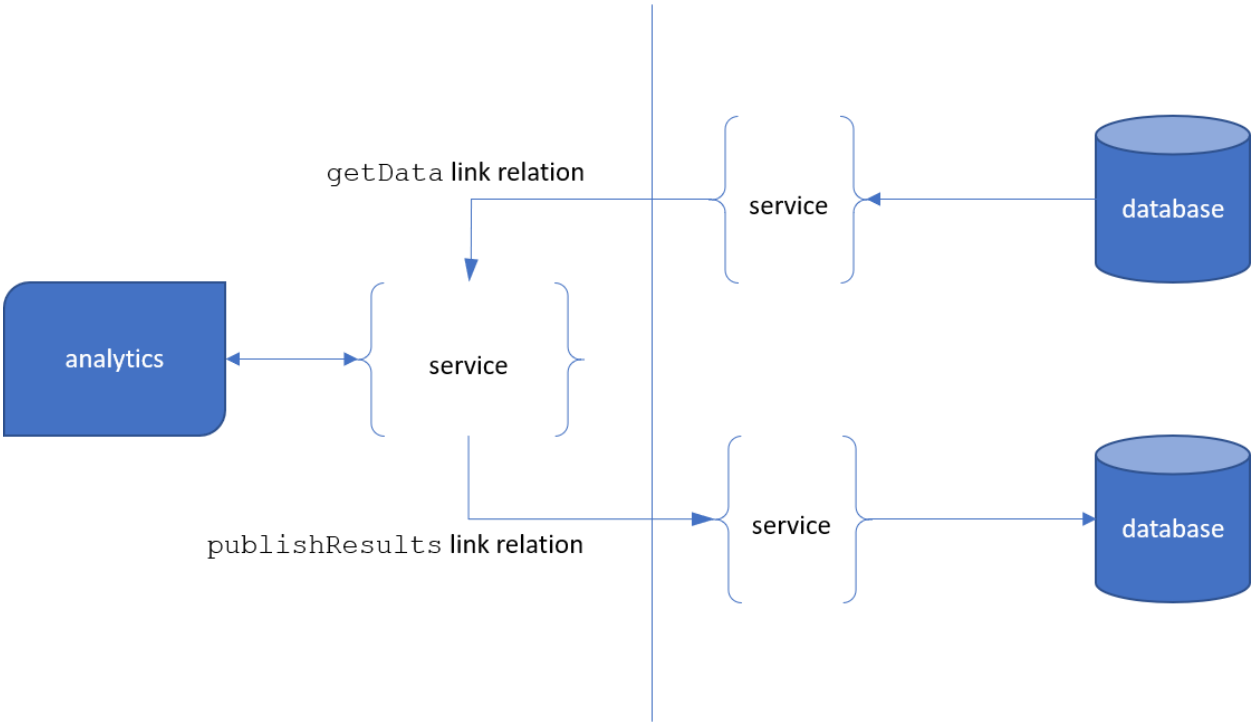

Figure 3 Analytics with REST Connections

The link relations identify the task that is to be performed without exposing the implementation or the identity. The URI can change, but the link relation stays the same. For example, suppose that the results database vendor changes and its identity is now `/data/mysql`. Because the client knows only the `publishResults` link relation, it never knows the difference when the service changes the link URI, and the analytics continue uninterrupted.

It is also important to note the stateless constraint for this system. Each request and response must be implemented in a way such that no state is maintained by the server. After the request is serviced and the response returned, the server is done with its work and is idle. The stateless constraint allows automatic load management and reliability. If another server needs to perform the work, the request is simply rerouted and the work continues. Typical examples of state would be context that persists in the session or the server waiting for another request to continue working.

Fielding (2008) says, "A REST API should spend almost all of its descriptive effort in defining the media type(s) … and/or hypertext-enabled mark-up for existing standard media types." The design effort spent here will allow the system to independently evolve over time without interruption.

SAS AS A RESTFUL ARCHITECTURE

How can HATEOAS (hypermedia as the engine of application state) be used within your SAS code? A good example that shows this approach can be obtained from Model Studio in SAS® Visual Mining and Machine Learning version 8.4. You can use Model Studio to create a simple pipeline that connects a Decision Tree node to the Data node. After you successfully run the pipeline, you can download the score API code by going to the **Pipeline**

Comparison tab, clicking the ⋮ icon, and selecting the **Download score API** link. An example of the downloaded code is shown in full in Appendix A. This section describes aspects of that code in relationship to HATEOAS.

The SAS code in Appendix A demonstrates how SAS can make use of HATEOAS. It consists of three macros (the main macro, `%DMScoreModel`, and two supporting macros, `%determineJobStatus` and `%echofile`) as well as some set-up code and the call to the main macro. The purpose of this code is to run the score code that is generated by the Decision Tree model against a specified set of data and produce an output data set. The downloaded score API code can be run in SAS® Studio. Any modeling node in a Model Studio pipeline will generate similar score API code that can be executed this way.

The HTTP procedure call made in the main macro sends a request to the model's scoreExecutions endpoint to run the score code. The information available in the response body is used to determine when the execution of the scoring process is complete. This endpoint, along with other SAS RESTful API endpoints, includes a 'links' section in the body of the response that is returned by a successfully completed HTTP request. It is the information in the 'links' section that enables the use of HATEOAS. For each link in the 'links' section, three pieces of information are needed: the relationship name, the HREF, and the Accept header type. Use this information to first find the relationship of interest and then to collect the URI and the Accept header type to specify in the next HTTP procedure call.

In the following DATA step, the `data _null_;` code block generates a map file that tells the JSON engine (in the LIBNAME statement) how to parse the 'links' section in the JSON file that the FILEREF=option specifies. You can use this map code to extract the content of the 'links' section that almost any SAS RESTful API endpoint returns.

For the `%determineJobStatus` macro, it is the 'self' relationship information that is needed. The second `data _null_;` code block extracts the HREF link and the Accept header type of the 'self' relationship and assigns them to macro variables so that they can later be used by this macro.

```
filename mapfile TEMP;
data _null_;
   file mapfile;
   put '{';
   put '  "DATASETS": [';
   put '    {';
   put '      "DSNAME": "links",';
   put '      "TABLEPATH": "/root/links",';
   put '      "VARIABLES": [';
   put '        {';
   put '          "NAME": "rel",';
   put '          "TYPE": "CHARACTER",';
   put '          "PATH": "/root/links/rel",';
   put '          "CURRENT_LENGTH": 63';
   put '        },';
   put '        {';
   put '          "NAME": "href",';
   put '          "TYPE": "CHARACTER",';
   put '          "PATH": "/root/links/href",';
   put '          "CURRENT_LENGTH": 2047';
   put '        },';
   put '        {';
   put '          "NAME": "type",';
   put '          "TYPE": "CHARACTER",';
```

```
   put '           "PATH": "/root/links/type",';
   put '           "CURRENT_LENGTH": 255';
   put '         }';
   put '       ]';
   put '     }';
   put '  ]';
   put '}';
run;


libname jsonlib JSON fileref=resp map=mapfile;

%let selfHref =;
%let selfType =;

data _null_;
   set jsonlib.links;
      where rel eq 'self';
   call symput("selfHref", href);
   call symput("selfType", type);
run;
filename mapfile;
libname jsonlib;
```

By obtaining the URI and associated media type in this manner, the SAS code has the information it needs to make a RESTful call to any link available in the 'links' section. The documentation for each of the public SAS RESTful APIs describes the available links (SAS Institute Inc. 2019).

The SAS code in the `%determineJobStatus` macro shows how to configure the HTTP procedure request to use this information so that the RESTful call, as specified by the relationship name, can be properly submitted. This is what HATEOAS is all about, making an HTTP request and then using the returned information to make additional HTTP requests to accomplish the desired task. In this example, that task is to determine when the execution of the score code completes and whether it completed successfully.

It is important to note that when the Accept header is obtained, it will not include the return type format specification. The desired format needs to be appended to the Accept type header before the HTTP procedure request is submitted. Most SAS RESTful APIs support at least the JSON and XML formats. The following code indicates that the response body is expected to be in JSON format:

```
"Accept"="&_mediaType.+json"
```

If XML is the desired format, then simply append `+xml` instead of `+json`.

The following lines of code, from the main macro, show that a call to the `%determineJobStatus` macro is made only if the scoreExecutionState is either "running" or "pending" when it is first checked. If the `%determineJobStatus` macro is called, then a request is made every second to see whether the state has changed from either of these two states (or the timeout limit has been reached), at which point the macro is exited. Once back in the main macro, a determination is made as to the current state of the scoring job, as follows:

```
/*
* If the job hasn't completed yet, then start polling the job until it
* is done or it times out.
*/
%if "&scoreExecutionState" in ("running" "pending") %then %do;
   %determineJobStatus(&selfHref, &selfType, scoreExecutionState);
   %if %sysevalf(&syscc > 4) %then %do;
      %put ERROR: An error was encountered while determining the job
           status. ErrCode: &syscc;
      %goto exitM;
   %end;
%end;

%if "&scoreExecutionState" eq "completed" %then %goto exitM;
%else
%if "&scoreExecutionState" in ("running" "pending") %then %do;
   %put WARNING: The job did not complete yet. Its state is:
        &scoreExecutionState;
   %goto exitM;
%end;
%else %if "&scoreExecutionState" in ("canceled" "timedOut") %then %do;
   %put WARNING: The job%str(%')s state is: &scoreExecutionState;
   %goto exitM;
%end;
%else %do;
   %put ERROR: The job failed. View the job%str(%')s log via the Jobs
        icon in SAS Environment Manager. The job%str(%')s name start
        out as 'The Scoring operation for Data Mining model ...';
   %goto exitM;
%end;
```

When the scoreExecutionState is set to "completed", then the data set that was generated by the score code and specified by the `outputTableName` macro variable will exist in the caslib that is specified by the `outputCasLib` macro variable.

Running in SAS® Studio

As mentioned previously, the downloaded score code can be run in the web-based SAS Studio application. When you are running Model Studio, you can start SAS Studio and run the downloaded score code by selecting **Develop SAS Code** from the list of actions available from the  icon and pasting the code into the editor pane of SAS Studio. Before you submit the code, modify the following two lines to specify the caslib to write to and the data set name to assign to the new file.

```
%let outputCasLib = OUTPUT_CAS_LIB_NAME;
%let outputTableName = OUTPUT_CAS_TABLE_NAME;
```

The SAS Studio environment provides you two important benefits: The first is that since you already authenticated yourself to the web application, you do not need to generate a separate token to pass to the compute server. Instead the HTTP request just needs to specify the following line for the compute service to be able to execute the code on your behalf:

```
OAUTH_BEARER=SAS_SERVICES
```

The second benefit is that you do not need specify the host and port information. Instead, the following code extracts that information from the run-time environment and assigns it to a macro variable, which is then used in the generation the URI:

```
%let servicesBaseUrl =;
data _null_;
   length string $ 1024;
   string= getoption('SERVICESBASEURL');
   call symput('servicesBaseUrl', trim(string));
run;
```

These two benefits enable you to submit the modified score API code on the server where the Model Studio project was originally created either immediately or at any later time. When submitted, the code will correctly execute provided that the referenced model and scoring data set both still exist on that server.

ANALYTICS USE CASE AND TOPOLOGY

Model scoring is an example of analytics you might want to perform. Figure 4 shows how a model is defined, trained, and published. Then you can use the published code to make business decisions.

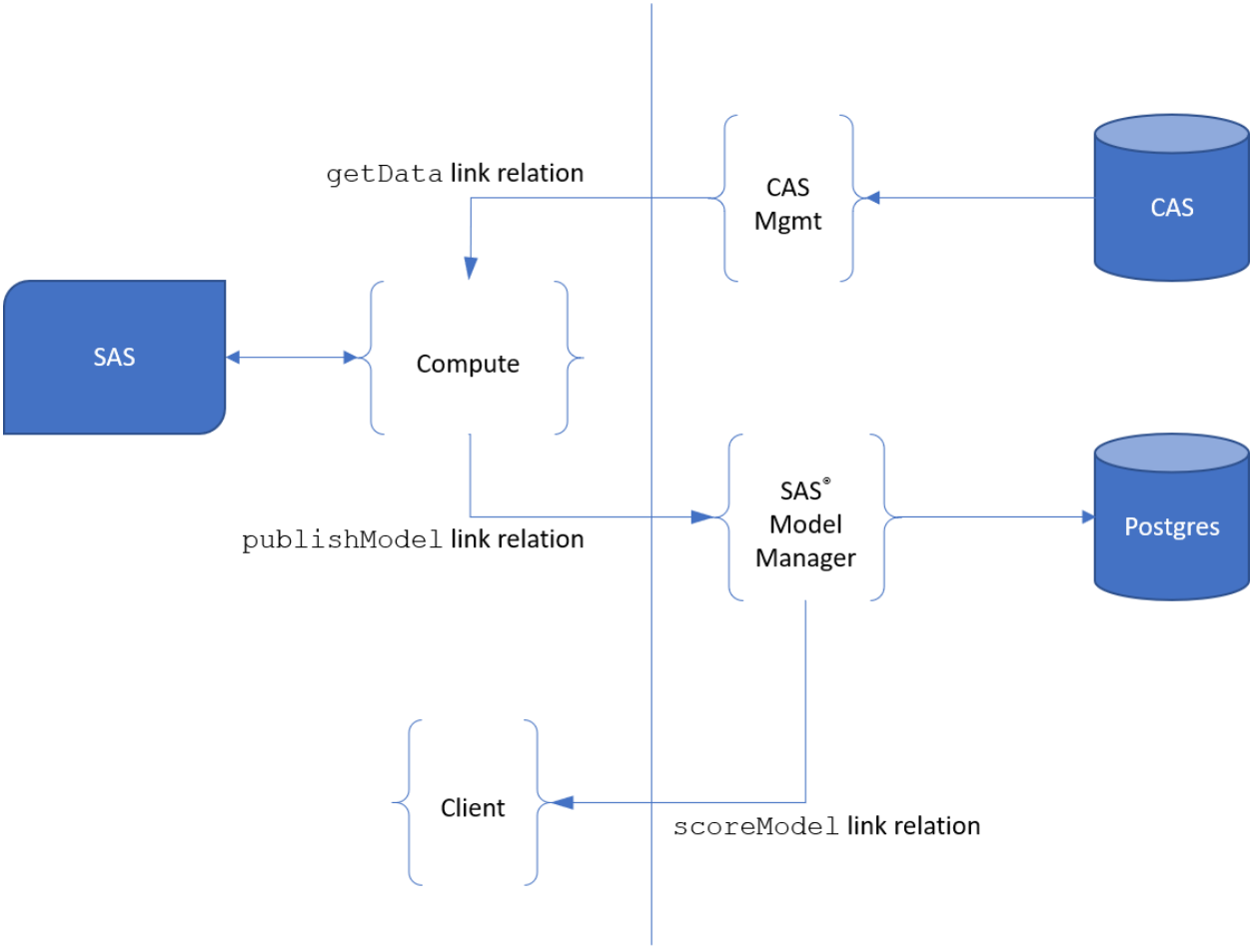

Figure 4 Model Scoring

Each of the services in Figure 4 performs a function and is completely isolated from the other services. If you replace SAS® Model Manager with another service that implements the `publishModel` and `scoreModel` link contracts, the Compute and Client services won't see any changes.

CONCLUSION

SAS software can behave as a cloud service and communicate with other cloud services RESTfully by taking advantage of the REST architectural style through the use of PROC HTTP and the use of JSON in the LIBNAME statement. SAS software can run as a cloud service to operate without database awareness, enforce security without credentials management, and recognize a multi-tenancy environment without any additional configuration. SAS code can move to the cloud and work undisturbed, even as the system around it evolves.

APPENDIX A

The following score API code has been downloaded from a Decision Tree node. The formatting has been slightly altered for presentation in this paper.

```
/*****************************************************************************
** Macro: Score Data Mining Model
**
** Description: Score the specified model by using the score execution
**              service.
*****************************************************************************/

/*****************************************************************************
* Edit the following options to customize the scoring operation:
* outputCasLib: (REQ)The name of the caslib where the score output table
*               is to be written.
* outputTableName: (REQ) The name of the output scoring table to create.
*****************************************************************************/

   %let outputCasLib = OUTPUT_CAS_LIB_NAME;
   %let outputTableName = OUTPUT_CAS_TABLE_NAME;


   /***********************************
   * The main macro DMScoreModel
   ***********************************/
   %macro DMScoreModel(projectId, modelId, datasourceUri, outputCasLib,
                       outputTableName)/minoperator;
      filename resp TEMP;
      filename headers TEMP;
      filename data TEMP;

      %let u=dataMining/projects/&projectId/models/&modelId/scoreExecutions;

      %let scoreModelUrl =&servicesBaseUrl.&u;

      %let syscc =0;
      proc json out=data pretty;
         write open object;
         write values "dataTableUri" "&datasourceUri";
         write values "outputCasLibName" "&outputCasLib";
         write values "outputTableName" "&outputTableName";
         write close;
```

```
      run;
      %if %sysevalf(&syscc > 4) %then %do;
         %put ERROR: PROC JSON set an error status code of: &syscc;
         %goto exitM;
      %end;

      %global SYS_PROCHTTP_STATUS_CODE SYS_PROCHTTP_STATUS_PHRASE;
      %let SYS_PROCHTTP_STATUS_CODE=;
      %let SYS_PROCHTTP_STATUS_PHRASE=;

      %let syscc =0;
      proc http
         method="POST"
         OAUTH_BEARER=SAS_SERVICES
         url="&scoreModelUrl"
         in=data
         headerout=headers
         out=resp;
         headers
         "Accept"="application/vnd.sas.score.execution+json"
         "Content-Type"=
"application/vnd.sas.analytics.data.mining.model.score.request+json";
      run;
      %if %sysevalf(&syscc > 4) %then %do;
         %put ERROR: PROC HTTP set an error status code of: &syscc;
         %goto exitM;
      %end;

      %if ^(&SYS_PROCHTTP_STATUS_CODE in (200 201 202)) %then %do;
         %put ERROR: PROC HTTP returned an HTTP status code of:
              &SYS_PROCHTTP_STATUS_CODE - &SYS_PROCHTTP_STATUS_PHRASE;
         %echoFile(resp);
         %goto exitM;
      %end;

      filename mapfile TEMP;
      data _null_;
         file mapfile;
         put '{';
         put '  "DATASETS": [';
         put '    {';
         put '      "DSNAME": "links",';
         put '      "TABLEPATH": "/root/links",';
         put '      "VARIABLES": [';
         put '        {';
         put '          "NAME": "rel",';
         put '          "TYPE": "CHARACTER",';
         put '          "PATH": "/root/links/rel",';
         put '          "CURRENT_LENGTH": 63';
         put '        },';
         put '        {';
         put '          "NAME": "href",';
         put '          "TYPE": "CHARACTER",';
         put '          "PATH": "/root/links/href",';
         put '          "CURRENT_LENGTH": 2047';
         put '        },';
         put '        {';
```

```
   put '           "NAME": "type",';
   put '           "TYPE": "CHARACTER",';
   put '           "PATH": "/root/links/type",';
   put '           "CURRENT_LENGTH": 255';
   put '         }';
   put '       ]';
   put '    }';
   put '  ]';
   put '}';
run;


libname jsonlib JSON fileref=resp map=mapfile;

%let selfHref =;
%let selfType =;

data _null_;
   set jsonlib.links;
      where rel eq 'self';
   call symput("selfHref", href);
   call symput("selfType", type);
run;
filename mapfile;
libname jsonlib;


libname jsonlib JSON fileref=resp;

%let scoreExecutionState=;

data _null_;
   set jsonlib.root;
   call symput("scoreExecutionState", state);
run;
filename resp;
libname jsonlib;

/*
* If the job hasn't completed yet, then start polling the job until it
* is done or it times out.
*/
%if "&scoreExecutionState" in ("running" "pending") %then %do;
   %determineJobStatus(&selfHref, &selfType, scoreExecutionState);
   %if %sysevalf(&syscc > 4) %then %do;
      %put ERROR: An error was encountered while determining the job
           status. ErrCode: &syscc;
      %goto exitM;
   %end;
%end;

%if "&scoreExecutionState" eq "completed" %then %goto exitM;
%else
%if "&scoreExecutionState" in ("running" "pending") %then %do;
   %put WARNING: The job did not complete yet. Its state is:
        &scoreExecutionState;
   %goto exitM;
```

```
   %end;
   %else %if "&scoreExecutionState" in ("canceled" "timedOut") %then %do;
      %put WARNING: The job%str(%')s state is: &scoreExecutionState;
      %goto exitM;
   %end;
   %else %do;
      %put ERROR: The job failed. View the job%str(%')s log via the Jobs
           icon in SAS Environment Manager. The job%str(%')s name starts
           with 'The Scoring operation for Data Mining model ...';
      %goto exitM;
   %end;

   %exitM:
%mend;


/************************************
* Output the file to the log.
************************************/
%macro echoFile(_fileRef);
   data _null_;
      infile &_fileRef;
      input;
      put _infile_;
   run;
%mend;


/************************************
* Determine when the job is finished running.
************************************/
%macro determineJobStatus(_jobUri, _mediaType, _rtnVal)/minoperator;
   %let &_rtnVal=running;

   /* Poll for a maximum of ~100 seconds */
   %let maxPollingAttempts =100;
   %let seconds            =1;
   %let loopLimit          =20;
   %let i                  =0;

   %let operation =JobStatus;

   %let url =&servicesBaseUrl&_jobUri;


   filename resp TEMP;
   %do %while (&i < &maxPollingAttempts);
      %do j=1 %to &loopLimit;
         %let i = %eval(&i + &seconds);
         %if &i >= &maxPollingAttempts %then %goto exitM;
         data _null_;
            call sleep(&seconds, 1);
         run;

         proc http
            out=resp
            method="GET"
```

```
            OAUTH_BEARER=SAS_SERVICES
            url="&url"
            ;
            headers
               "Accept"="&_mediaType.+json"
            ;
         run;
         %if %sysevalf(&syscc > 4) %then %do;
            %put ERROR: PROC HTTP set an error status code of: &syscc;
            %goto exitM;
         %end;

         %if &SYS_PROCHTTP_STATUS_CODE eq 200 %then %do;
            libname job JSON fileref=resp;
            data _null_;
               set job.root;
               call symput("&_rtnVal", trim(state));
            run;
            libname job;
            %if ^("&&&_rtnVal" in ("running" "pending")) %then
               %goto exitM;
         %end;
         %else %do;
            %let syscc=&SYS_PROCHTTP_STATUS_CODE;
            %put ERROR: An unexpected HTTP STATUS code of
                 &SYS_PROCHTTP_STATUS_CODE was returned. Exiting.;
            %goto exitM;
         %end;
      %end;
      %let seconds = %eval (&seconds + 1);
   %end;

   %exitM:
   filename resp;
%mend;


/************************************
* Obtain the servicesBaseUrl
************************************/
%let servicesBaseUrl =;
data _null_;
   length string $ 1024;
   string= getoption('SERVICESBASEURL');
   call symput('servicesBaseUrl', trim(string));
run;

%let projectId     = a3b2ff6f-0e10-40c6-b19a-5a8a0f505139
%let modelId       = 9e4a3d21-2841-4bee-95a3-2895022b793e;
%let datasourceUri = /dataTables/dataSources/cas~fs~cas-shared-
default~fs~Public/tables/HMEQ;

%DMScoreModel(&projectId, &modelId, &datasourceUri, &outputCasLib,
              &outputTableName);
```

REFERENCES

Fielding, R. T. 2000. "*Architectural Styles and the Design of Network-Based Software Architectures.*" PhD diss., University of California, Irvine. Available: https://www.ics.uci.edu/~fielding/pubs/dissertation/top.htm

Fielding, R. T. "*REST APIs Must Be Hypertext-Driven*." Available: https://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven Last modified October 20, 2008. Accessed on February 20, 2019.

SAS Institute Inc. 2019. REST APIs for SAS Viya and CAS. Accessed March 15, 2019. Available https://developer.sas.com/guides/rest.html.

CONTACT INFORMATION

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

Tom Caswell
SAS
tom.caswell@sas.com

Fred Burke
SAS
fred.burke@sas.com