SAS® GLOBAL FORUM 2020 #SASGF

Paper SAS4278-2020

Ten Minutes to Your First Hello World: REST APIs

Andy Bouts, SAS Institute Inc.

ABSTRACT

Like many other software applications, your time to first Hello World (TtFHW) with the SAS® Viya® REST APIs should be seamless and simple. The goal of this paper is to demonstrate with a standard workflow that the SAS Viya playbook works out of the box. Then, once you have used the playbook, you'll be enabled to envision how you can extend it for your use cases. The playbook uses standard Python modules to connect to the SAS Viya REST APIs and is designed with easy to read variable declarations and helper functions for ease of use. The GitHub project includes all dependencies and is designed to work with SAS Viya 3.4 or 3.5.

INTRODUCTION

Chances are that when you learned new technologies over the past few decades, it was suggested that you begin by developing your first "Hello World" as an example to help you learn. This practice has become quite common.

This paper follows a similar practice. The objective is to facilitate your time to first Hello World (TtFHW) with the SAS Viya REST APIs so that the process is seamless and simple. Accompanying this paper is a Jupyter based playbook that works out of the box with SAS Viya, uses Jupyter for the interface, and follows a standard workflow. The playbook is used to demonstrate a generally accepted practice in order to prepare an Analytical Base Table (ABT). Analysts typically denormalize the data and create statistically relevant columns when creating an ABT. The resulting ABT can then be used for subsequent analysis and modeling in SAS Viya.

This TtFHW playbook is designed to work with SAS Viya 3.4 or 3.5, and it requires minimal Python packages.

OBJECTIVE:

By executing the following playbook, you will create a standard ABT and promote the ABT so that it can be used in other SAS Viya applications.

- Process and prepare the WATER_CLUSTER data set from the Samples CAS library.
    - The WATER_CLUSTER data set contains data on home water consumption and is used in example Visual Analytics reports
- Run a SAS DATA step to add an indicator variable by calling the RunCode action.
- Aggregate the data set with the equivalent of PROC MEANS, and then promote the ABT for use by other SAS Viya applications.

DISCLAIMER:

The suggested steps in this playbook are not exhaustive around creating an ABT; your individual requirements may require more or less work.

PREREQUISITES:

- SAS Viya 3.4 or 3.5
- Python 3.x, note that all developed was performed with 3.7.4
- Jupyter {Hub | Lab | Notebook} is recommended but not required
- Python requests package, the requests module allows us to make HTTP calls. Some have said it has similarities to PROC HTTP.
- Python JSON package, the JSON module allows us to work with JSON objects and events.
- Python pandas package, the pandas module allows us to work with data objects in various formats, like DataFrames.
- Python getpass package, the getpass module allows for a password prompt with variable assignment for the session.
- Git, in order to clone the project: https://github.com/sascommunities/sas-global-forum-2020/tree/master/papers/4278-2020-Bouts

SUPPORTING DOCUMENTATION:

- Information supporting CAS REST APIs can be found at developer.sas.com.
- Information supporting the CAS Action Sets behind the CAS REST APIs can be found in the following two locations:
  - SAS® Viya® 3.5: System Programming Guide
  - SAS® Viya® 3.5 Actions and Action Sets by Name and Product

ESSENTIAL PREPARATION:

There are a few options for connecting to the CAS REST APIs. This playbook leverages OAuth tokens for authentication. Client secrets are required to use OAuth tokens.

- Reference this blog for instructions on how to configure the client secrets.
- Work with your SAS Environment Administrator to set up the appropriate client secrets.

SECTION 1: PREPARATION

1.1: SETUP

In this step, in order to set up for this playbook, we'll need to perform python imports in order to load the desired python packages from the prerequisites for use.

Then, we will assign some common variables that will be used throughout the rest of the playbook.

The first change that you will need to make in this section is to define the `Base_url` in the variable assignment:

```
base_url = 'https://<URL>'
```

In this example, the URL is probably in a form similar to this:

```
<host.subDomain.topLevelDomain>
```

For this playbook to connect to your SAS Viya environment, it is essential for you to work with your SAS administrator and ask them to provide a ClientID and a ClientSecret. Perhaps they need to create these if they have not already been created. Please reference the Essential Preparation section of this paper for more information.

The next change that you need to make is to define the `oAuthCliendId` and the `oAuthClientSecret` in the variable assignment:

```
oAuthCliendId = 'your-oAuthCliendId'

oAuthClientSecret = 'your-oAuthClientSecret'
```

1.2: AUTHENTICATE

In this step, we leverage the work completed in Step 1.1 to authenticate to your SAS Viya environment and retrieve an OAuth token as well as a session from the SAS Cloud Analytics Server (CAS).

To authenticate, use some standard Python functionality to capture the credentials that you use with your SAS Viya environment. Please note the reassignment statement near the middle of this step, which immediately reassigns the password (pw) variable after use:

```
# immediately reassign the pw variable to null since you are done with it
pw = None
```

If you have alternative requirements for securing your user credentials, please follow those guidelines to maintain the trust established by your organization's security posture.

If this step executes successfully in Jupyter, you are presented with an OAuth token as well as a CAS session ID in the cell response area. These values can be suppressed instead of printed to the screen by updating which print statement rows are (un)commented.

There are many aspects of OAuth token security parameters. An example parameter is token duration or expiration. These types of security parameters are established by the configuration that the SAS administrator used when they created the `oAuthCliendId` and the `oAuthClientSecret`.

1.3: VARIABLE ASSIGNMENT

In this step, we assign some variables that are used when calling the REST API endpoints.

For this playbook, we will read the WATER_CLUSTER SASHDAT file that's typically preconfigured in the Samples CAS library on SAS Viya. For those of you who are familiar with the standard Sashelp library, Samples can be thought of as similar in nature to Sashelp.

Then, we create a temporary session-defined version of WATER_CLUSTER so that we can work with the data in CAS. For our temporary file, we work with it in the standard Casuser library that typically comes preconfigured on SAS Viya.

As with prior versions of SAS, the library names are not case sensitive. These variables are:

```
# Establish data and library variables
sourceCasLib = 'samples'
sourceDataPathTable = 'WATER_CLUSTER.sashdat'
destinationCasLib = 'casuser'
destinationDataPathTable = 'WATER_CLUSTER_<initials>'
destinationDataAbt = 'WATER_CLUSTER_<initials>_ABT'
```

Note: Replace `<initials>` with your own unique value.

1.4: HELPER FUNCTIONS

In this step, we declare several helper functions in Python. The helper functions facilitate code reuse and efficiency.

`def printLoadVrbls()` prints the variables that are defined from Step 1.3 and can be used each time a REST API endpoint is called to verify the variables that are used.

`def callEndpoint()` calls the REST API endpoint according to the headers and payload. Note: A global variable `responseObj` is defined in this function.

`def printPayload()` prints the payload to help the user verify the actual payload that was passed to the REST API endpoint.

`def printResponse()`parses the response object and indicate success or failure to the user based on various attributes of the response object.

`def printResponseLog()` parses the log element of the response object and returns the log to the user for review.

`def printResponseObj()` prints the entire return object in JSON so that the user can review the full results of the REST API endpoint call.

`def printTable():` parses an element of the REST API endpoint response object and casts that field into a pandas DataFrame for ease of review. The `rows` field is parsed and is defined by this hierarchy:

```
responseObj.get('results').get('ColumnInfo').get('rows')
```

At this point, your setup should be defined.

SECTION 2: WORKING WITH DATA IN CAS

2.1: LOAD A DATA SET INTO CAS

Now that our preparation is completed, we can work with CAS by using data that's typically preconfigured in SAS Viya. In order to work with the data, we load the WATER_CLUSTER data set that's in the Samples CAS library on SAS Viya. Like I mentioned, the WATER_CLUSTER data set should be available as a SASHDAT file on every deployment. When this step is complete, the WATER_CLUSTER_<initials> data set should be loaded in your Casuser library.

Note, if you wish to load other data sets, just you'll just update the variables in Step 1.3.

We load the data into CAS using the `table.loadTable` action:

1. Call `printLoadVrbls()` to verify the variable assignments.
2. Define both a URL and a payload.
3. Call `callEndpoint()`, which calls the REST API endpoint and sets `responseObj`.
4. Call `printResponse()`, which obtains and parses the response object.
5. Call `printPayload()` to verify the variable assignments.
6. Optionally, you can uncomment (remove the '#' from) `# printResponseLog()`to display the log element of the API response object.
7. Optionally, you can uncomment `# printResponseObj()`to display the entire API response object.

If everything works as expected, you should see a response like the following:

```
-The API response "status" is "0", indicating the action was successful.
```

If you observe any other response, then either the data set was already loaded in CAS or there was some other error. To further analyze, you might desire to uncomment the optional print functions for debugging.

Step 2.1 was based on this [SAS documentation]

2.2: OPTIONAL – VERIFY THAT THE DATA SET WAS LOADED INTO CAS

Presuming that your WATER_CLUSTER_<initials> data set was loaded into your Casuser library, you might desire to verify that the data set is ready for you analytical preprocessing.

We verify that the data is available for use in CAS using the `table.tableInfo` action:

1. Call `printLoadVrbls()` to verify the variable assignments.
2. Define both a URL and a payload.
3. Call `callEndpoint()`, which calls the REST API endpoint and sets `responseObj`.
4. Call `printResponse()`, which obtains and parses the response object.
5. Call `printPayload()` to verify the variable assignments.
6. Optionally, you can uncomment (remove the '#' from) `# printResponseLog()`to display the log element of the API response object.
7. Optionally, you can uncomment `# printResponseObj()`to display the entire API response object.

If everything works as expected, you should see a response like the following:

```
-The API response "status" is "0", indicating the action was successful.
```

If you observe any other response, then either the data set does not appear to be loaded in CAS or there was some other error. To further analyze, you may desire to uncomment the optional print functions for debugging.

Step 2.2 was based on this [SAS documentation]

2.3: RUN SAS DATA STEP CODE

Now that your WATER_CLUSTER_<initials> data set has been loaded into your Casuser library, it's time to begin preparing your Analytical Base Table (ABT). Perhaps an initial step toward preparing your ABT would be to create a new Boolean field based on the character field `US Holiday`. Since it's typically more efficient to perform operations on numbers compared with unstructured text, this is a common example of pre-processing to prepare data for analytics. Working with the SAS DATA step makes this relatively easy to accomplish using the following code:

```
data casuser.water_cluster_<initials> (replace=yes);
    set casuser.water_cluster_<initials>;
    if 'US Holiday'n = null then US_Holiday_Ind = 0;
    else US_Holiday_Ind = 1;
run;
```

The results of this code are that if there is any text in the `US Holiday` field, then the new indicator `US_Holiday_Ind` field will be populated with a 1, otherwise it will contain a 0.

Allowing users to run the SAS DATA step code through a REST API endpoint is an extremely powerful capability.

We execute the `dataStep.runCode` action:

1. Call `printLoadVrbls()` to verify the variable assignments.
2. Define both a URL and a payload.

3. Call `callEndpoint()`, which calls the REST API endpoint and sets `responseObj`.
4. Call `printResponse()`, which obtains and parses the response object.
5. Call `printPayload()` to verify the variable assignments.
6. Optionally, you can uncomment (remove the '#' from) `# printResponseLog()` to display the log element of the API response object.
7. Optionally, you can uncomment `# printResponseObj()` to display the entire API response object.

If everything works as expected, you should see a response like the following:

```
-The API response "status" is "0", indicating the action was successful.
```

Often, in addition to a status of 0, you should note that there might be additional elements in the response objects that could prove helpful. If you would like additional information from the REST API endpoint, you may want to review the following fields (note that a helper function `printResponseLog()` was created to assist your review and you can easily extend this playbook to parse additional fields):

```
responseObj.get('log')
responseObj.get('status')
responseObj.get('results')
```

If you observe any other response, then I would suggest debugging your SAS DataStep code in order to determine if there was some other error.

Step 2.3 was based on this SAS documentation

2.4: OPTIONAL – VERIFY THAT THE DATA STEP WORKED AS EXPECTED

Presuming that your SAS DATA step code ran successfully and replaced WATER_CLUSTER_<initials> data set in your Casuser library, you might desire to verify that the new `US_Holiday_Ind` field is ready for use.

We'll verify that the new `US_Holiday_Ind` field is available for use in CAS using the `table.columnInfo` action:

1. Call `printLoadVrbls()` to verify the variable assignments.
2. Define both a URL and a payload.
3. Call `callEndpoint()`, which calls the REST API endpoint and sets `responseObj`.
4. Call `printResponse()`, which obtains and parses the response object.
5. Call `printPayload()` to verify the variable assignments.
6. Call `printTable()` to cast the fields into a pandas DataFrame for ease of review.
7. Optionally, you can uncomment (remove the '#' from) `# printResponseLog()` in order to display the log element of the API response object.
8. Optionally, you can uncomment `# printResponseObj()` in order to display the entire API response object.

If everything works as expected, you should see a response like the following:

```
-The API response "status" is "0", indicating the action was successful.
```

If you observe any other response, then either the data set does not appear to be loaded in CAS or there was some other error. To further analyze, you might desire to uncomment the optional print functions for debugging.

The `printTable()` call should return the following output, showing your new field as #22:

Error! Reference source not found.: Expected `printTable()` Output

	0	1	2	3	4	5	6	7	8
0	Year		1	double	8	12		0	0
1	Month	Month	2	double	8	12		0	0
2	Day		3	double	8	12		0	0
3	Date		4	double	8	10	MMDDYY	10	0
4	Serial		5	double	8	4	BEST	4	0
5	Property		6	double	8	4	BEST	4	0
6	Address		7	char	28	28	$CHAR	28	0
7	City		8	char	7	7	$CHAR	7	0
8	Zip		9	double	8	5	BEST	5	0
9	Lat		10	double	8	10	BEST	10	0
10	Long		11	double	8	9	BEST	9	0
11	Property_type	Property_type	12	double	8	1	BEST	1	0
12	Meter_Location	Meter_Location	13	char	8	8	$CHAR	8	0
13	Clli		14	char	8	8	$CHAR	8	0
14	DMA		15	double	8	1	BEST	1	0
15	Weekday		16	double	8	12		0	0
16	Weekend		17	double	8	12		0	0
17	Daily_W_C_M3		18	double	8	12		0	0
18	Week	Week	19	double	8	12		0	0
19	US Holiday		20	char	27	27	$CHAR	27	0
20	CLUSTER	Cluster	21	double	8	12		0	0
21	null		22	char	27	27		0	0
22	US_Holiday_Ind		23	double	8	12		0	0

If you observe any other response, then either the data set does not appear to be loaded in CAS or there was some other error.

Step 2.4 was based on this SAS documentation

2.5: CREATE AN ANALYTICAL BASE TABLE (ABT)

The SAS DATA step code run successfully, replaced the WATER_CLUSTER_<initials> data set in your Casuser library, and created the new `US_Holiday_Ind` field that should be ready for use. The next step is to create a summarized aggregate ABT.

Working with SAS PROC MEANS makes this task relatively easy to accomplish:

```
proc means data=casuser.water_cluster_ab noprint;
    var Daily_W_C_M3;
    by Year Month City Clli Zip Meter_Location US_Holiday_Ind;
    output out=casuser.water_cluster_ab_ABT_proc (drop=_type_) mean= std=
min= max= /autoname;
run;
```

We execute the `aggregation.aggregate` action:

1. Call `printLoadVrbls()` to verify the variable assignments.
2. Define both a URL and a payload.
3. Call `callEndpoint()`, which calls the REST API endpoint and sets `responseObj`.
4. Call `printResponse()`, which obtains and parses the response object.
5. Call `printPayload()` to verify the variable assignments.
6. Call `printTable()` to cast the fields into a Pandas DataFrame for ease of review.
7. Optionally, you can uncomment (remove the '#' from) `# printResponseLog()` to display the log element of the API response object.
8. Optionally, you can uncomment `# printResponseObj()` to display the entire API response object.

If everything works as expected, you should see a response like the following:

```
-The API response "status" is "0", indicating the action was successful.
```

If you observe any other response, then either the aggregation does not appear to have worked as expected in CAS or there was some other error. To further analyze, you may desire to uncomment the optional print functions for debugging.

Step 2.5 was based on this SAS documentation

2.6: OPTIONAL – VERIFY THAT THE AGGREGATION WORKED AS EXPECTED

Presuming that your aggregation code ran successfully and created your ABT data set in your Casuser library, you might desire to verify that the new ABT is ready for use.

Like in 2.4, we verify that the new ABT is available for use in CAS using the `table.columnInfo` action:

1. Call `printLoadVrbls()` to verify the variable assignments.
2. Define both a URL and a payload.
3. Call `callEndpoint()`, which calls the REST API endpoint and sets `responseObj`.
4. Call `printResponse()`, which obtains and parses the response object.
5. Call `printPayload()` to verify the variable assignments.
6. Call `printTable()` to cast the fields into a pandas DataFrame for ease of review.
7. Optionally, you can uncomment (remove the '#' from) `# printResponseLog()` to display the log element of the API response object.

8. Optionally, you can uncomment `# printResponseObj()` to display the entire API response object.

If everything works as expected, you should see a response like the following:

```
-The API response "status" is "0", indicating the action was successful.
```

If you observe any other response, then either the data set does not appear to be loaded in CAS or there was some other error. To further analyze, you may desire to uncomment the optional print functions for debugging.

The `printTable()` call should return the following output, showing your summarized ABT:

Error! Reference source not found.: Expected `printTable()` Output

	0	1	2	3	4	5	6	7	8
0	Year		1	double	8	12		0	0
1	Month	Month	3	double	8	12		0	0
2	City		5	char	7	7	$CHAR	7	0
3	Clli		7	char	8	8	$CHAR	8	0
4	Zip		9	double	8	5	BEST	5	0
5	Meter_Location	Meter_Location	11	char	8	8	$CHAR	8	0
6	US_Holiday_Ind		13	double	8	12		0	0
7	_Daily_W_C_M3_Summary_Min_		15	double	8	12	BEST	12	0
8	_Daily_W_C_M3_Summary_Max_		16	double	8	12	BEST	12	0
9	_Daily_W_C_M3_Summary_Mean_		17	double	8	12	BEST	12	0
10	_Daily_W_C_M3_Summary_Std_		18	double	8	12	BEST	12	0

If you observe any other response, then either the data set does not appear to be loaded in CAS or there was some other error.

Step 2.6 was based on this SAS documentation

2.7: POTENTIAL NEXT STEPS – FETCH A SUBSET OF THE ABT DATA

Presuming that your aggregation code ran successfully and created your ABT data set in your Casuser library, a potential next step is to fetch a subset of the ABT data.

I might suggest that your next step in this process could be to use the `table.fetch` action documentation to return a number of rows from your ABT.

If I were going to build a fetch API call, I would start with the code from 2.6 above, update the url, and then potentially update the payload by adding a maximum number of rows to return, for example:

```
payload = {
        'table': {
            'caslib': destinationCasLib,
            'name': destinationDataAbt
        },
        'maxRows': 15,
    }
```

To view the data I would suggest creating another helper function similar to `printTable()`, but updated to parse the `table.fetch` response object.

Step 2.7 was based on this SAS documentation

End of Section 2 – Your ABT Should be Complete

If everything worked as expected, you have just created an ABT and promoted it to be used in other SAS Viya applications by completing these steps:

- Locate and load the WATER_CLUSTER data set from the Samples CAS library.

- Run a SAS DATA step that uses the RunCode action.
- Aggregate the data set with the equivalent of PROC MEANS and promote it to be used by other SAS Viya applications.

CONCLUSION

By following the steps in these sections, you should have been able to create your own Analytical Base Table (ABT) based on the WATER_CLUSTER SASHDAT file that's typically preconfigured in the Samples CAS library on SAS Viya.

As a reminder, for those of you who are familiar with the standard Sashelp library, Samples can be thought of as similar in nature to Sashelp.

The playbook steps were organized to help you get to your first "Hello World" as quickly as possible. You are encouraged to take this playbook and make it your own as you uncover additional value from learning and using the CAS REST APIs.

Once you have created a WATER_CLUSTER_<initials> ABT based on the suggested playbook steps, the resulting ABT can then be used for subsequent analysis and modeling in SAS Viya. Furthermore, you should be enabled to explore additional actions that are published as REST API endpoints so that you can build valuable assets with these APIs.

ACKNOWLEDGMENTS

Many thanks to everyone who reviewed this API playbook, especially:

Joe Furbee, https://www.linkedin.com/in/josephfurbee/

Matthew Perry, https://www.linkedin.com/in/matthew-perry-27606/

CONTACT INFORMATION

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

Andy Bouts
https://www.linkedin.com/in/andybouts/