Paper SAS6363-2016

REST at Ease with SAS®: How to Use SAS to Get Your REST

Joseph Henry, SAS Institute Inc., Cary, NC

ABSTRACT

Representational State Transfer (REST) is being used across the industry for designing networked applications to provide lightweight and powerful alternatives to web services such as SOAP and Web Services Description Language (WSDL). Since REST is based entirely on HTTP, SAS® provides everything you need to make REST calls and to process structured and unstructured data alike. This paper takes a look at how some enhancements in the third maintenance release of SAS 9.4 can benefit you in this area. Learn how the HTTP procedure and other SAS language features provide everything you need to simply and securely use REST.

INTRODUCTION

In the 2015 version of this paper (http://support.sas.com/resources/papers/proceedings15/SAS1927-2015.pdf), I presented a basic introduction on how to communicate with RESTful web services using the SAS DATA step and the HTTP procedure. This paper is an addendum to the 2015 paper, highlighting the additions that were made in the third maintenance release of SAS 9.4, as well as some updated techniques for reading and writing data.

This paper will work through an example of authenticating to the SAS Middle Tier using the SASLogon REST API to get a service ticket (ST) from the Central Authentication Server.

GETTING STARTED

To start, you are going to need a few utility macros. These macros will be used throughout the example and will make writing and debugging your code a bit simpler.

ECHOFILE

```
/*

Macro that simply echoes the contents of a fileref to the SAS log
*/
%macro echofile(file);

data _null_;
 infile &file;
 input;
 put _infile_;
run;

%mend;
```

CHECK_RETURN

```
/*
Check the returned status code against what is expected
*/
%macro check_return(code,expected);
%if &code ne &expected %then %do;
 %put ERROR: Expected &expected, but received &code;
 %abort;
%end;
%mend;
```

RESPONSE HEADERS

The first thing that you need to authenticate with the Central Authentication Server protocol is to POST your credentials as a form to SASLogon to create a ticket-granting ticket (TGT). When you are successful, you should get a 201 Created response code with a “Location” header that has the URL of the TGT. Sample SAS code to perform this first part is shown here:

```
%let username=sastrust@saspw;
%let pwd=Pass99;

filename input TEMP;
filename resp TEMP;
filename headers TEMP;

/*
 * Create the input file for the first request
 */
data _null_;
   file input recfm=f lrecl=1;
   put "username=&username.%nrstr(&password)=&pwd";
run;

proc http
 method="POST"
 url="http:// SASLogon.server:7980/SASLogon/v1/tickets"
 in=input
 headerout=headers
 out=resp

 HEADEROUT_OVERWRITE;
run;

%echofile(headers);
```

This code produces an HTTP request:

```
> POST SASLogon/v1/tickets
username=sastrust@saspw&password=Pass99
```

You might notice the new flag HEADEROUT_OVERWRITE in the procedure statement. This flag was added to make it easier to parse the return headers. There are some occasions during authentication or redirects that might cause the response headers to contain the headers for more than one response. This would cause you to have to write an additional bit of code to make sure you only parsed that LAST response. The HEADEROUT_OVERWRITE flag will make sure that the response headers fileref will only contain the headers from the final response as shown in Output 1.

```
HTTP/1.1 201 Created
Date: Fri, 15 Jan 2016 19:30:36 GMT
Server: Apache-Coyote/1.1
X-UA-Compatible: IE=edge
Location: http://SASLogon.server:7980/SASLogon/v1/tickets/TGT-297614-
44jv5P1RAfQ3rLszqlswaQehFZ0KpBo6SkqvRyfPlC49LG4i5F-cas
Content-Type: text/plain;charset=UTF-8
Content-Length: 0
Keep-Alive: timeout=5, max=100
Connection: Keep-Alive
```

Output 1. Partial SAS LOG from PROC HTTP Statement

PARSING RESPONSE HEADERS

After you have successfully created a TGT, you need to parse the response headers in order to get the URL of the TGT. The following code is a very simple way to parse the headers and store any needed data in macro variables:

```
%global hcode;
%global hmessage;
%global location;

data _null_;
 infile headers termstr=CRLF length=c scanover truncover;
  input @'HTTP/1.1' code 4. message $255.
        @'Location:' loc $255.

  call symputx('hcode',code);
  call symput('hmessage',trim(message));
  call symput('location',trim(loc));
run;
```

Because headers are structured, we can simply use the input statement in the DATA step to extract the values that we want. This code also shows how to easily extract the Status Line, which contains the status code of the request as well as a status message. After this code executes, you should have a macro variable, location, that has the URL needed in order to move onto the next step.

STATIC INPUT

The next step is quite similar to the first step. You are going to use the TGT to create a Service Ticket (ST) for a given URL. You do this by sending the URL for which you want a ticket as a POST form to the TGT URL. For example, if the URL that you want access to is http://sas.server:7980/SASWIPClientAccess/rest/modules/code, then you would send a request like this one:

```
> POST /SASLogon/v1/tickets/TGT-297614-
44jv5P1RAfQ3rLszqlswaQehFZ0KpBo6SkqvRyfPlC49LG4i5F-cas

service=http://sas.server:7980/SASWIPClientAccess/rest/modules/code
```

Previously, we executed a DATA step to create a fileref that had the formatted body, and that fileref was given to PROC HTTP as the IN value. In the third maintenance release for SAS 9.4, you can skip the DATA step if you have static text as shown below:

```
%let serviceurl= sas.server:7980/SASWIPClientAccess/rest/modules/code;
proc http
 method="POST"
 url="&location"
```

```
 in="service=http://&serviceurl."
  headerout=headers
 out=resp

 HEADEROUT_OVERWRITE;
run;

%echofile(headers);
%echofile(resp);
```

This feature is very useful for situations where the input is all static text. Not only does this feature save a bit of execution time and lines of code, but it also prevents sensitive information (like the user name and password from previous example) from making its way to temporary files on disk that might not be cleaned up right away.

After this code executes, you should have a fileref **resp** that contains the Ticket needed to access your service. Once again, using the input statement in the DATA step is a very efficient way of extracting this data:

```
%global ticket;

data _null_;
 infile resp;
 input @;
 call symput('ticket',trim(_infile_));
run;
```

HTTP DEFAULTS

Now you should have an ST, all you have to do is append the ST to the service URL with the query parameter **ticket** and make the call.

```
proc http
 url="sas.server:7980/SASWIPClientAccess/rest/modules/code?ticket=&ticket."
 out=resp
 headerout=headers
 ;
run;
```

This code produces an HTTP request:

```
> GET /SASWIPClientAccess/rest/modules/code HTTP/1.1
> Host: sas.server:7980
```

You should notice two things about the about code:

1. There was no METHOD specified in the procedure. In the third maintenance release for SAS 9.4, if METHOD and IN are both missing, the default METHOD is now a GET.
2. The URL does not have a protocol. That is, it is missing “HTTP://”. In the third maintenance release for SAS 9.4, if the URL does not have a protocol, it will default to “HTTP://”.

This concludes the Central Authentication Server example, but there a few more PROC HTTP features that can help you with REST service communication.

INPUT HEADERS

In the third maintenance release for SAS 9.4, it is much easier to pass in request headers. PROC HTTP has a new HEADERS statement. This statement makes it very easy to construct requests that contain user-defined headers:

```
filename out TEMP;
proc http
   url="http://httpbin.org/headers"
   out=out;
 headers
   "My-Header"="my value";
run;

%echofile(out);
```

```
{
  "headers": {
    "Accept": "*/*",
    "Host": "httpbin.org",
    "My-Header": "my value",
    "User-Agent": "SAS/9"
  }
}
```

Output 2. Response Body from INPUT HEADERS Example

AUTHENTICATION

In the third maintenance release for SAS 9.4, it is now possible to specify what type of authentication should be used in the HTTP procedure.

A good example of why you would want to do this, is when you want to authenticate to a server using Negotiate (Kerberos) with a specific user name and password. In previous releases, BASIC authentication would have always been tried first, but this would defeat the purpose of using a more secure form of authentication. An example of this is shown below:

```
proc http
   url="http://server.sas.com"
   webusername="testuser"
   webpassword="testpassword"
   /*
    * If needed, try and authenticate using Negotiate
    * or NTLM, but not BASIC
    */
   AUTH_NEGOTIATE
   AUTH_NTLM;
run;
```

This code would only try to authenticate using Negotiate or NTLM, but it would never try BASIC.

PERSISTANT CONNECTIONS AND COOKIES

The third maintenance release for SAS 9.4 has the ability to use persistent HTTP connections and HTTP cookies across multiple invocations of the HTTP procedure. This is very good for situations where you are making lots of individual calls to PROC HTTP or when the server uses cookies to prevent unneeded client activity.

PERSISTENT CONNECTIONS

Persistent connections are enabled by default, and an example is shown below:

```
filename out TEMP;
proc http
   url="http://www.sas.com"
   headerout=headers;
run;
```

%*ECHOFILE*(HEADERS);

The first time you run the preceding code you will get back some headers that look like this result:

```
< HTTP/1.1 200 OK
< Keep-Alive: timeout=15, max=99
...
```

If you run the code again, you see this result:

```
< HTTP/1.1 200 OK
< Keep-Alive: timeout=15, max=97
...
```

Notice how the max value goes does each time you run the code. This is because the same TCP connection is being used each time. This will save the step of having to resolve the host name and connect to the server, which especially saves time if you are making many small requests to the same server.

You can disable this feature by using the parameter NO_CONN_CACHE:

```
proc http
   url="http://www.sas.com"
   headerout=headers
   NO_CONN_CACHE;
run;

%echofile(headers);
```

COOKIES

Cookies can be a very useful thing, and now PROC HTTP uses them. A good example of why you would want to use cookies is authentication. A lot of times after you authenticate to a server, you will get a cookie that, if you use it, will prevent you from having to authenticate again. This can be a very large time saver in many situations.

Cookies are enabled by default, but you can disable them by using the argument NO_COOKIES.

CONCLUSION

SAS provides all the tools you need to use RESTful web services, and the third maintenance release for SAS 9.4 makes it even easier. Parsing and extracting information from response headers has been made more convenient with the addition of the HEADEROUT_OVERWRITE option. You now have the ability to send data and headers without the need for a DATA step. You also now have more control over many aspects of your HTTP request including: persistent connections, cookies, and authentication. The DATA step provides a convenient way to format URLs and input data and to interpret responses. The third maintenance release for SAS 9.4 improves upon an already powerful platform for use of just about any RESTful web service out there.

CONTACT INFORMATION

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

Joseph Henry
100 SAS Campus Drive
Cary, NC 27513
SAS Institute, Inc.
Joseph.Henry@sas.com

http://www.sas.com