Paper SAS4426-2020

REST Just Got Easy with SAS® and PROC HTTP

Joseph Henry, SAS Institute Inc.

ABSTRACT

Hypertext Transfer Protocol (HTTP) is the lifeblood of the web. It is used every time you upload a photo, refresh a web page, and in most every modern application you use today from your phone to your TV. A big reason why HTTP has become so ubiquitous with modern technology is because of a software architectural style known as Representational State Transfer (REST). REST has become the standard for interacting with independent systems across the web, which means that the consumption of REST APIs is, or is going to be, mandatory. The HTTP procedure in SAS® has enabled developers to use REST APIs simply and securely for many years now, but the demands of current systems keep increasing. PROC HTTP has been given quite a few updates to make using this great procedure even easier than it ever was before.

INTRODUCTION

The HTTP procedure has been a staple in SAS for many years for communicating with the web and Representational State Transfer (REST) APIs. Over the past few years, PROC HTTP has seen many updates that continue to increase its functionality and usability. SAS® Viya 3.5 continues this trend by introducing some new syntax to the procedure that makes many common tasks simpler as well as more accessible to newer SAS developers who might be used to other HTTP frameworks. The enhancements addressed in this paper are also available in SAS®9 as part of the second release of SAS 9.4M6.

BASIC INTRODUCTION TO HTTP COMMUNICATION

HTTP is simply a way for clients and servers to communicate. Most of the time the client is the web browser you are using, and the server is the computer hosting the website you are looking at. In the case of SAS, the client is PROC HTTP and the server is typically the web service you want to access.

The location of a server is identified by its web address, which is a Uniform Resource Locator (URL). A URL is broken up into a few pieces like:
http(s)://[*authority*]*path*[?*query*][#*fragment*].

The authority is the hostname (e.g., sas.com) while the path makes up what we call the endpoint. The endpoint is the "resource" you are acting on. Many times, simply calling the endpoint is enough, but most of the time you need to pass information of what you want to do in the form of parameters.

QUERY PARAMETERS

The query string is the part of the URL between the first question mark (?) and either the end of the URL or a number sign (#). In a REST API, the query string is commonly used to pass parameters to an endpoint. Each query parameter typically consists of a name-value pair. Each name-value pair is separated by an ampersand (&) like:

http://httpbin.org/get?firstname=Joseph&lastname=Henry

This URL contains two query parameters. Since the query string is simply part of the URL, the simplest way to pass query parameters in PROC HTTP is to just use the URL option like:

```
proc http
 url="httpbin.org/get?firstname=Joseph&lastname=Henry";
run;
```

This is simple enough, but chances are you will want your code to be a bit more flexible with the parameters, which probably means using macro substitution for the values. The problem that you will run into, is that query parameters are separated by an & and SAS macro variables are dereferenced by an &, which causes a conflict. This leads to having to surround the & in the URL with %NRSTR() like this:

```
%let firstname=Joseph;
%let lastname=Henry;

proc http
 url="httpbin.org/get?firstname=&firstname%nrstr(&lastname)=&lastname";
run;
```

That is a simple enough fix to be able to differentiate between the & in the URL and the & for SAS macro, but what if the query parameter value itself contains an &? What if it contains one of the other reserved characters for a URL?

The answer is you need to URL-encode the special characters in order to differentiate between a character that means something special for a URL and just a plain old character. You can accomplish this using the DATA step function URLENCODE() like:

```
%let firstname=Joseph;
%let lastname=Henry;
%let company=Stuff & Things Inc.;

data _null_;
 encoded = urlencode("&company.");
 call symputx("encoded_company",encoded,G);
run;

proc http
  url="httpbin.org/get?firstname=&firstname%nrstr(&lastname)=&lastname%nrst
  r(&company)=&encoded_company";
run;
```

While this does work, your code can start to get cluttered with extra %NRSTR(), and having to run a DATA step each time adds more overhead and extra code. Luckily a new option in SAS Viya 3.5 makes this much easier.

QUERY OPTION

The QUERY= option was added in SAS Viya 3.5 as a way to easily add query parameters without the need for an external DATA step or using %NRSTR(). The previous example can be rewritten like:

```
%let firstname=Joseph;
%let lastname=Henry;
%let company=Stuff & Things Inc.;

proc http
  url="httpbin.org/get"
  query = ("firstname"="&firstname"
           "lastname"="&lastname"
           "company"="&company");
  run;
```

This is much cleaner and makes the code much more readable. A few things to note:

1.) A query string will be generated from the name-value pairs inside of the (). This string will be appended to any existing query string in the URL.

2.) The ? will be added if it does not exist.

3.) All name-value pairs will be joined with an = and separated with an &.

4.) All name-value pairs will be URL-encoded unless the name-value pair is preceded with the token NOENCODE.

FORM DATA PARAMETERS

Another very common way to send parameters is by using forms. You are probably very familiar with what a form is already, as they appear on the web all the time as a way to enter parameters for a request. An example form is shown below in Figure 1.

Figure 1

Parameters in a form are very similar to parameters in a query string, except that form parameters are sent in the body of the request instead of in the URL as query parameters are.

Form parameters are normally sent as a POST request with the content-type application/x-www-form-urlencoded. Like query parameters, the content is typically URL-encoded, and each name-value pair is separated with an &.

To send the same parameters as the previous example as a form POST, you would do something like this:

```
%let firstname=Joseph;
%let lastname=Henry;
%let company=Stuff & Things Inc.;

data _null_;
 encoded = urlencode("&company.");
 call symputx("encoded_company",encoded,G);
run;

proc http
 url="httpbin.org/post"
 method=POST
 in =
    "firstname=&firstname%nrstr(&lastname)=&lastname%nrstr(&company)=&encod
    ed_company";
run;
```

Notice that once again, an external DATA step might be needed to handle special characters (namely the &). Also note that the use of %NRSTR() is needed to explicitly use a literal & in the input.

While this works fine, the need to have an extra DATA step and multiple uses of %NRSTR() makes the code less readable and more complicated than it ideally should be. Like before, a new option in SAS Viya 3.5 makes this much easier.

FORM INPUT

The IN= FORM option was added in SAS Viya 3.5 as a way to easily send form parameters without the need for an external DATA step call or using %NRSTR(). The previous example can be rewritten like:

```
%let firstname=Joseph;
%let lastname=Henry;
%let company=Stuff & Things Inc.;

proc http
 url="httpbin.org/post"
 method=POST
 in = FORM ("firstname"="&firstname"
            "lastname"="&lastname"
            "company"="&company");
run;
```

This might not seem like a huge difference, but as the number of parameters increase, readability and ease of programming is greatly increased.

Like the QUERY= option, arguments are URL-encoded by default, but you can choose to have individual parameters not encoded by using the NOENCODE option like:

```
in = FORM ("firstname"="&firstname"
           NOENCODE "lastname"="&lastname"
           "company"="&company");
```

MULTIPART DATA

Multipart requests have been around for a very long time as part of certain web forms, but are now making their way into more REST APIs. A multipart request is basically a way to send one or more different sets of data combined in a single request body. This type of request is typically used for something like uploading a file along with metadata about the file.

An example would be that you need to upload a file to a cloud-storage provider. The file is simply a stream of data stored somewhere in the cloud, but you need to give it a display name (a human-readable name). The information (metadata) about the file (including the name) would be described via a JSON body like:

```
{
    "name": "Picture"
}
```

Using multipart, it would be possible to send the JSON along with the file contents in a single request. This not only simplifies using the API, it also speeds up performance since a lot of the overhead with REST APIs is connecting to the web server.

Prior to SAS Viya 3.5, performing a multipart upload was not straightforward. You had to know quite a bit about how multipart data was encoded, but it was possible. An example of uploading a File with the above JSON in a multipart request would look something like this:

```
/*Image to upload*/
filename image "profile.png";

/* must create a boundary string for multipart */
%let boundary=%sysfunc(uuidgen());

/* File descriptor (metadata) */
filename meta TEMP;
data _null_;
  file meta recfm=f lrecl=1;
  put "{";
  put """name"": ""profile.png"",";
  put "}";
run;

/* Temp file to hold our formatted multipart input*/
filename in TEMP;

data _null_;
    file in termstr=CRLF;
   if _n_ = 1 then do;
         /*
         This first part is our JSON data.
       */
         put "--&boundary.";
         put 'Content-Type: application/json';
         put ;
         put "{";
         put " ""name"": ""Picture"" ";
         put "}";
         put "--&boundary.";
         put 'Content-Type: image/png'
         put ;
         /* end here for now. Next step we will append the file */
```

```
     end;
run;

/* append the meta file */
data _null_;
    file in mod recfm=f lrecl=1;
   infile meta recfm=f lrecl=1;

    input;
    put _infile_;
run;

/* end the meta part and start the next part*/
data _null_;
    file in termstr=CRLF;
         put ;
         put "--&boundary.";
         put 'Content-Type: image/png';
         put ;
         /* end here for now. Next step we will append the file */
run;

/* open the temporary file and append the file to upload*/
data _null_;
    file in mod recfm=f lrecl=1;
   infile image recfm=f lrecl=1;

    input;
    put _infile_;
run;

/* Add the final boundary*/
data _null_;
    file in mod termstr=CRLF;
   put;
    put "--&boundary--";
run;

proc http
    method="post"
    url = "httpbin.org/post"
    in = in
    ct="multipart/related; boundary=&boundary.";
run;
```

That is a lot of code. Not only is it hard to read and understand, but it would be very specific to the particular API you are using, which means it would be very hard to write a general-purpose usage.

Once again, SAS Viya 3.5 introduces a way to upload multipart data quite easily.

MULTIPART INPUT

SAS Viya 3.5 introduces another type of input named MULTI to allow for easy multipart uploads. The syntax is:

IN = **MULTI** "*type*" (*input* <**HEADER**="*header*">*,...)
TYPE specifies what type of multipart will be used, for instance `"related"` will translate into sending a content-type of multipart/related.

Inside the parenthesis is a comma-separated list of inputs where INPUT is either a string or a fileref, which can be followed by zero or more headers.

The previous code can be rewritten simply as:

```
filename image "profile.png";
filename meta TEMP;
data _null_;
  file meta recfm=f lrecl=1;
  put "{";
  put """name"": ""photo"",";
  put "}";
run;

proc http
 url="httpbin.org/post"
 method=POST
 in = multi "related" ( meta header="Content-Type: application/json",
                        image header="Content-Type: image/png");
run;
```

This is a big improvement on the previous version, and actually makes sending multipart requests with PROC HTTP viable.

CONCLUSION

SAS Viya 3.5 adds a few major improvements to PROC HTTP that not only improve usability but add significant functionality to an already powerful procedure. PROC HTTP will continue to evolve in future SAS releases with the goal of eliminating barriers and simplifying use. SAS and PROC HTTP should provide everything you need in order to use any REST API you need.

REFERENCES

Fielding, Roy, and Julian Reschke. 2014. "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing." RFC 7230. June 2014. Internet Engineering Task Force (IETF). Available at https://tools.ietf.org/html/rfc7230.

Berners-Lee, T., R. Fielding, and L. Masinter. 2005. "Uniform Resource Identifier (URI): Generic Syntax." STD 66, RFC 3986. January 2005. Internet Engineering Task Force (IETF). Available at https://tools.ietf.org/html/rfc3986.

Freed, N., and N. Borenstein. 1996. "Multipurpose Internet Mail Extensions (MIME) Part Two: Media Types." RFC 2046. November 1996. Internet Engineering Task Force (IETF). Available at http://www.ietf.org/rfc/rfc2046.txt.

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