Paper 4209-2020

A Beginners Guide to Consuming RESTful Web Services in SAS®

Laurent de Walick, PW Consulting

ABSTRACT

Web services are a method to exchange data between applications or systems using web technology like HTTP and machine-readable file formats like XML and JSON. Representational State Transfer (REST) is the most popular architecture used to implement web services. Web services using the REST architecture are called RESTful web services.

In recent years SAS has included procedures and libname engines for all standards to support consuming RESTful web services.

This paper presents how web services can be consumed in SAS. It will explore the PROC HTTP and discuss the different options that must be set correctly to consume a web service. It shows how parameters can be generated from existing SAS data using PROC STREAM and can be submitted when calling a web service. And finally, it describes how the output from a web service can be read into SAS using the JSON and XML libname engine.

INTRODUCTION

There is a big chance you wanted to use data from an online service for some analysis. You can scrape the data from a website, or download it manually, but this is often not desirable. If you investigated how to automate the process to acquire the data, you've most likely come across the term REST API. So, what is a REST API?

API

An Application Programming Interface, or API, is an interface between two or more applications. The API is a set of rules that allow multiple applications to communicate with each other. This can be as simple as returning data from a database, but also perform complex calculations and return the results. The application is only allowed to connect to endpoints for posting or reading data, making it a secure method to allow to applications to interoperate.

HTTP(S)

HTTP stands for HyperText Transfer Protocol and is a client server protocol that it the foundation of any data exchange on the web. Web Services also rely on HTTP to exchange data between the client and server. HTTP send information in plain text and is not secure. HTTPS is the secure variant that encrypts data in transit. SAS supports HTTP, making it possible to use web services from SAS.

Authentication

Authentication is the process of identifying the client who is doing a request. HTTP supports multiple authentication schema such as anonymous authentication and basic authentication.

In basic authentication passwords are encoded but not encrypted and not considered secure. This might be enough for internal applications, in combination with HTTPS, but very few public APIs will use on basic authentication. They will use the anonymous schema and use on authentication at the application level.

REST

The most popular API standard for web applications is REST. This determines how the API looks like. REST stands for Representational State Transfer and was defined in 2000 by Roy Fielding in his PhD dissertation "Architectural Styles and the Design of Network-based Software Architectures" [1].

The REST architecture is based on a client/server model. A stateless protocol is used for communication between client and server. Accessing a REST web service is called a request. The data returned by the web service is the response.

Request

A request consists of 4 elements

- endpoint
- header
- method
- data (optional)

Endpoint

The endpoint is the internet address where the web service can be accessed. It is Uniform Address Location (URL) and typically has the following format.

```
https://api.example.url/users/memberships?type=free&sort=lastname
```

In the above example the root-endpoint is `https://api.example.url` and `/users/memberships` are the path to a specific web service. The final part of the endpoint, `?filter=free&sort=lastname`, is optional. This the query string and can be used to add parameters to the web service.

Method

The method defines the type op request send to a web server. It indicates the action to perform for the requested resource. Possible actions are Create, Read, Update or Delete (CRUD). The method that supports those actions are GET, POST, PUT, PATCH and DELETE. Methods are case sensitive and should always be upper cased.

Method	Type of action
GET	Read a representation of a resource. The web service looks for data and sends the results back.
POST	Create new resources; Create new entries in the database
PUT or PATCH	Update existing resources; Change existing records in the database
DELETE	Delete a resource; Remove records from the database

Table 1. Common methods

Not all web services accept all methods. Each web service should have documentation available that describes what method is valid.

For example, the following request will return a list of all available users.

```
GET https://api.example.url/users/
```

In the next example a new user with the name Mike will be created.

```
POST https://api.example.url/users/mike
```

Header

Headers are used to send additional information to with a request. The information is needed by the server to understand how it should process the request. A header consists name-value pairs that are formatted by its case-insensitive name followed by a colon (:) and then by its value. A header can contain any number of name-value pairs.

The following header tells the server that the server can expect data in the JSON format (Content-Type), but that the client expects the result to be in XML (Accept).

```
"Content-Type: application/json"
"Accept: application/xml"
```

Data

The data, also called the body, message or payload, contains information that is send to the server. Data is only valid when using the POST, PUT, PATCH or DELETE methods.

Response

The response consists of a header and data. Each response also has a status code indicating how the request was handled.

HTTP Status Codes

The status codes are part of the HTTP protocol and can be used to determine quickly if a request has been complete successfully or failed and why. The status codes are grouped in five classes:

- Informational responses (100–199)
- Successful responses (200–299)
- Redirects (300–399)
- Client errors (400–499)
- Server errors (500–599)

Most status codes are defined in the HTTP/1.1 standard (RFC 7231), but servers can return non-standard code. If the code is not standard, the client should be able to determine the type of errors by the class.

JSON

In the past XML was the de facto standard used to exchange data between systems. The rise of SOAP as the default for APIs was an important driver for the popularity of XML. The introduction of REST was paired to the rise of JavaScript Object Notation (JSON) as format for data exchange.

JSON is an open-standard file format or data interchange format that uses human-readable text to transmit or store data objects consisting of attribute–value pairs and array data types. It is lightweight and the most common data format used by REST web services.

The REST architecture does not mandate the use of a specific format to exchange data. Both JSON and XML can be used and it's up to the published of the service to select the desired format.

JSON versus XML

- Both JSON and XML are "self-describing"
- Both JSON and XML are hierarchical
- Both JSON and XML have well-documented open standards (RFC 7159, RFC 4825)
- JSON is smaller. For the same data, JSON is almost always significantly smaller, leading to faster transmission and processing.
- XML separates data from metadata via the use of attributes and elements
- XML supports mixed content

CONSUMING WEB SERVICES USING SAS

Now that the basic concepts have been explained it's time to discuss how web services can be access from SAS. REST web services use HTTP and SAS provides to methods to access URLs over HTTP:

- The FILENAME statement with URL access method
- The HTTP procedure

The filename statement only supports the GET method and can only be used to read data. PROC HTTP supports any method that conforms to the HTTP standard and can be used for the other methods.

GET REQUEST

In the first example we will do a get request to The Open Brewery DB [2]. The Open Brewery DB is a free API for public information on breweries, cideries, brewpubs, and bottle shops. In the following example we will use the API to retrieve a list of Brewpubs in cities named Washington. This is access from the following endpoint:

```
https://api.openbrewerydb.org/breweries?by_city=washington&by_type=brewpub
```

FILENAME statement

The FILENAME statement with the URL access method creates a file reference to an online location. The FILENAME statement uses the following syntax:

```
FILENAME fileref URL http://url.to/web-service-endpoint ' <url-options>;
```

We use a data step to read its contents and write them to the SAS log. We also add the DEBUG option to have the HTTP headers written to the log.

```
FILENAME request HTTP
'http://api.openbrewerydb.org/breweries?by_city=washington&by_type=brewpub'
DEBUG;

DATA _NULL_;
 INFILE request;
 INPUT;
 PUT _INFILE_;
RUN;
```

When run, the code returns the following output to the log:

```
NOTE: >>> GET /breweries?by_city=washington&by_type=brewpub HTTP/1.0
NOTE: >>> Host: api.openbrewerydb.org
NOTE: >>> Accept: */*
NOTE: >>> Accept-Language: en
NOTE: >>> Accept-Charset: iso-8859-1,*,utf-8
NOTE: >>> User-Agent: SAS/URL
NOTE: >>>
NOTE: <<< HTTP/1.1 200 OK
NOTE: <<< Date: Fri, 07 Feb 2020 09:22:28 GMT
NOTE: <<< Content-Type: application/json; charset=utf-8
NOTE: <<< Connection: close
NOTE: <<< Set-Cookie: __cfduid=d4fc8d90c3c418c08b92614f29bba8dd91581067347;
expires=Sun, 08-Mar-20 09:22:27 GMT; path=/;
      domain=.openbrewerydb.org; HttpOnly; SameSite=Lax; Secure
NOTE: <<< Cache-Control: max-age=86400, public
NOTE: <<< Etag: W/"fab459aaa6d4afec7b8ccb593d6eec4b"
NOTE: <<< X-Request-Id: 266e578e-1eab-4010-bd23-a3014d9d0163
NOTE: <<< X-Runtime: 0.414939
NOTE: <<< Strict-Transport-Security: max-age=31536000; includeSubDomains
NOTE: <<< Vary: Origin
NOTE: <<< Via: 1.1 vegur
NOTE: <<< CF-Cache-Status: DYNAMIC
NOTE: <<< Expect-CT: max-age=604800, report-uri="https://report-
uri.cloudflare.com/cdn-cgi/beacon/expect-ct"
NOTE: <<< Server: cloudflare
NOTE: <<< CF-RAY: 56144ca9dc231762-FRA
NOTE: <<<
NOTE: The infile REQUEST is:

Filename=http://api.openbrewerydb.org/breweries?by_city=washington&by_type=
brewpub,
      Local Host Name=sasglobalforum2020.paper.host,
      Local Host IP addr=10.10.10.10,
      Service Hostname Name=api.openbrewerydb.org,
      Service IP addr=104.24.124.147,
      Service Name=N/A,Service Portno=443,
      Lrecl=32767,Recfm=Variable
[{"id":1768,"name":"Bluejacket","brewery_type":"brewpub","street":"300
Tingey St SE","city":"Washington","state":"District of Columb
ia","postal_code":"20003-4625","country":"United States","longitude":"-
77.0006981","latitude":"38.8750965","phone":"","website_url":
"http://www.bluejacketdc.com","updated_at":"2018-08-
24T00:26:14.349Z","tag_list":[]},

...

{"id":1774,"name":"Gordon Biersch Brewery Restaurant - Navy
Yard","brewery_type":"brewpub"
,"street":"100 M St SE","city":"Washington","state":"District of
Columbia","postal_code":"20003-3519","country":"United States","lon
gitude":"-
77.0052971","latitude":"38.8766834","phone":"2024842739","website_url":"htt
p://www.gordonbiersch.com/locations/navy-yard?a
ction=view","updated_at":"2018-08-24T00:26:16.619Z","tag_list":[]}]
NOTE: 1 record was read from the infile REQUEST.
      The minimum record length was 3763.
      The maximum record length was 3763.
```

Let's examine the output. 3 lines are highlighted yellow are discussed next.

The first highlighted line is from the header send by SAS to the server in the request. SAS tells the server that it accepts any kind of data.

```
NOTE: >>> Accept: */*
```

The next line is the HTTP status code returned with the response. The return code is 200 OK indicating that the request was successful.

```
NOTE: <<< HTTP/1.1 200 OK
```

The third and last highlighted line tells SAS that the output in the response is in json format and in the UTF-8 character set.

```
NOTE: <<< Content-Type: application/json; charset=utf-8
```

The data in the response is highlighted in gray. This is the JSON file that the web services returned. To be able to process the data we don't want the output written to the SAS log, but need it in a dataset.

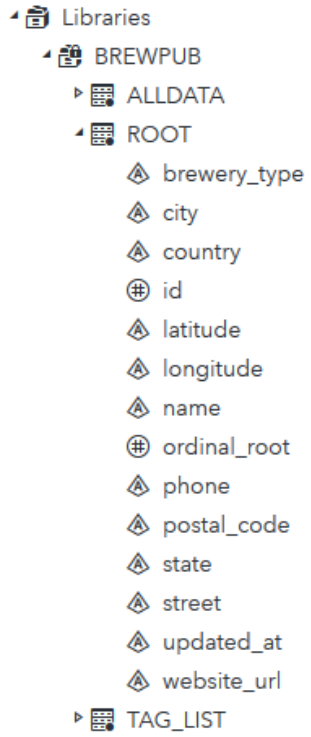
LIBNAME JSON Engine

SAS introduced the JSON LIBNAME engine in Maintenance 4 of SAS 9.4. With the libname engine it is possible to read data from JSON like it is a data set. It is a read-only library and the JSON file is read only once, when the JSON engine LIBNAME statement is assigned. To read the JSON file again, you must reassign the JSON libref.

For the brewpub request the LIBNAME statement has the following format:

```
LIBNAME brewpub JSON FILEREF=request;
```

This creates a library with several tables. Because the content is different for each JSON file, the libname engine always creates an ALLDATA and ROOT and root table. Other tables are created based on the data in the JSON file. It's like the XML engine some people might be familiar with. For the Open Breweries API this is the results:



Display 1 Contents of the BREWPUB library

In this case the JSON structure is flat and all data is in the ROOT table. The next code prints a list of brewpubs in Washington DC:

```
proc print data=brewpub.root (where=(state='District of Columbia'));
 var name street phone;
run;
```

That leads to the following results:

Obs	name	street	phone
1	Bluejacket	300 Tingey St SE	
2	District ChopHouse and Brewery	509 7th St NW	2023473434
3	Gordon Biersch Brewery Restaurant - Washington	900 F St NW	2027835454
4	Right Proper Brewing Company	624 T St NW	2026072337
5	The Public Option	1601 Rhode Island Ave NE	2023975129
6	Gordon Biersch Brewery Restaurant - Navy Yard	100 M St SE	2024842739

Output 1 Contents of the ROOT data set

HTTP Procedure

PROC HTTP can be used for more complex HTTP request, but nothing prohibits from using it for simple GET requests too. PROC HTTP has the following syntax;

```
proc http
 url="http://url.to/web.service.endpoint"
 method=POST
 in=request
 out=response;
 headers
  "Content-Type"="application/json"
  "Accept"="application/json";
run;
```

The next code retrieves all breweries in Washington. The arguments provided to PROC HTTP are:

- URL: The endpoint of the web service the request if for. This is the only mandatory argument.
- Method: The method used in the request. GET is the default value and the argument can be omitted in this case.
- Out: The destination of the output. In this case we create a file reference to temporary location and assign a JSON libname to the response fileref.

```
filename response temp;

proc http
 url="https://api.openbrewerydb.org/breweries?by_city=Washington"
 method=GET
 out=response;
run;
```

The response is a JSON file with information about all the Washington breweries.

```
[
    {
        "id": 1767,
        "name": "Bardo Brewpub",
        "brewery_type": "micro",
        "street": "25 Potomac Ave SE",
        "city": "Washington",
…
        "phone": "",
        "website_url": "",
        "updated_at": "2018-08-11T21:39:47.705Z",
        "tag_list": []
    }
]
```

The LIBNAME JSON engine can be used again to make the data in the JSON file useable in SAS.

POST REQUEST

The GET request is easy to use but has limited possibilities. In the next example the request is a post request where data is sent in the body. The data in the body is a new person records this is "saved" in the database. The example uses the free Dummy Rest API Example [3] service that simulates a POST action and does not actually write the create to a database.

The body is a JSON file that is placed somewhere on the SAS server. The JSON file looks like:

```
{
 "name":"Laurent",
 "age":"40"
}
```

PROC HTTP is used to use the web server. A POST request is done where the JSON is send as body content. The PROC HTTP requires the next arguments

- URL: Endpoint to the web service
- Method: POST
- In: File reference to the JSON that is send as body content
- Out: File reference to the results that are returned by the web service
- CT: The mime type of the body, in this case application/json because our body content is in the JSON format.
- PROC HTTP also has a debug option. In this case it's set to 1.

```
filename payload '/data/payload.json';

proc http
 url="http://dummy.restapiexample.com/api/v1/create"
 method="POST"
 in=payload
 out=response
 ct="application/json";
 debug level=1;
run;
```

When executed the following is written to the log:

```
> POST /api/v1/create HTTP/1.1
> User-Agent: SAS/9
> Host: dummy.restapiexample.com
> Accept: */*
> Connection: Keep-Alive
> Content-Length: 56
> Content-Type: application/json
>
< HTTP/1.1 200 OK
< Accept-Ranges: bytes
< Access-Control-Allow-Origin: *
< Cache-Control: no-store, no-cache, must-revalidate
< Content-Type: application/json;charset=utf-8
< Date: Fri, 07 Feb 2020 12:35:34 GMT
< Expires: Thu, 19 Nov 1981 08:52:00 GMT
< Host-Header: c2hhcmVkLmJsdWVob3N0LmNvbQ==
```

```
< Server: nginx/1.16.0
<
NOTE: PROCEDURE HTTP used (Total process time):
      real time           0.59 seconds
      cpu time            0.04 seconds
```

The highlighted parts in from the log tells that a POST request was done with a body containing JSON. The request was successful (200 HTTP status code) and JSON body was returned.

The JSON body contains the following :

```
{
  "status": "success",
  "data": {
    "name": "Laurent",
    "salary": null,
    "age": "40",
    "id": 99
  }
}
```

Name and age correspond to the values I put in the body. Because no salary was posted it is set to null by the service. A new id is generated for the record.

DELETE Request

The DELETE request is like a GET request. An endpoint is accessed with the DELETE method and a record is deleted from the database.

```
proc http
 url="http://dummy.restapiexample.com/api/v1/delete/21"
 method="DELETE"
 out=response
 ct="application/json";
 debug level=1;
run;
```

ADDITIONAL OPTIONS

JSON maps

The JSON libname engine has an automap function that generates a data set for each object in the JSON. It also generates a data set named ALLDATA, which contains all JSON information in a single data set.

There are situations where the automap function is not the optimal solution to organize data in library. Complex hierarchies in the JSON file can lead to many datasets and datatypes are not always determined correctly. To improve data management the JSON libname engine has the possibility to provide a map file that that maps the data in data sets according to a user definition. In addition, the map file can be used to set the length, format and informat of each column.

Using a map file can also improve performance when assigning a library on large JSON files. The automap function needs to read and parse all data to be able to determine what data sets are created. A JSON map contains this information and reduces the initial work.

The 2018 Global Forum Paper "Using Maps with the JSON LIBNAME Engine in SAS" [4] explains in more detail how a JSON map can be created.

XML

Extensible Markup Language (XML) is a markup language to create documents that are both human and machine-readable. The specification for XML is defined in a free and open standard. XML is widely used for the representation of arbitrary data structures such as those used in web services [5].

Many standards use XML to structure and exchange data. As a result, XML is also a common format to exchange data using REST APIs. SAS has been able to process XML since SAS 9.1 using a libname engine. In SAS 9.2 an improved version of the libname engine was introduced.

The XML Libname engine works much like the JSON LIBNAME Engine. It also has an automap feature that generates data sets based on the contents and supports map files to describe data and have a user define the contents of the library. A libref for the XMLV2 engine can be assigned to either a specific XML document or to the physical location of a SAS library in a directory-based environment. [6]

```
LIBNAME response XMLV2 XMLFILEREF=response XMLMAP=mapflref;
```

Authentication

Many web services require the consumer to authenticate before a service can be consumed. The FILENAME statement with URL Access method only supports basic authentication while PROC HTTP supports most common authentication methods.

- Basic Authentication: Uses the WEBUSERNAME and WEBPASSWORD arguments to submit the correct credentials in the HTTP call.
- NTLM or Kerberos: Uses the current user running the SAS process to authenticate.
- OATH_BEARER: For service that user OAuth authentication.

OAuth Authentication

Many web services are moving to OAuth for authentication. OAuth is an open standard authorization framework that allows a user to grant a website or application limited access to an HTTP service. OAuth works using tokens to authenticate and authorize an application to access a service

The 2017 SAS Global Forum Paper "Show Off Your OAuth" [2] contains a step by step explanation on how to implement OAuth in SAS.

Proxy

Many SAS servers can only access outside service through a HTTP proxy server. If this is the case the proxy server and optional credentials can be provided in PROC HTTP.

- PROXYHOST: The hostname of the proxy server
- PROXYPORT: The port the proxy server listens to
- PROXYUSERNAME: A username required to login to the proxy server (optional, only needed if server requires credentials)
- PROXYPASSWORD: The password for the username

CONCLUSION

Improvements in PROC HTTP and the addition of the JSON libname engine has simplified the use of REST APIs in SAS. With many REST web services available online, adding third party or open data to SAS projects has never been easier.

REFERENCES

[1] Wikipedia, "Representational State Transfer," 2019. [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer.

[2] "Open Brewery DB API Documentation," [Online]. Available: https://www.openbrewerydb.org/documentation.

[3] "Dummy Rest API Example," [Online]. Available: http://dummy.restapiexample.com/.

[4] A. Gannon, "Using Maps with the JSON LIBNAME Engine in SAS," in *SAS Global Forum*, 2018.

[5] Wikipedia, "XML," [Online]. Available: https://en.wikipedia.org/wiki/XML.

[6] J. Henry, "Show Off Your OAuth," in *SAS Global Forum*, 2017.

[7] SAS, "LIBNAME Statement: JSON Engine," in *SAS® 9.4 Global Statements: Reference*, 2020.

[8] SAS, SAS® 9.4 XMLV2 and XML LIBNAME Engines: User's Guide, 2020.

[9] SAS, "HTTP Procedure," in *Base SAS® 9.4 Procedures Guide, Seventh Edition*, 2020.

RECOMMENDED READING

- *Base SAS® Procedures Guide*
- *SAS® 9.4 XMLV2 and XML LIBNAME Engines: User's Guide*

CONTACT INFORMATION

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

Laurent de Walick
PW Consulting
laurent.de.walick@pwconsulting.nl
https://www.pwconsulting.nl