Paper 117-2022

Unlocking the Web With Python and SAS: Shortcuts to Accessing Data Using Python and SAS

Joe Matise, NORC at the University of Chicago

ABSTRACT

Learning that you need to access your data through a REST API can be intimidating for data scientists, with complicated syntax, needing to manage authorization keys or login credentials, and no helpful SAS code anywhere to be seen for most. The good news is that you do not have to be a web guru to use them - you just need to use the right tools.

In this paper, we will show examples of using Python pulled directly from API documentation to connect to REST APIs, and then show how to get that data into SAS or Viya.

No prior knowledge of Python or REST APIs is expected or required. Most SAS programmers capable of writing basic data step code should understand the level of code we use in this example.

INTRODUCTION

Increasingly often, accessing data means downloading it over the web. Whether that data is publicly available data or data from a private system, most of the time it is available using a REST API. APIs can seem difficult to understand and complicated to implement, especially for SAS programmers used to data that is accessed using LIBNAMEs and the local file system.

Fortunately, there is an option to simplify this work, by taking advantage of code developed by others. Thanks to the popularity of Python, many APIs have example code and/or packages that are freely available and easily discovered – often in the API documentation itself.

Even for users with no Python knowledge whatsoever, these examples are often runnable exactly as-is or with minimal modifications. Once that's been completed, use the Python as an outline for how the SAS code needs to be written, and convert to SAS code piece by piece.

ACCESSING DATA USING AN APPLICATION PROGRAMMING INTERFACE (API)

WHAT IS AN API?

An API, or an Application Programming Interface, is a method by which two programs can communicate. APIs of one form or another exist in nearly every computer application – smartphones have many APIs to allow programs to interact with its operating system, for example, and SAS has many APIs of its own for communicating with SAS, Viya, and other SAS components.

APIs define the grammar by which a program accepts input. That definition includes what methods the program accepts (think of these as the verb in a sentence – the thing the application should do), the resources the API makes available (think of these as nouns – things that can be accessed), and the parameters or properties the methods accept (which play the role of adjectives or adverbs, though of course they are usually also nouns).

For example, every computer operating system has an API for placing a window on the screen. It might have a resource `Window`, a verb `create`, and parameters like `positionX`, `positionY`, `sizeX`, `sizeY`. It might be called like so:

```
Window.create(positionX:50, positionY:25, sizeX:30, sizeY:20)
```

That would create a window on the screen at the specified position with the specified size.

APIs often do not require a specific programming language to be used to utilize them. So long as the program uses the right grammar (the methods, resources, and properties the API specifies), the API will accept the input and provide a response.

APIS FOR ACCESSING DATA: THE REST API

In the context of this paper, the kind of APIs we refer to are those used to access data from websites, whether publicly accessible or data that is restricted to certain users. While there are many kinds of APIs for accessing data, the most commonly seen kind of APIs are REST APIs.

REST (or RESTful) APIs are APIs that conform to a particular style. They are for the most part stateless – meaning each request is separate, nothing is saved in between; the server does not need to keep track of anything. They are like the order counter at a fast-food place – the customer asks for something, the server gives it to them, each transaction unique.

Web REST APIs generally support simple HTTP(S) based communication. Information needed – the method, resource, parameters, and/or properties – are contained in the HTTP headers or the URL. It might look something like this:

```
https://yoursite.com/api/v6/data.html?action=get&region=6&year=2021
```

This URL indicates that we are accessing the `data` resource, using the `get` method, passing parameters for the `region` and `year` properties, which likely act as filters in this case. Other information, such as our authentication key, may be present in HTTP headers.

REAL-WORLD EXAMPLES

Here are several real-world examples of APIs that can be used to access data which might be relevant to SAS developers.

The U.S. Census

Available at: https://www.census.gov/data/developers/guidance/api-user-guide.Overview.html

The Census has several APIs available for accessing and using Census data. The primary APIs are the Census Data API, which is used to directly access data tables, the TIGERweb API, which is used to define the boundaries of the Census tracts, and the Geocoder API, which is a tool for identifying the exact coordinates of an address,

The Census Data API is an example of a public API; no authentication is required for the basic functionality,

An example of a Census Data API query from their documentation:

https://api.census.gov/data/2019/pep/charagegroups?get=NAME,POP&HISP=2&for=state:*

That query has several components:

- Object: /pep/charagegroups (identifies the Census table)
- Method: get (indicates the API should return datasets)
- Properties: 2019 (year), NAME, POP (variables to return), HISP=2 (a filter), and for=state:* (a filter)

Run the query (by clicking the link above, or typing it into your browser, or loading it in to language-of-your-choice via the appropriate HTTP request library) and you see the API returns a JSON file containing the data you requested – the population of non-hispanic residents in each state. ("Name" here is the name of the data item – in this case, the state or geography name).

Salesforce

Available at: https://developer.salesforce.com/docs/atlas.en-us.api_rest.meta/api_rest/

Not all APIs are public use APIs. One example – the example we will use in the rest of the paper – is that of Salesforce, a CRM (Customer Relationship Management) platform used by many businesses to store data about their customers or other people or entities they have relationships with.

The Salesforce API suite is full-featured and allows developers to retrieve data as well as modify it. The primary APIs that are commonly used are the REST API, the SOAP API, and the Bulk API (which is REST-based, but not truly REST since the server does have a state to keep track of).

The Bulk API is a straightforward way to obtain significant amounts of data; rather than needing to wait in an active connection for the server to send a large amount of data that might take minutes to complete, the API allows the user to submit a query and then wait for the query to complete – then download it in one large file. Whereas the REST API is like a fast food restaurant - you place the order and pick it up immediately, or nearly so – the Bulk API is like ordering ahead, then driving to the pick-up window. This allows the developer (and the server) to drop the connection, do other things, and periodically check back in to see if the query is done yet.

An example of a bulk query job, from the Salesforce documentation, might be:

1. Submit the query.
   `https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/query`
2. Check on the status of the query.
   `https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/query/7986gEXAMPLE4X2OPT`
3. Retrieve the results.
   `https://MyDomainName.my.salesforce.com/services/data/v55.0/jobs/query/7986gEXAMPLE4X2OPT/results/`

Note that the actual query is not visible in the URL – it is instead supplied in the headers, as is the authentication token, and most of the information is passed back in response headers (such as the query ID, which is returned in the response headers to the first call, and the status, which is returned in the second call). Storing these properties in the headers allows for more complex information to be passed.

IMPORTANT COMPONENTS OF APIS

While APIs have many methods, objects, and properties, depending on what they are intended for and how they are designed, there are a few methods that are common all data Web APIs.

Authentication

For some APIs, this is simple: establishing an HTTPS connection is all the authentication you need. For private APIs, however, typically one of a few methods of authentication is used. OAuth 2.0 is a common protocol, which requires sending a token (similar to a password) to the server to identify the application, and then returns another token (typically with a shorter lifespan) that the application can then use to identify itself in further calls.

Obtaining an access token for OAuth 2 often requires the user to enter their username and password in an interactive window, and then save the resulting access token in a file. That token should be treated as a password – it identifies the user, and gives a user with that token all of the abilities and permissions the user has.

These access tokens can have expiration dates, and typically can be revoked by the user at any time.

Metadata Discovery

The user needs to have a way of discovering information about the service. This can come in a variety of forms, whether it is finding out which version of the API is supported by the server, finding what tables are available or what variables are available on those tables, or finding out information about those tables and variables, such as table counts, variable types, etc.

Some of that information is needed to utilize the API at all; for example, many URLs include the API version as a component of the URL itself. Thus, the developer must find out what version is supported – and as that number can change over time, the application may need to discover it dynamically.

In the Salesforce API, for example, the URL is straightforward to discover the version number – simply the part of the URL before the version number (`/services/data/`) will return the version number, and adding to that URL the version number allows the user to retrieve what resources are available (`/services/data/v##.#/`).

Other useful pieces of metadata include information about limits (how many rows a user can download per time unit, for example), what objects are available (Salesforce's equivalent of tables), and what relationships exist between those objects.

Data Access

Web Data APIs must, of course, be able to return data to the user. There is typically one or more methods devoted to data access, often returning data in semi-structured forms such as JSON or XML. Simple data access often returns the data simply as the body of the HTTP request; this can be inefficient or risk errors if the amount of data is extremely large, however. APIs that expect to support larger amounts of data downloads will typically offer a Bulk API, like Salesforce does, which permits the user to submit a query, obtain the status of the query, and then retrieve the results.

YOUR API CHEAT CODE: PYTHON

WHY REINVENT THE WHEEL?

While all of the above can be done in any programming language (or even without programming at all), many APIs have pre-existing code written by users of the open source community – or the API developer themselves – that allows you to use their API by simply including those libraries and calling their methods. Perhaps the most common language used for this is Python – which benefits from being extremely popular, adept with data, and easy to understand. For SAS users, it also has an additional benefit: Python is very easily translated to SAS.

PUT TOGETHER THE BUILDING BLOCKS

If there is sample code on the website directly (for example, Quickbooks TSheets has an API reference at https://tsheetsteam.github.io/api_docs/ which includes inline code for Python and many other languages), you can simply paste that code into your Python editor and run it, perhaps with minor modifications to include authentication elements or other site-specific information. Typically, no knowledge of Python is needed to use these code snippets – the code is sufficiently clear that any user can run it and make simple modifications to it.

For many larger applications, there will be pre-built Python packages that include all of the code necessary for communicating with the API. Salesforce, for example, has the `simple_salesforce` package. A good explanation of how to use it is available at https://developer.salesforce.com/blogs/2021/09/how-to-automate-data-extraction-from-salesforce-using-python .

Once the package is installed, using `pip install simple_salesforce`, it is straightforward to connect to the API:

```
from simple_salesforce import Salesforce
import requests
import pandas as pd
from io import StringIO
sf = Salesforce(username='',password='', security_token='')
```

Then, the `sf` object allows you to access the data directly – whether through a pre-built report or through a SOQL query (similar to SQL, but specific for Salesforce).

```
results=sf.query_all("""
    Select
    CreatedDate,
    Listing__r.RecordTypeSnapshot__c,
    Name,
    Listing__r.ProviderName__c
    from UserInstall__c
    where CreatedDate=LAST_N_DAYS:7
    """)
```

Even without knowing any Python, it is very straightforward to modify these sample queries to fit your needs. An application to download reports could be built in literally minutes, without needing to have a deep understanding of the details of the API.

Other APIs may have similar packages available; check the documentation, as well as the Python Package Index (https://pypi.org/), to discover them.

TEST IT OUT – OR PUT IT IN PRODUCTION!

The sample code or packages can be used for testing queries for further development in SAS or other languages or can be used directly themselves. Even if the remainder of the codebase is SAS, accessing the API may well be simpler to keep in Python if the API documentation is also in Python.

However, if a pure SAS toolchain is desired, this can be utilized as a step in that process – by using the pre-written Python code to build the structure of the needed program, and then convert each piece to its equivalent in SAS.

CONVERTING TO SAS

For some, Python won't cut it for production – either because it's not a language that is fully supported by the team, or due to its open-source nature, or some other reason. If that applies in your location, consider the Python a steppingstone to SAS. Since Python and SAS operate in roughly the same manner – and for the API, it's all the same – it is straightforward to convert the calls to SAS.

SAS ESSENTIALS

The primary tool that a SAS programmer needs for calling APIs is PROC HTTP, which will submit the URL and parameters to the website.

PROC HTTP Syntax

Basic PROC HTTP syntax looks like this:

```
proc http
   url="https://your-url.com/api/resource?option=parameter"
      method='GET'
      headerin=inheader
      headerout=outheader
      out=output
     ;
run;
```

URL is the actual URL to call, which contains all of the parameters except those that are in the headers. This URL will exactly match what Python uses.

METHOD is usually "GET" for queries that only supply parameters in the header and/or the URL, or "PUT" for queries that push a data file to the server as part of the query. For most data access queries, "GET" is correct here. Python code should have an equivalent method parameter, or if not present it is likely "GET".

HEADERIN is a filename for a SAS file that stores information provided in the header of the query. This is often information like the authentication token or the content type expected or provided. These will be stored in Python in a data structure (often JSON, or something that is easily converted to JSON, such as a dictionary or list).

HEADEROUT is a filename for a SAS file that will contain the headers passed along with the results of the query. The most important information typically in this file is the result code – 200 is typically a successful query, while other codes such as 404 (not found), 401 (not authorized), or 400 (bad request) can tell you different issues with your query.

OUT is a filename for a SAS file that will contain the output requested. This may be very short or nonexistent in some requests if the entire response is contained in the headers.

JSON or XML?

Many responses are returned in a structured format, such as JSON and XML. SAS can read these fairly easily, so long as they are well formed. Both JSON and XML are read by using a LIBNAME with the same name as the filename earlier. In the case of XML, there are two choices for libname engine, XML and XMLV2; the latter is newer and more capable. The syntax is nearly identical, as below:

```
libname output json;

libname output xmlv2;
```

In either case, the name of the libname matches the name of the filename used in the PROC HTTP statement. Both will attempt to parse the file and create one or more datasets; sometimes further parsing is necessary, but often the automatic datasets have the information needed. In the case of JSON, the ALLDATA dataset, which contains all of the JSON data in a single dataset, is often sufficient.

The choice between JSON or XML is mostly a matter of personal preference, although some APIs will only return one or the other. Typically, when there is a choice, the ACCEPT (for returned data) or CONTENT-TYPE (for sent data) header is used to specify which is preferred.

ONE PIECE AT A TIME

When converting the code, take the Python code one piece at a time, and convert to SAS. Ultimately, the goal is to produce the exact same call to the API (the same URL parameter and the same header); working on this one piece at a time makes it easy to compare the output of each version. Some notes follow to help with specific issues at each step.

Authentication

Authentication typically requires passing a token and receiving a bearer token or other identifier, and typically involves only the headers. Specifically trap HTTP response 401 here (Unauthorized), as that indicates the login is invalid in some way.

Metadata Discovery

Typically, this is used to drive further work and to convert to SAS look towards data-driven programming techniques in SAS, such as using PROC SQL SELECT INTO to create a list of values from a column in a dataset.

Data Access

Here is where the most care should be made as to the data type – SAS and Python may not have equal capabilities to parse JSON or XML. SAS is most comfortable with CSV but can handle both JSON and XML; test both if they are options to see which SAS is able to automatically parse better. Python often defaults to JSON, which may require extra work to parse in SAS if the format is not perfectly suited to SAS's parser (or if it is a format less familiar to the user).

If bulk data access is required, most of the action will take place in headers until the final file is downloaded. When it is downloaded, if it is provided as a ZIP file, SAS can read that directly without using an unzip utility. Look at the FILENAME Statement: ZIP Access Method documentation for more information.

CONCLUSION

Accessing data using Web APIs can seem like a daunting task. By taking advantage of the ease of use of Python, users can reduce the difficulty level of accessing data and cut down on development time, while still producing SAS code at the end.

ACKNOWLEDGMENTS

The author would like to thank Troy Hughes and the board of WUSS for inviting him to speak at the conference, and Jeff Vose and David Trevarthen for providing the encouragement and support for the author and the entire Data Services team to learn and use different technologies, including this one.

CONTACT INFORMATION

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

Joe Matise
NORC at the University of Chicago
matise.joe@gmail.com
https://github.com/snoopy369/presentations