SESUG Paper 103-2025

Give it a ReST! Working with APIs in SAS

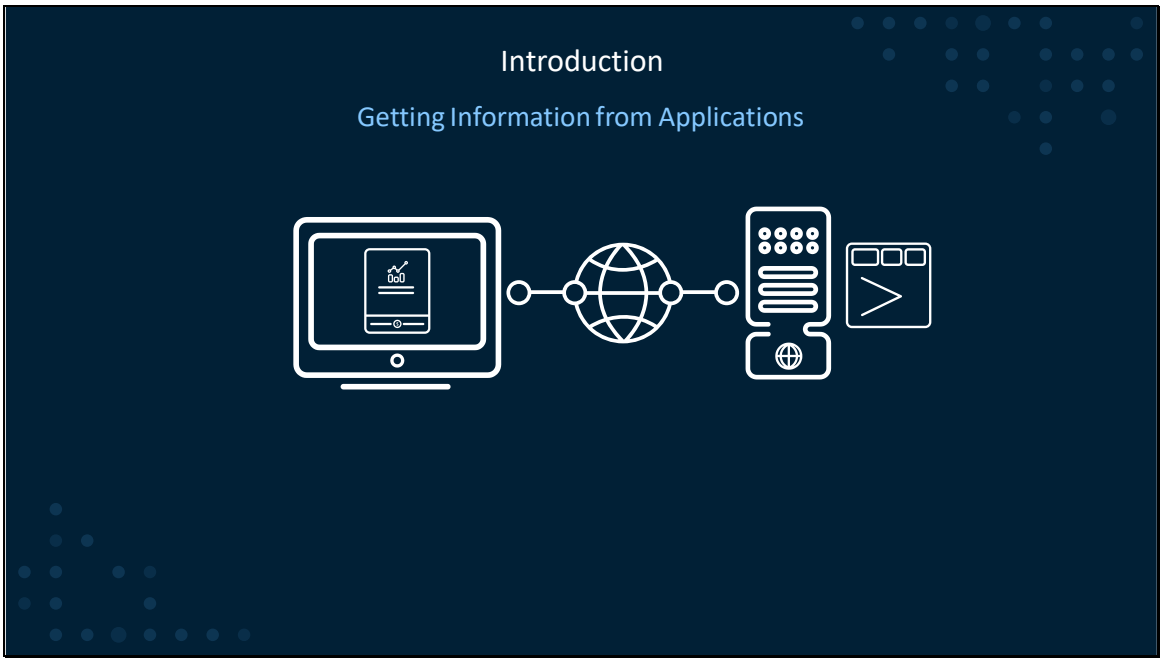
Mark Jordan, SAS Jedi (Retired)

ABSTRACT

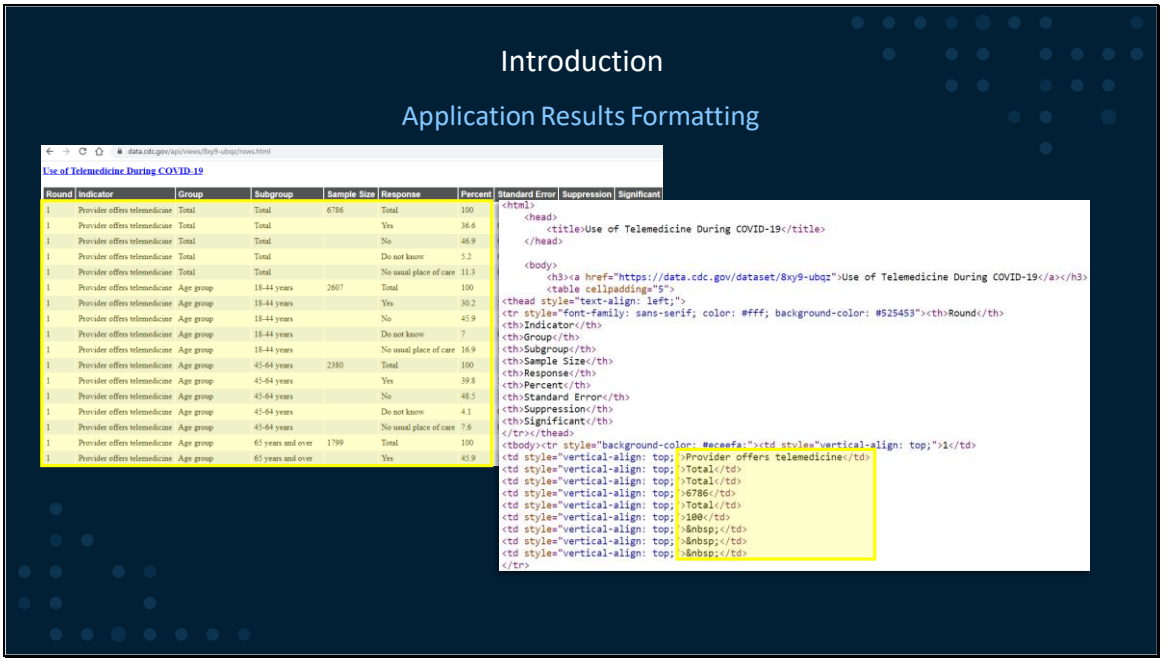
Application programming interfaces, or APIs, provide a way for diverse computer programs and systems to reliably communicate with each other. Representative State Transfer (ReST) APIs are the most common type of API used for communication via hypertext transfer protocol (HTTP) . Base SAS has robust tools baked in for working with ReST APIs. This paper includes a brief introduction to API concepts and demonstrates:

- Using PROC HTTP to submit API requests
- Using the JSON and XMLV2 LIBNAME engines to work with API response results
- Working with the World Bank API to obtain data in real-time

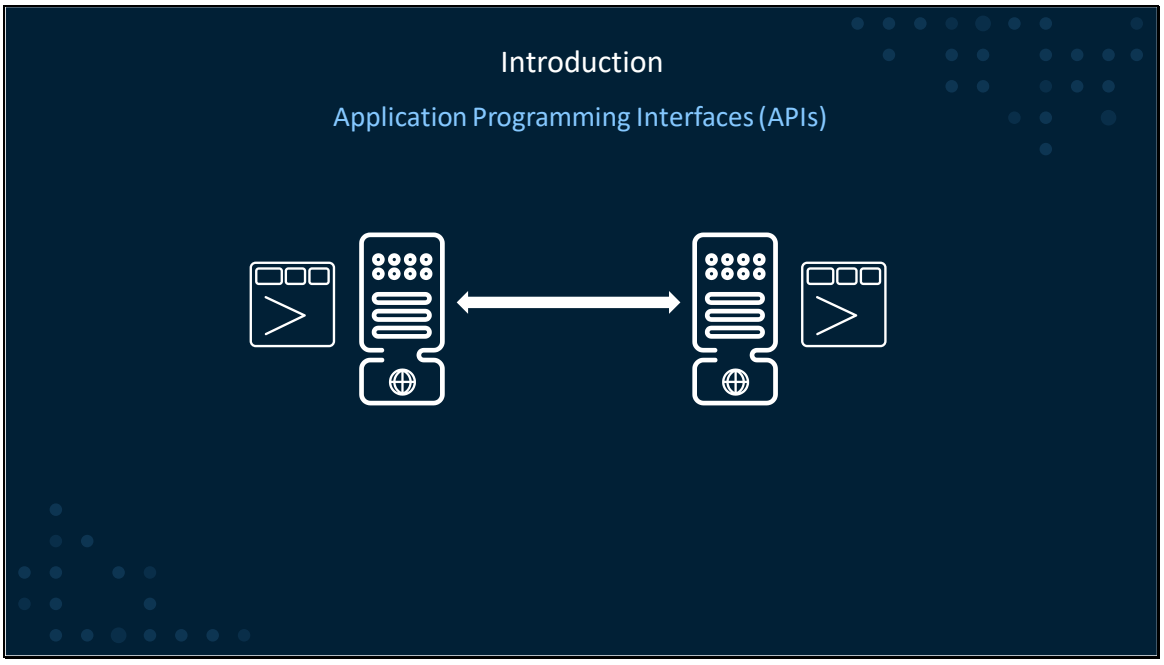
INTRODUCTION



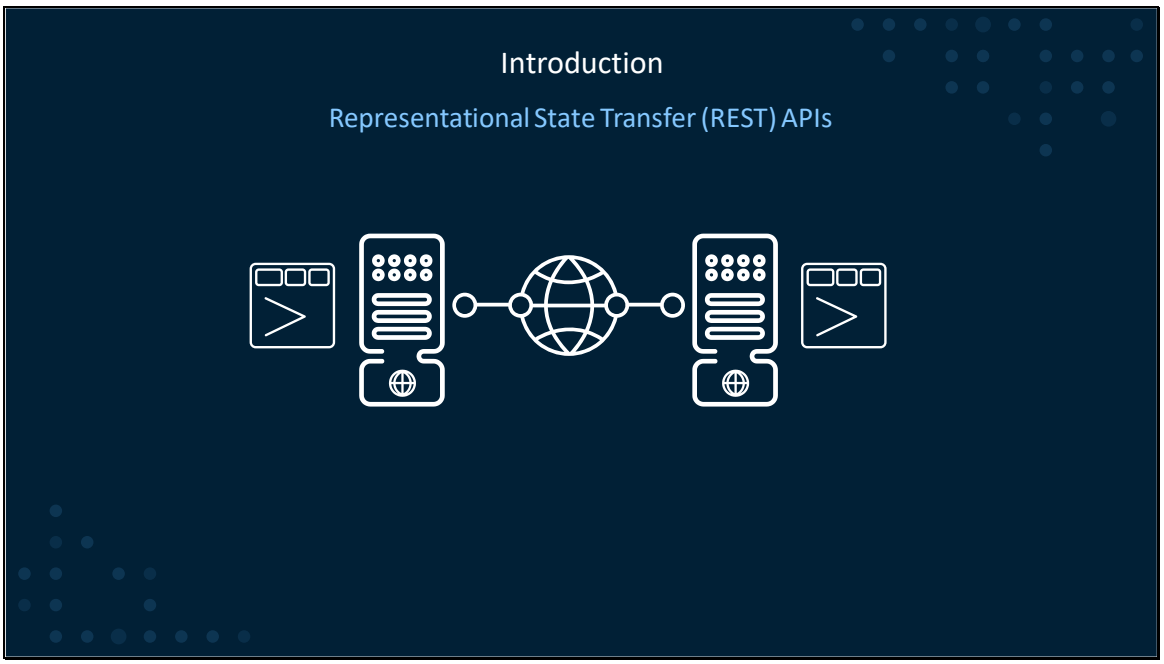
We're used to using powerful applications every day. We don't even give it much thought as we connect to a server somewhere "out there" to get the information we need to do our work on our local computer. Very frequently, that information comes in the form of an HTML page.



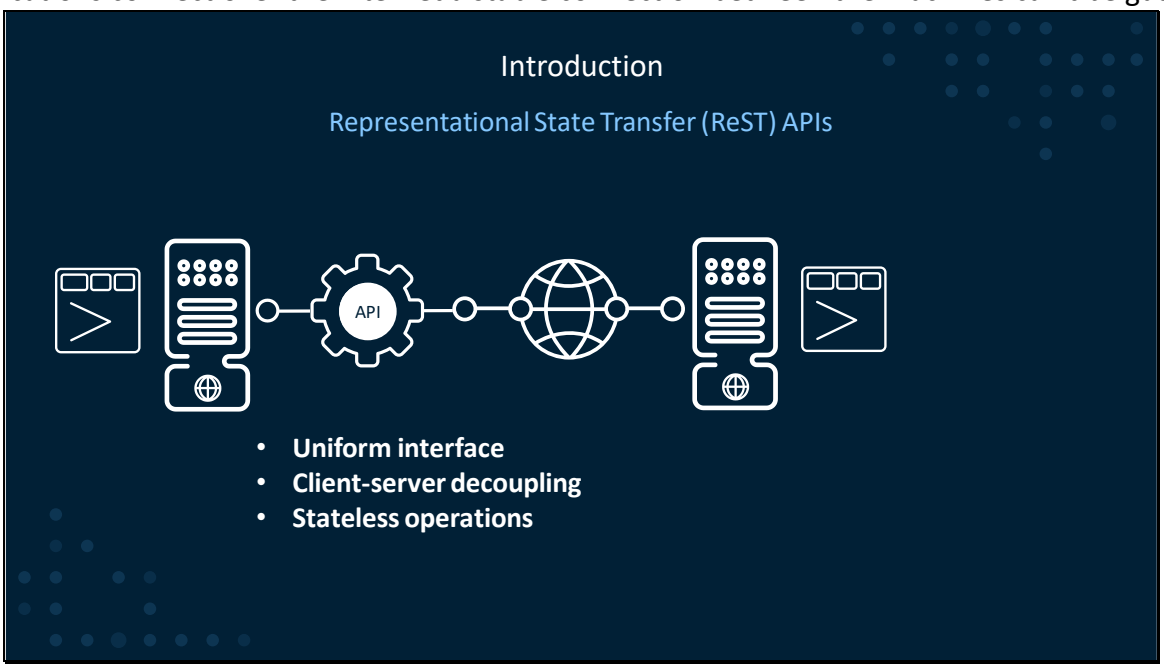
HTML is a beautiful format for presenting information for human consumption, but if you want the information for programmatic processing, HTML is not a machine-friendly format.
See: https://blogs.sas.com/content/sasdummy/2017/12/04/scrape-web-page-data



Application Programming Interfaces, or APIs, come in a variety of styles and architectures. APIs enable applications to connect, communicate and transfer data in standardized, machine-friendly formats.

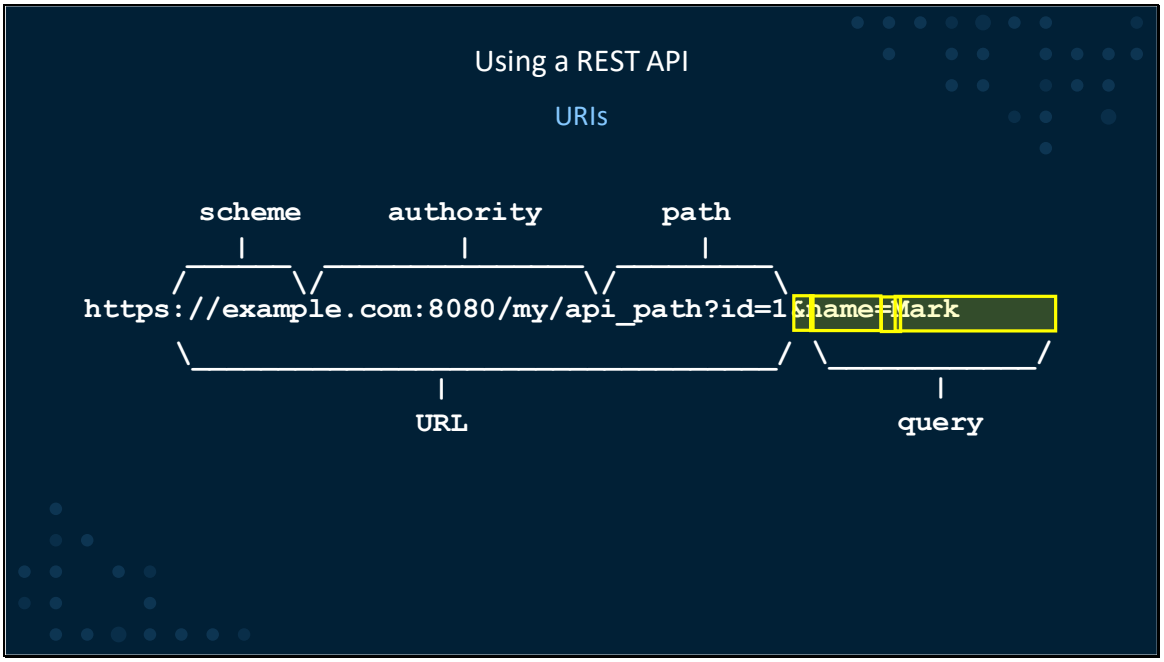


When applications connect over the internet a stable connection between the machines can't be guaranteed.



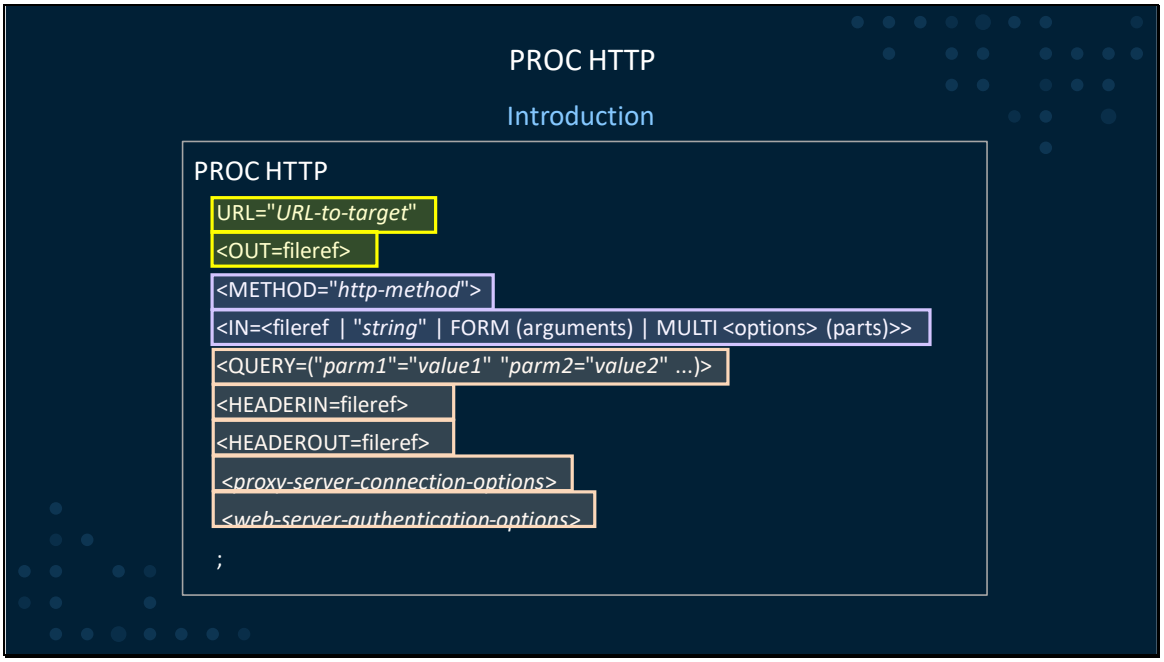
ReST API architecture is designed to facilitate operating under these conditions. A ReST API provides these fundamental properties:
1. A uniform interface.
2. Client-server decoupling.
3. Stateless operations.

USING REST APIS

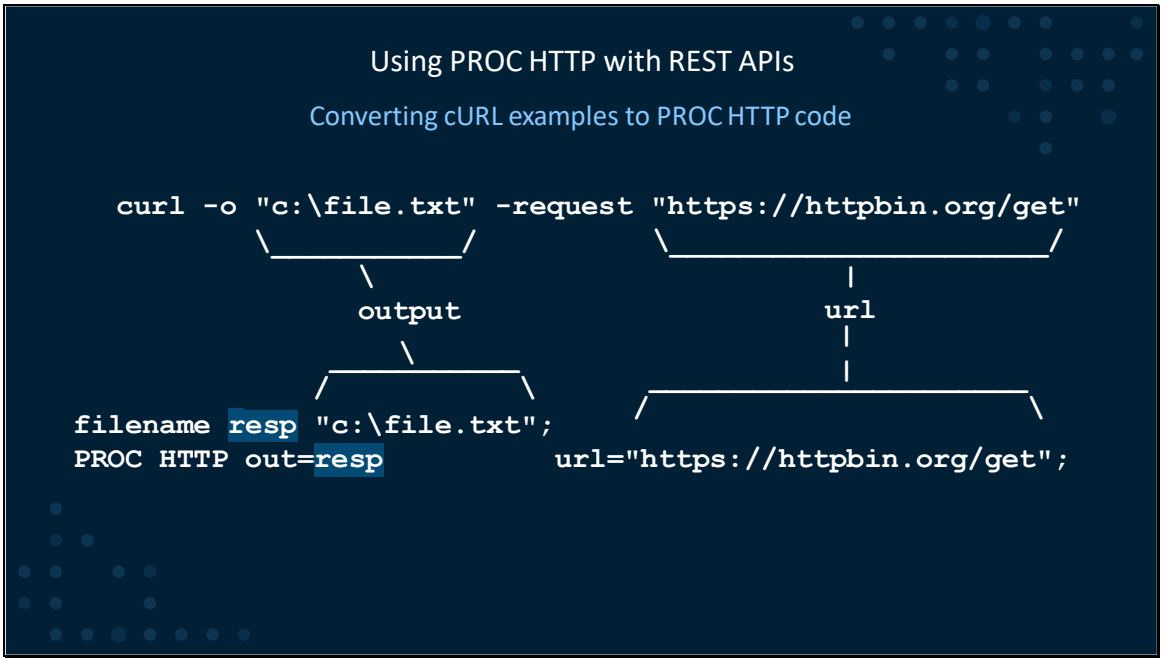


A URI scheme, authority and path are collectively referred to as the URL. Any query is separated from the URL by a question mark with the parameters expressed as ampersand-delimited name-value pairs.

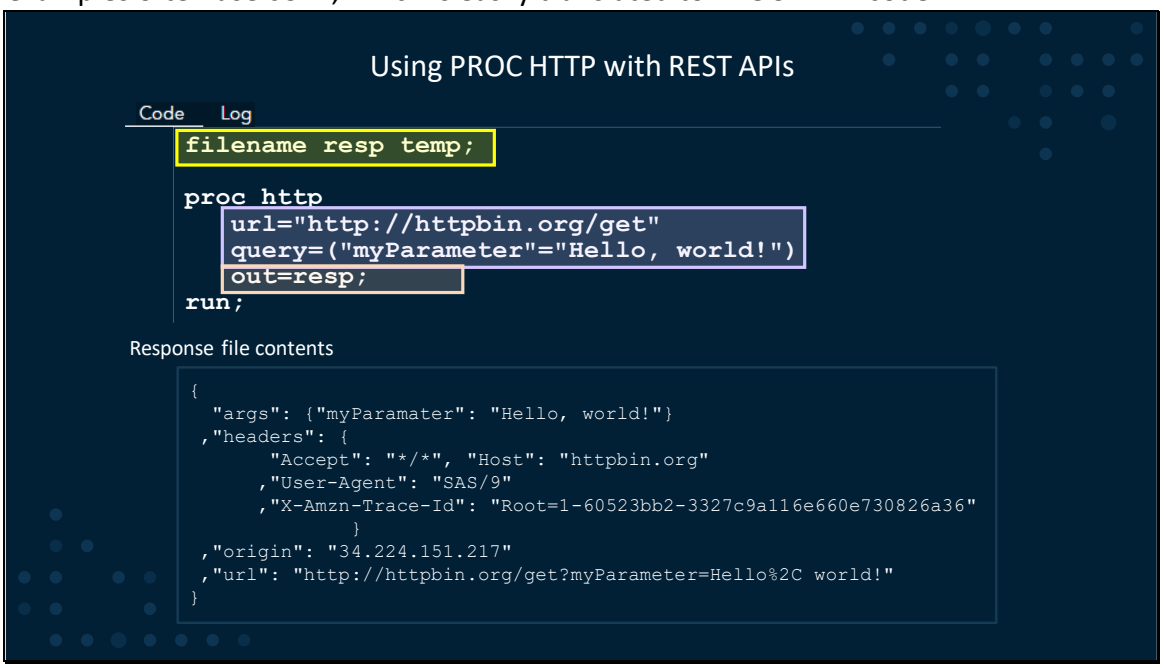
PROC HTTP



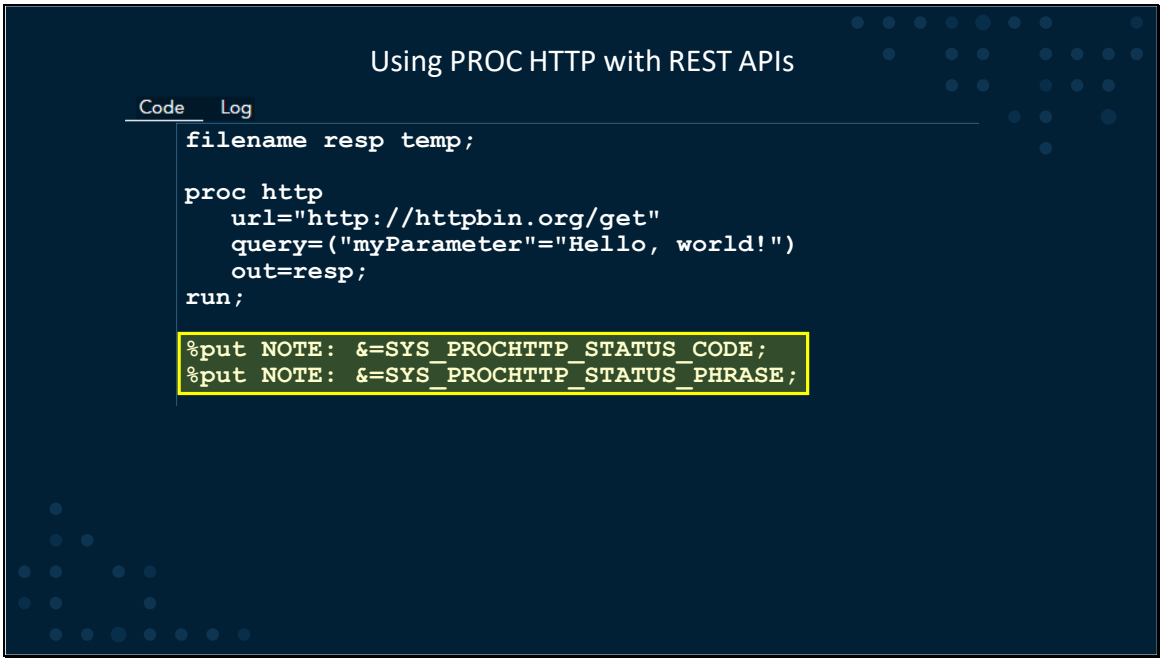
With PROC HTTP, the URL= and OUT= parameters specify where to retrieve data from and save to. The OUT= parameter requires a fileref – a literal path and file name will produce an error. You can use the METHOD= parameter to specify the method for the API call. The IN= parameter is used when posting data. Other parameters can be used to express the query as key/value pairs, define the API call request and response headers, and authenticate to the server.



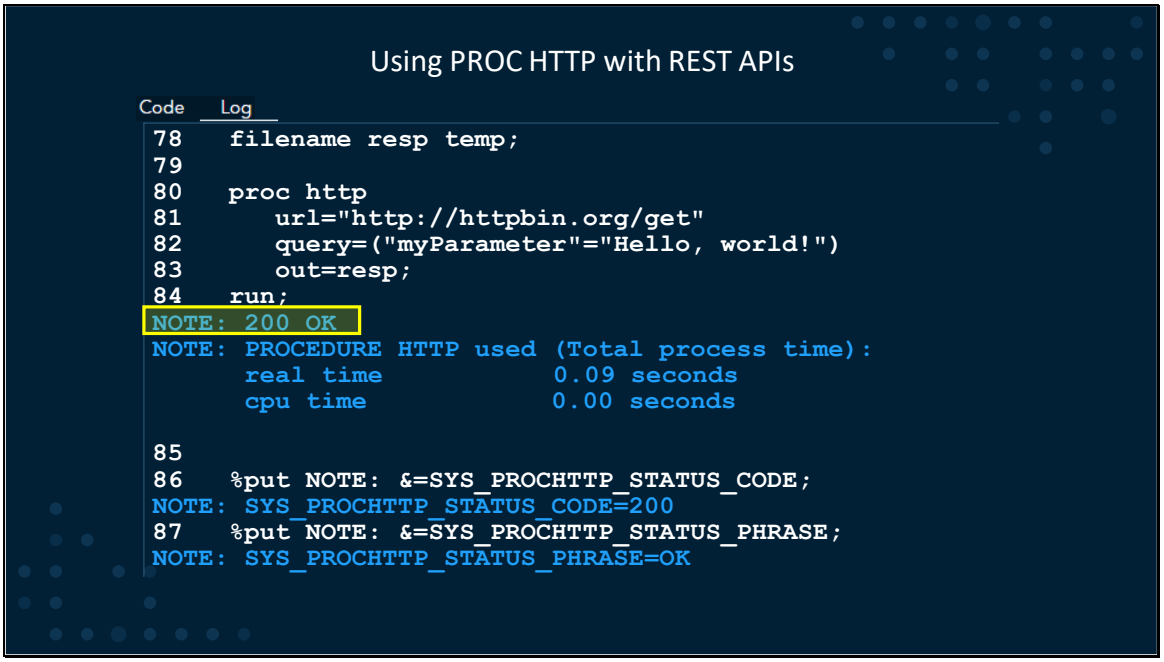
APIs usage examples often use cURL, which is easily translated to PROC HTTP code.



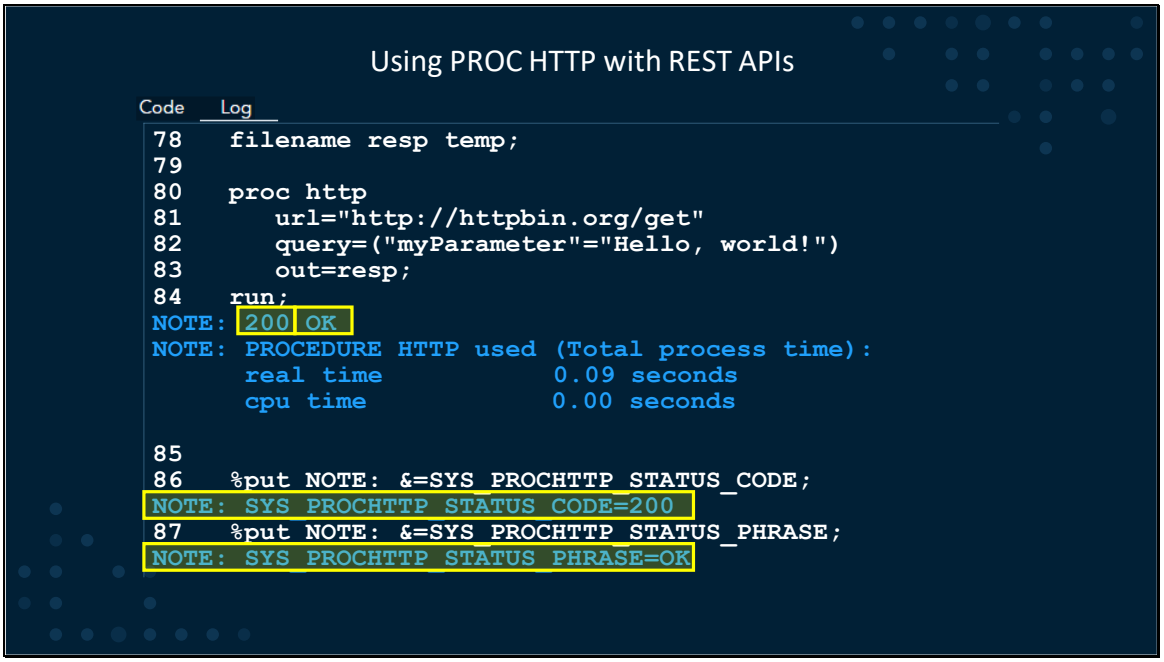
You can use a volatile file for the response file by specifying the TEMP keyword on the FILENAME statement. The response from the API call is written to the volatile file resp. The file content can be accessed until the SAS session ends or the fileref is cleared, at which point it is automatically deleted.



An API provides status in the response header, using codes defined in section 10 of RFC 2616. PROC HTTP automatically sets the values of two macro variables, SYS_PROCHTTP_STATUS_CODE and SYS_PROCHTTP_STATUS_PHRASE.



PROC HTTP also automatically writes the status to the SAS log.



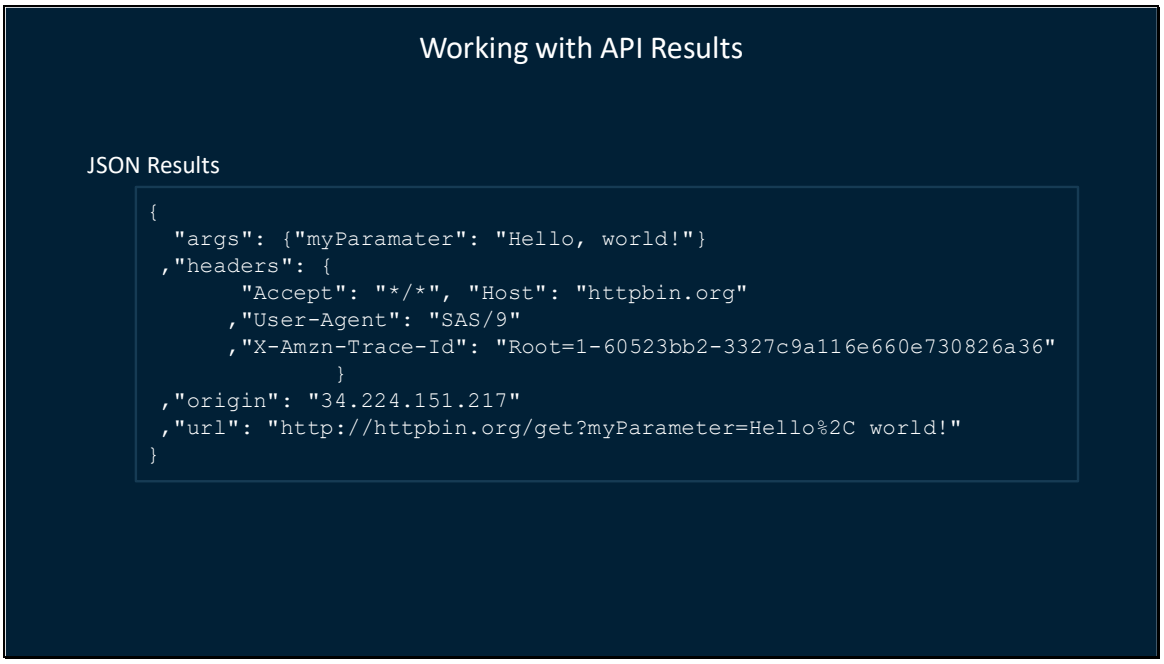
The SYS_PROCHTTP_STATUS_CODE macro variable contains the numeric status code and SYS_PROCHTTP_STATUS_PHRASE contains the text of the status returned.

DEMO: API FUNDAMENTALS WITH PROC HTTP

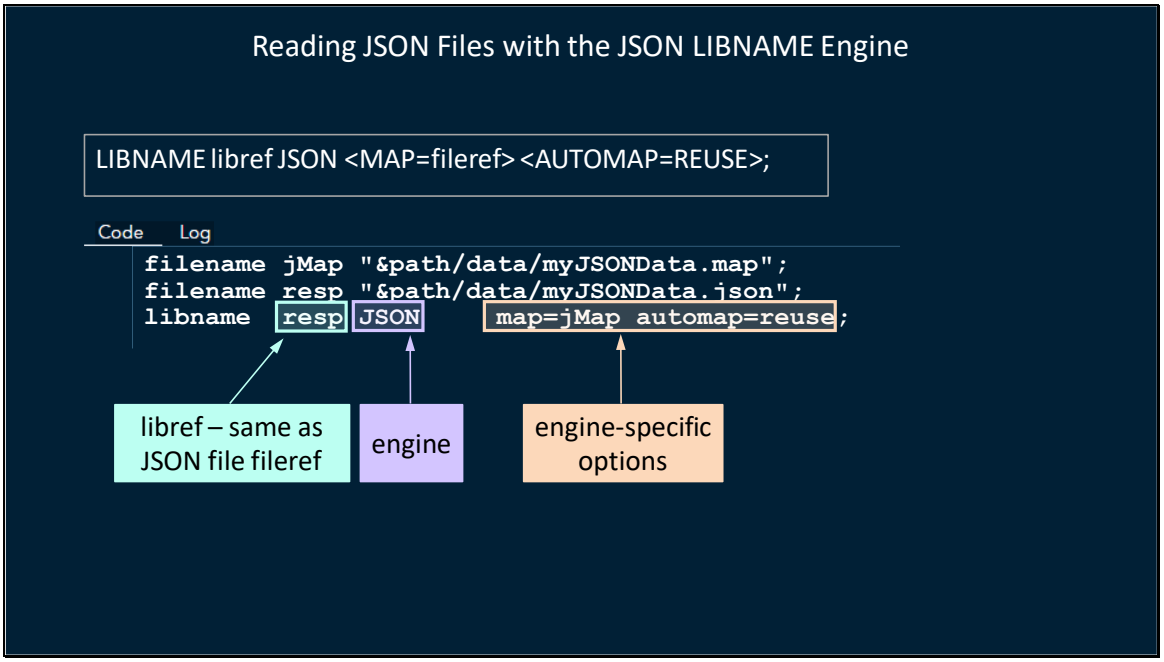
```
/* Set up volatile file filerefs for response, header and input */
filename resp TEMP;
filename headout TEMP;
filename input TEMP;

/* Add content to the input file */
data _null_;
   file input ;
   put "this is some sample text";
run;

/* POST the input file contents to the API */
proc http
   url="http://httpbin.org/post"
   in=input
   out=resp
/*    method=post */
   headerout=headout;
run;


%put NOTE: &=SYS_PROCHTTP_STATUS_CODE;
%put NOTE: &=SYS_PROCHTTP_STATUS_PHRASE;
/*** Review the log - note status message */
/* Review macro variable values associated with PROC HTTP */

/* Review the header from the API response - note status codes in header */
title "Header Output";
data _null_;
  file print;
  infile headout;
  input;
  put _infile_;
run;
title;

/* Review the body of the API response in the Log */
data _null_;
    rc=jsonpp('resp','log');
run;


/* Generic GET to retrieve data from the API */
proc http
   url="http://httpbin.org/get"
   out=resp;
run;

/* Review the body of the API response in the Log */
data _null_;
    rc=jsonpp('resp','log');
run;

/* Macro variable values */
%put NOTE: &=SYS_PROCHTTP_STATUS_CODE;
%put NOTE: &=SYS_PROCHTTP_STATUS_PHRASE;

/* Bad Request */
proc http
```

```
    url="http://httpbin.org/bad_request"
    out=resp;
run;

/* Review the body of the API response in the Log */
data _null_;
    rc=jsonpp('resp','log');
run;

/* Macro variable values */
%put NOTE: &=SYS_PROCHTTP_STATUS_CODE;
%put NOTE: &=SYS_PROCHTTP_STATUS_PHRASE;
```

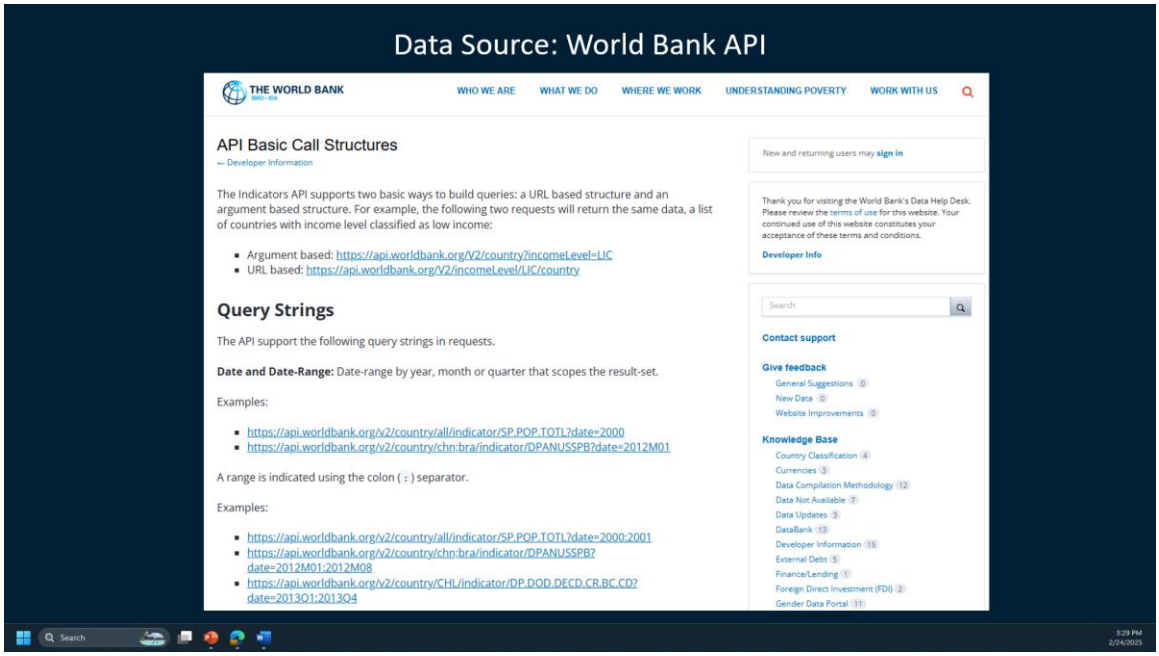
WORKING WITH JSON RESULTS



APIs frequently give you a choice of result formats. JavaScript Object Notation, or JSON, is a very lightweight, structured text format with data expressed as key/value pairs, returned in either UTF-8 or some other UTF encoding. JSON can express complex data objects, like dictionaries and arrays, and is well suited for machine reading.



Base SAS includes the JSON LIBNAME engine. If your JSON is well-formed and not overly complex in structure, the engine is capable of automatically producing the map required to read the JSON file. In this example, the fileref jMap references a volatile file we'll use for the map, and the fileref resp references a volatile file containing the JSON data received from a previous PROC HTTP step. We read the data using a LIBNAME JSON statement. Because the libref matches the JSON fileref, we don't need to specify the JSON file location. To retain a copy of the JSON map file, use the MAP= and AUTOMAP= options to specify a permanent storage location for the map file and to automatically generate the map. AUTOMAP=REUSE overwrites the map file if it exists.



Let's build this report.

2020 population and exchange rate
For countries with populations > 100,000,000

Code	Country	Exchange Rate, USD	Population
USA	United States	1	331,526,933
BRA	Brazil	4.150609	208,660,842
CHN	China	6.931297	1,411,100,000
EGY	Egypt Arab Rep.	15.89869	109,315,124
MEX	Mexico	18.81759	126,799,054
ETH	Ethiopia	32.02918	118,917,671
PHL	Philippines	50.83247	112,081,264
RUS	Russian Federation	62.07638	145,245,148
IND	India	71.31127	1,402,617,695
BGD	Bangladesh	84.87826	166,298,024
JPN	Japan	109.2672	126,261,000
PAK	Pakistan	154.7134	235,001,746
NGA	Nigeria	362.3992	213,996,181
IDN	Indonesia	13730.36	274,814,866

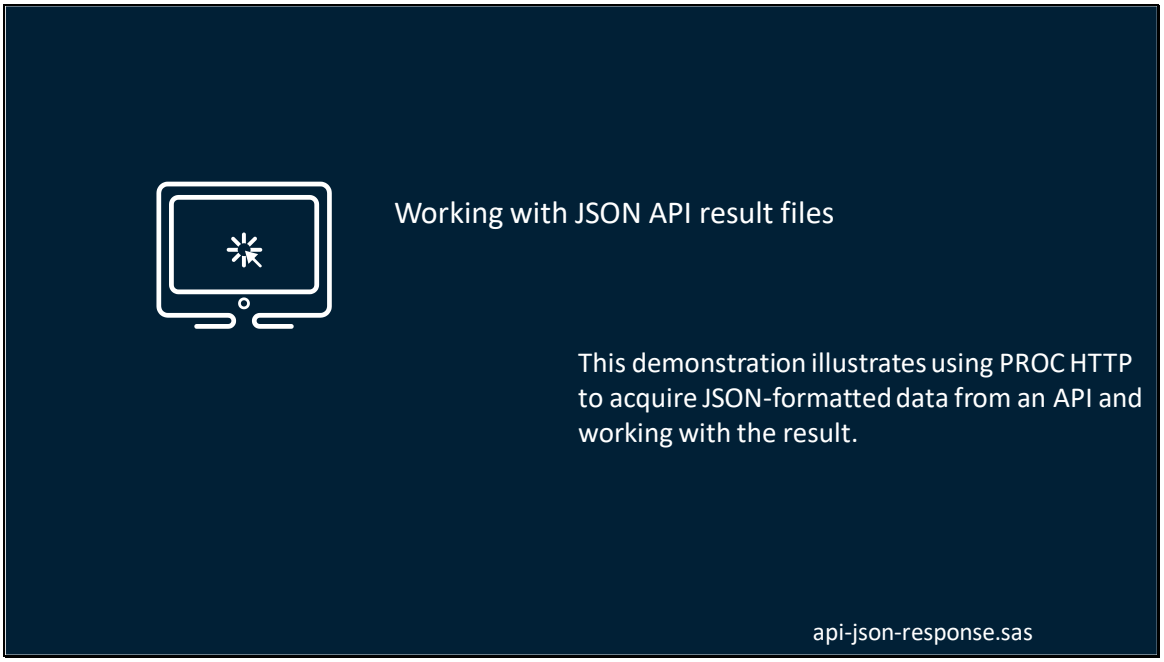
We'll use data provided by the World Bank, which has a robust and well-documented API available.

DEMO: WORKING WITH JSON API RESULT FILES

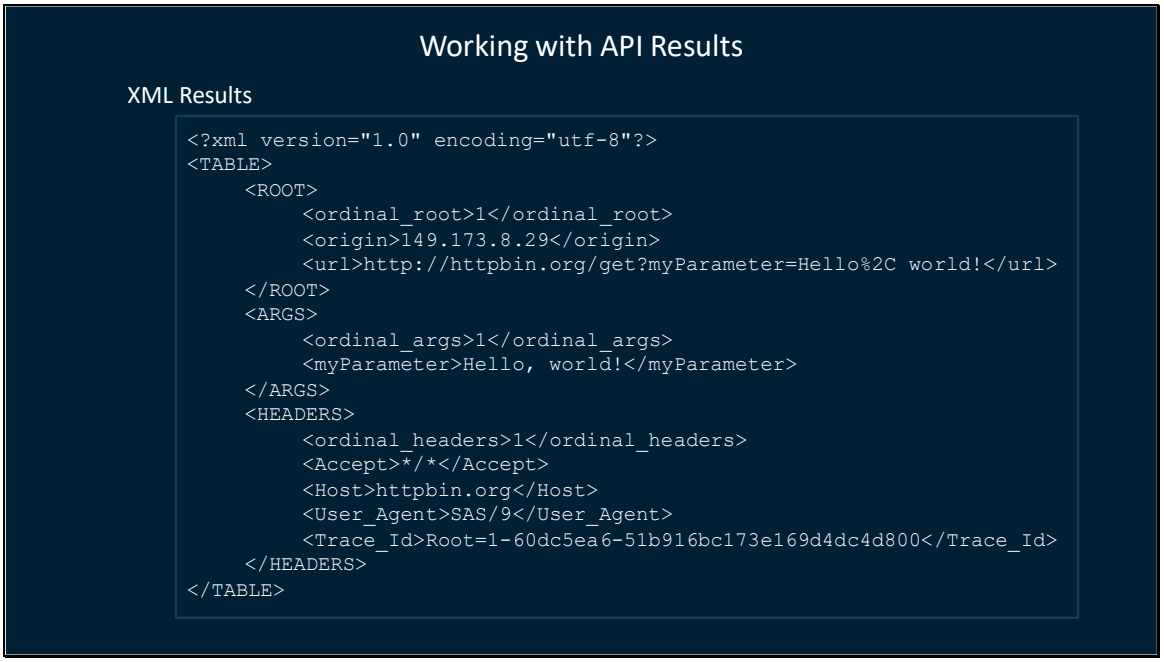
```
/* Make an API call that produces JSON results */
filename resp "&path/output/worldbank_pop.json";
proc http
url='https://api.worldbank.org/v2/country/all/indicator/SP.POP.TOTL?date=2020&format=js
on&per_page=5000'
   out=resp;
run;

/*****************************************************************************
 Alternate way of writing the PROC HTTP step using QUERY:

proc http url="http://api.worldbank.org/v2/country/all/indicator/SP.POP.TOTL"
   query=('per_page'='5000'
          'format'='JSON'
          "date"="2020")
   out=resp;
run;

*****************************************************************************/
/*** Open the JSON file as text - note it's all one line. ***/

/* Print the response file to the log for easier viewing */
data _null_;
    rc=jsonpp('resp','log');
run;


/* Explore the JSON library */
libname resp json;
proc contents data=resp._all_;
```

```
run;

proc print data=resp.root (obs=5);
run;

/* Extract the desired data from the JSON & add country name from SAS data set */
proc sql;
create table api.country_pop_2020 as
select Country
      ,2020 as Year
      ,Code
      ,Value format=comma16. as Population
    from api.countries as c
       inner join
         resp.root as r
    on r.CountryISO3Code=c.code
    order by country
;
reset outobs=15;
select *
   from api.country_pop_2020
    order by Population desc
;
quit;

/* Don't need the resp fileref or libref anymore  */
libname resp;
filename resp;

/* Construct a data-driven API call */
/* Get a semicolon-delimited list of high population country codes (>100,000,000)*/
proc sql noprint;
select code into :codes separated by ';'
   from api.country_pop_2020
   where Population > 100000000
    order by Population desc
;
quit;

%put NOTE codes=%superq(codes);

/* Just for demo: The list can be used to construct a data-driven API call to get
exchange rates */
data _null_;

url=%tslit(https://api.worldbank.org/v2/country/&codes/indicator/DPANUSSPB?date=2020M01
%nrstr(&)format=json);
   call symputx('URL',url);
run;

%put NOTE: codes=%superq(codes);
%put NOTE- URL=%superq(url);


/* Make the API call - get results as JSON */
filename resp "&path/output/worldbank_econind.json";
proc http
url=%tslit(https://api.worldbank.org/v2/country/&codes/indicator/DPANUSSPB?date=2020M01
%nrstr(&)format=json)
   out=resp;
run;
```

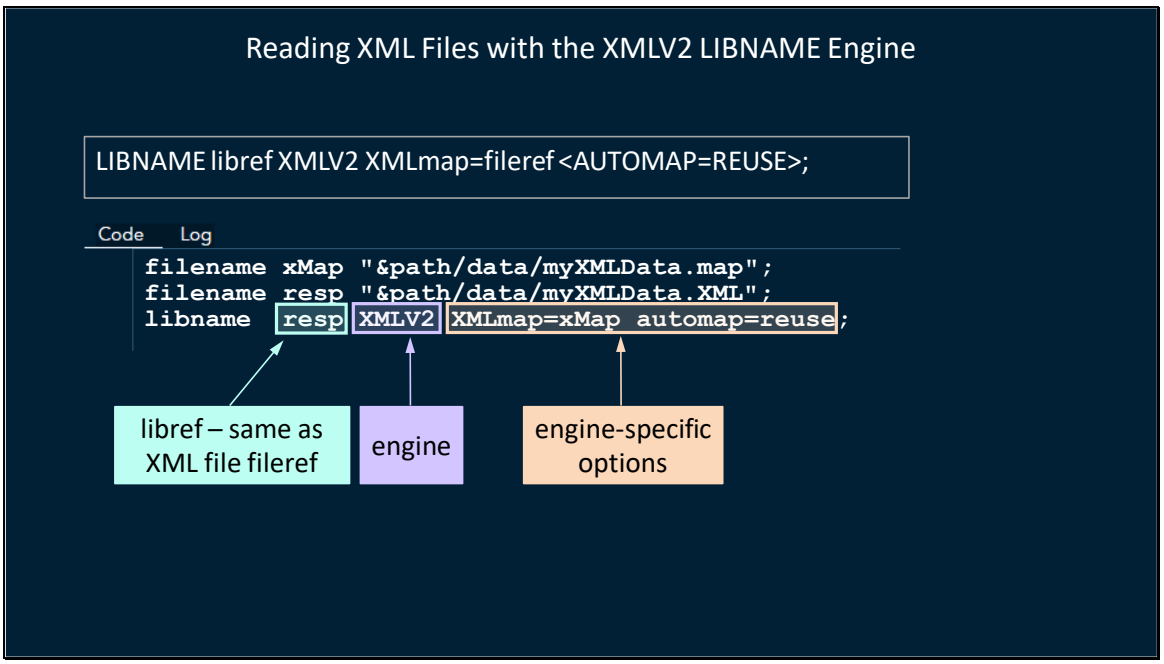
```
/* Explore the JSON Library */
libname resp json;
proc contents data=resp._all_;
run;

title "ROOT data sample";
proc print data=resp.root(obs=5);
run;
title "COUNTRY data sample";
proc print data=resp.country(obs=5);
run;
title;

/* Extract the exchange rate data from the JSON */
proc sql;
create table api.high_pop_xchng_2020 as
select c.ID as Code
      ,c.value as Country
      ,Date
      ,r.Value as XR 'Exchange Rate, USD'
   from
     resp.country as c
     inner join
     resp.root as r
   on c.ordinal_root=r.ordinal_root
;
select *
   from api.High_pop_xchng_2020
    order by xr desc
;
quit;

/* Don't need the resp fileref or libref anymore  */
libname resp;
filename resp;

/* Produce the final report with the data from the API calls. */
proc sql;
title   "2020 population and exchange rate";
title2  "For countries with populations > 100,000,000";
select e.*, Population
   from api.high_pop_xchng_2020 (drop=date) as e
        inner join
        api.country_pop_2020 as p
   on e.Code=p.code
    order by XR
;
quit;
title;
```



WORKING WITH XML RESULTS



Another common response format is eXtensible Markup Language (XML). XML was intended to define documents, so it can't express complex data objects like arrays or dictionaries. It is more verbose, but much more flexible than JSON, and can handle many different encodings. Because it is so verbose, XML can be a bit harder for humans to read, but it's well suited for machine reading.



The Base SAS XMLV2 LIBNAME engine makes it easy to work with XML data. If the XML is well-formed and not too complex, this engine is also capable of automatically producing the required map file. In this example, the fileref xMap references a permanent file location for the map file, and the fileref resp references a permanent file location for the XML data file. The LIBNAME XMLV2 statement allows us to read the XML data. An again, because the libref matches the fileref, we don't need to specify where to find the XML file. The XMLMAP= option specifies the location for the permanent map file. Setting AUTOMAP=REUSE overwrites the map file if it exists.

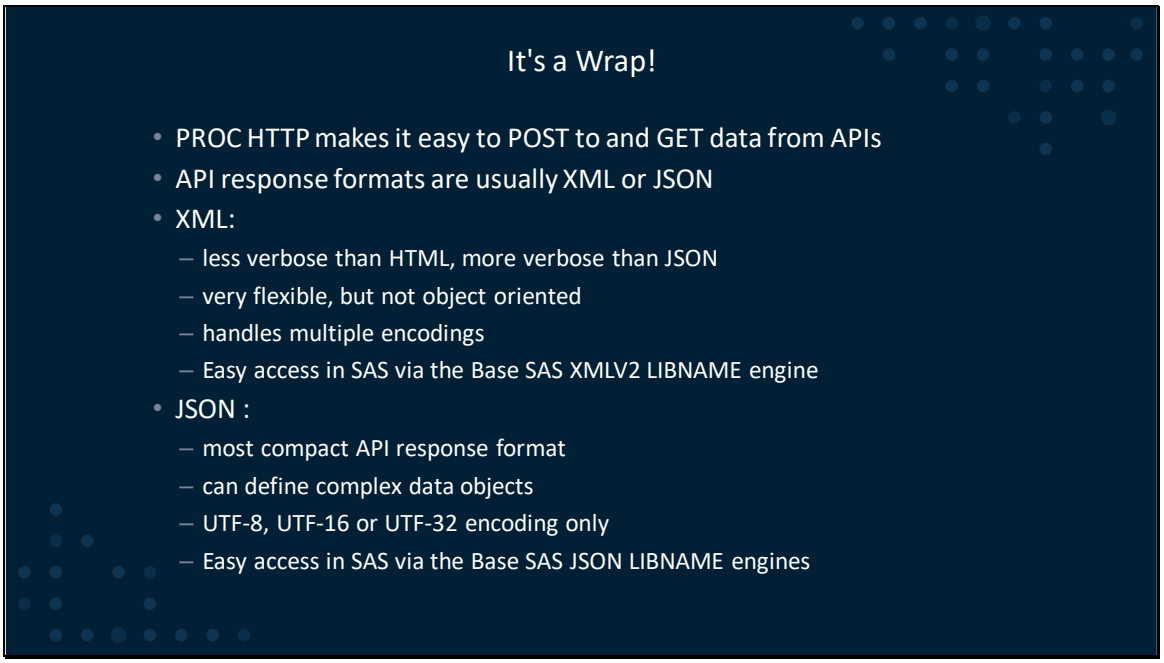
DEMO: WORKING WITH XML API RESULT FILES

```
/* Make an API call that produces XML results */
filename xMAP "&path/output/worldbank_pop_xml.map";
filename resp "&path/output/worldbank_pop.xml";
proc http
url='https://api.worldbank.org/v2/country/all/indicator/SP.POP.TOTL?date=2020&format=xml&per_page=5000'
   out=resp;
run;

/*** Open the XML file - note it's well formatted and syntax highlighted in the
browser. ***/

/* Explore the XML library */
libname resp XMLV2 xmlmap=xmap automap=reuse;
/* View table properties and variables in RESP Library window */
/* Note that DATA1 has the information we want, but value is character, not numeric */
title "resp.data1 sample";
proc print data=resp.data1(obs=5);
run;
title "resp.country sample";
proc print data=resp.country(obs=5);
run;

/* Extract the desired data from the XML & add country name from SAS data set */
```

```
proc sql;
create table api.country_pop_2020_xml as
select c.Country
      ,date as Year
      ,CountryISO3Code as Code
      ,input(Value,32.) format=comma16. as Population
    from api.countries as c
       inner join
         resp.data1 as r
    on r.CountryISO3Code=c.code
    order by country
;
title "Join results sample";
select *
   from api.country_pop_2020_xml (obs=5)
;
quit;
title;

/* Don't need the resp fileref or libref anymore  */
libname resp;
filename resp;

/* Could re-run the SQL to make this data-driven, but the macro variable already
exists*/
%put NOTE: codes=%superq(codes);
/* NOTE: codes=BGD;BRA;CHN;IND;IDN;JPN;MEX;NGA;PAK;RUS;USA */

/* Make the API call - get results as JSON */
filename xMAP "&path/output/worldbank_econind_xml.map";
filename resp "&path/output/worldbank_econind.xml";
proc http
url=%tslit(https://api.worldbank.org/v2/country/&codes/indicator/DPANUSSPB?date=2020M01
%nrstr(&)format=XML)
   out=resp;
run;

/* Explore the XML library */
libname resp XMLV2 xmlmap=xmap automap=reuse;
/* View table properties and variables in RESP Library window */
/* Note that
   DATA1 has exchange rate and date, but no country code. Date data is text.
   COUNTRY country_id has country codes
   Both tables have data1_ordinal so we can join
*/

/* Extract the exchange rate data from the XML */
proc sql;
create table api.exchange_rates_202001_xml as
select rc.country_id as Code
      ,c.Country
      ,input(substr(rd.date,1,4),4.) as Date
      ,rd.Value as XR 'Exchange Rate, USD'
   from
     resp.data1 as rd
     inner join
     resp.country as rc
     on rd.data1_ordinal=rc.data1_ordinal
     inner join
     api.countries as c
     on rc.country_id=c.code
```

```
;
title "Final result table";
select *
   from api.exchange_rates_202001_xml
;
quit;

/* We clear the libname and libref, the permanent XML and MAP files remain */
libname resp;
filename resp;

/* Produce the final report with the data from the API calls. */
proc sql;
title   "2020 population and exchange rate";
title2  "For countries with populations > 100,000,000";
select e.*, Population
   from api.exchange_rates_202001_xml (drop=date) as e
        inner join
        api.country_pop_2020_xml as p
   on e.Code=p.code
   order by XR
;
quit;
title;
```

CONCLUSIONS



RESOURCES

Download a ZIP file containing a PDF of this paper and the code used to create the presentation at https://bit.ly/GiveItaRest. The **demos** code folder contains two bonus programs: one demonstrates using PROC HTTP to download SAS code from GitHub, and the other demonstrates using PROC HTTP and the SAS Viya API to download a Visual Analytics report as a PDF file.

CONTACT INFORMATION

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

Mark Jordan
Email: sas.jedi@gmail.com
X: @SASJedi