Paper CT04

Data from the Cloud ... and back again!

John McDade, PHASTAR, Glasgow, United Kingdom

ABSTRACT
Cloud data storage is ever increasing, allowing users to access files from different devices and work collaboratively when required. This paper looks at different ways of accessing that data from SAS, with authentication when needed. Cloud data can be downloaded, modified at source or new data uploaded. Often programming with cloud-based data is reserved for application developers but we will look at some simple methods of connecting to the cloud within a clinical trial setting. The paper will look at downloading controlled terminology versions from NCI website and the CDISC Library and also reading and writing data to Microsoft SharePoint®.

INTRODUCTION
As technology evolves, more data is living in cloud storage and the ability to access and transform the data is key to efficient processing. From receiving patient data from mobile devices, accessing industry standards, study teams collaborating on online documents, a host of activity depends on online data.

CLOUD DATA
Cloud data and cloud computing is something we rely on daily but there are misconceptions about what this is. Despite the name conjuring up images of data freely floating around, all cloud data is physically stored at a finite location. In practice the big players in Cloud Storage will have data centres across the world comprising huge servers to best meet customer demands.

Where the data is stored exactly is less important, the key concept is the accessibility of the data. When we think of Google Drive, OneDrive, Gmail etc. these can be accessed from multiple devices provided we have an internet connection and appropriate login credentials. Similar can be applied programmatically using SAS with PROC HTTP and the relevant security information.

Programmers are familiar with different methods of importing files, normally these can be referenced with a physical file path visible to the SAS session without much complexity:

```
*--- simple proc import;
proc import datafile="P:\Study_A\Data\Raw\conmed_flags.xlsx"
  out=conmeds
  dbms=xlsx
  replace;
  sheet="sheet1";
  getnames=yes;
run;
```

Accessing files on a remote server can be anywhere between trivial and challenging depending on the security involved. The examples in the following sections will take us through them.

PROC HTTP
For all the example code in this paper, we will be using PROC HTTP [1]. This is a SAS procedure that issues Hypertext Transfer Protocol (HTTP) requests and receives the response over the internet. These requests contain a method which provides the instructions to carry out the request. The methods are a set of verbs (e.g. GET, POST, PUT, DELETE) but starting with SAS 9.4M3, any method that conforms to the HTTP/1.1 standard and is recognizable by the target web server is acceptable.

PROC HTTP code is very much tailored for the task at hand but below is a simple GET request. Note that when the IN parameter is omitted from PROC HTTP, the GET method is the default, but has been included below for clarity. The URL below targets an open-source HTTP request and response service, making it ideal to test our connection:

```
*--- Temporary filename for request response data;
filename resp temp;

*--- Sample request to testing site;
proc http
   url="http://httpbin.org/get"
   method="GET"
   out = resp;
run;
```

We can view the response in the SAS log, many of the return codes from PROC HTTP will be familiar from browsing the web. The return codes are given as automatic SAS macro variables and can be used in debugging, a response of "200 OK" means everything has ran as expected.

```
NOTE: 200 OK
NOTE: PROCEDURE HTTP used (Total process time):
      real time           0.40 seconds
      cpu time            0.01 seconds
```

Note the use of the reserved work TEMP in the filename statement, this is common when using PROC HTTP. In this case a temporary file containing the response is associated with your SAS WORK area and can be viewed in SAS using infile to read the response.

```
*--- Read response;
data resp;
 infile resp length=len;
 length text $1000.;
 input text $varying1000. len;
run;
```

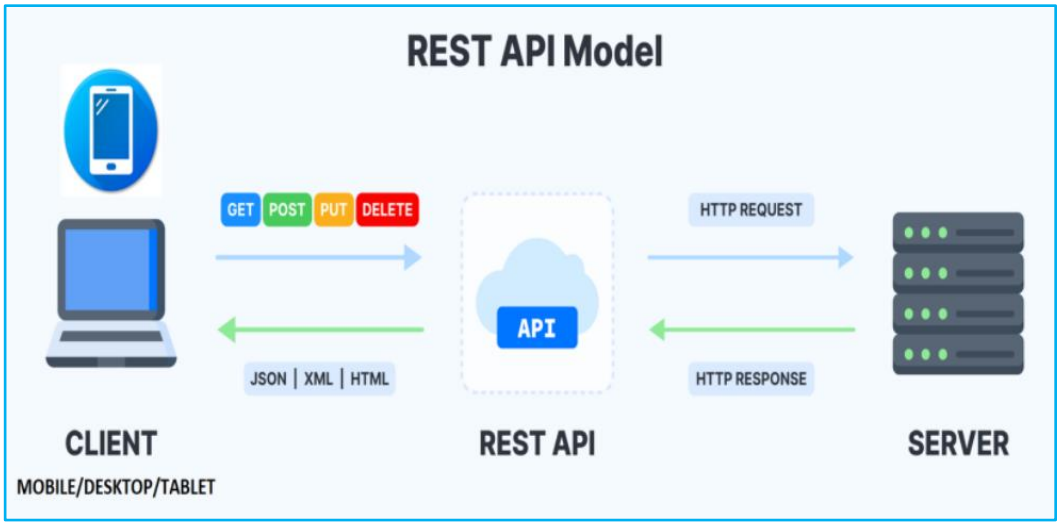
Partial view of RESP dataset produced from infile statement:

	text
1	{
2	"args": {},
3	"headers": {
4	"Accept": "*/*",
5	"Host": "httpbin.org",
6	"User-Agent": "SAS/9",

REST API

It is impossible to talk about cloud-based data interactions without reference to REST APIs. An API, or *application programming interface*, is a set of rules that define how applications or devices can connect to and communicate with each other. A REST API is an API that conforms to the design principles of the REST, or *representational state transfer* architectural style. For this reason, REST APIs are sometimes referred to as RESTful APIs [2]. REST is the most popular API standard for web application.

Figure below shows the REST API model [3]. We can view the HTTP Requests via the methods through the API and the HTTP Response to this. Connection to the CDISC Library and SharePoint, discussed later in this paper, will be through REST APIs.



CAPTURING NCI CONTOLLED TERMINOLOGY

The National Cancer Institute (NCI) website contains an archive [4] of the current and all previous controlled terminology for SDTM, ADaM and SEND.

For SDTM this is released quarterly in a number of formats to suit user needs. The URL required can be extracted from the link by simply right clicking on the link and 'copy link address'. This is the URL that is supplied to the PROC HTTP call as below for the XLS file. This is a very simple example of retrieving a file online and copying it to a local area, no requirement for any security credentials in this example. This can be further processed to e.g. SAS dataset for integration with automation tools.

```
%let model=SDTM; *CDISC model SDTM or ADaM;
%let ct_date=2022-09-30; * CT release date;

*--- URL download path;
%let
urlxls=%str(https://evs.nci.nih.gov/ftp1/CDISC/&model./Archive/&model.%20T
erminology%20&ct_date..xls);

*--- company area for storing CT;
%let storexls=%str(P:\Company Shared Folders\CDISC\&model.\Controlled
Terminology\&model. Terminology &ct_date..xls);
```

```
*--- write file to CT area;
filename storexls "&storexls";
proc http
 url= "&urlxls"
 method="GET"
 out=storexls;
run;
```

CDISC LIBRARY

BACKGROUND

The CDISC Library is a cloud-based metadata repository that lives on the Microsoft Azure platform. Connection to the Library is via the CDISC Library API and this allows us to view the standards directly in the browser or connect programmatically. Earlier this year, the CDISC Library **[5]** became freely available to all, including non-members. This is part of CDISC's drive to enhance collaboration in the industry and ensure maximum support for interoperability and automation. While the NCI archive provided the Controlled Terminology files, the CDISC Library provides this and a host of standards metadata covering different CDISC models and versions. This allows the CDISC Library to be more of a one-stop shop for all standards.

JSON RESPONSE

The default response from the CDISC Library and REST APIs in general is in JSON (JavaScript Object Notation) format. The primary reason for this is that JSON is lightweight, more so than XML. It is still possible to receive standards in various formats form the CDISC Library but in XLSX for example, only complete standards can be extracted. This means a complete controlled terminology file or an entire CDISC model Implementation Guide (IG). With JSON individual domains within the IG can be targeted and individual codelists can be retrieved making JSON advantageous. JSON can be extra work to get it into a useful format but there are options with SAS having the JSON libname engine **[6]** and excellent work by Lex Jansen on using SAS and PROC LUA to parse the JSON response **[7]**.

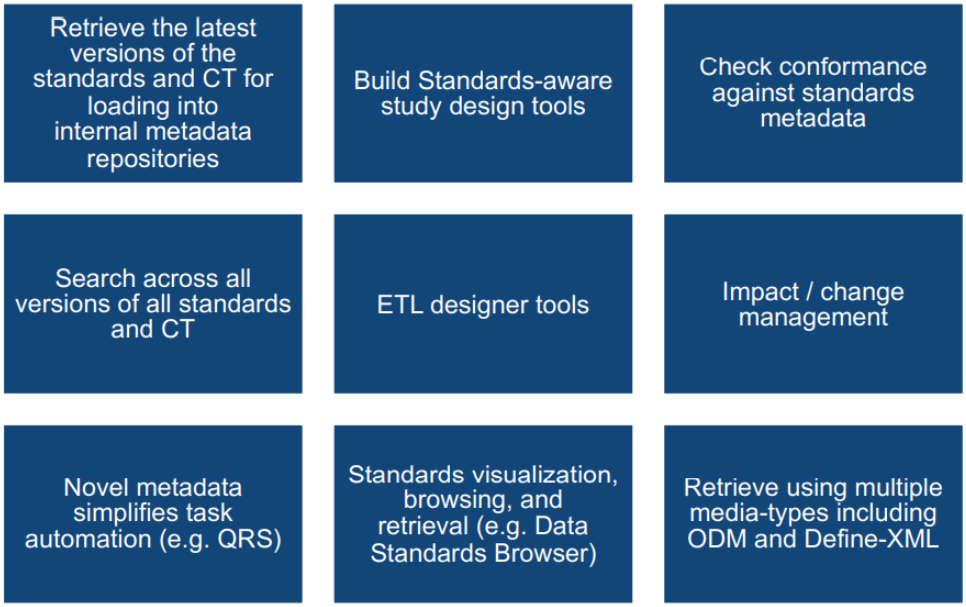
CONTROLLED TERMINOLOGY EXAMPLE

To repeat the above exercise of retrieving the Controlled Terminology file we first need to sign up for a CDISC Library account and receive an API key. This is the first look at using security alongside PROC HTTP and it is the method of choice for the CDISC Library API. With this in hand we can use the API key, given as a macro variable in this case to download the desired file:

```
%let model=SDTM; *CDISC model SDTM or ADaM;
%let ct_date=2022-09-30; * CT release date;

filename response "P:\Company Shared Folders\CDISC\&model.\Controlled
Terminology\&model.ct-&ct_date..xlsx";

proc http
  url = "https://library.cdisc.org/api/mdr/ct/packages/
         %lowcase(&model.)ct-&ct_date"
  out = response;
  headers  "api-key"="&api_key."
           "Accept"="application/vnd.ms-excel";
run;
```

PRACTICAL APPLICATION

At this point it would be worthwhile asking exactly what the purpose is of downloading these files and what benefit it can bring. CDISC have given some examples of the more common use cases [8]:



From PHASTAR perspective, the primary purpose is to ensure internal metadata repositories are maintained. At a fundamental level we can now browse standards within SAS allowing more flexible filtering, much easier than wading through excel / PDF documents. At a more technical level these standards are integral to a number of automation tasks including creation of dataset specifications, SDTM code and define.xml.

MICROSOFT SHAREPOINT

BACKGROUND

SharePoint is a document management and collaboration platform that companies use as part of the Microsoft 365 suite, commonly known as SharePoint Online. SharePoint is protected by user login credentials and everything you can access within the platform is user specific, this can be granted on an individual basis or on the fact you as a user belong to certain groups.

Typically, documentation within SharePoint can remain in the platform and it comes with its own host of analytical tools [9], but there are cases when we can benefit from reading or writing to SharePoint from the Clinical programming environment. Study Tracker documents hosted in SharePoint can be used to help support study team workflow such as QC status of deliverables and titles / footnotes information for a given output.

The Study Tracker document is in the form of an Excel Online file and makes study team collaboration simple as large numbers of people can work within the same document simultaneously. Excel Online is part of Microsoft 365 and replaces the traditional shared desktop Excel Workbook for collaboration, shared desktop Excel files are now officially considered a legacy feature. Updates made within the Study Tracker will be picked up in real time by SAS without the need to worry about saving or closing files. SharePoint Online gives the desired functionality and security 'out of the box', as an existing company platform without the need for an expensive / bespoke tool.

CONNECTION TO SHAREPOINT

In order to connect SAS to SharePoint we will again be using PROC HTTP but this time the security method will be OAuth2 and requires the use of the REST-based Microsoft Graph API. Tremendous work with detailed steps, videos and sample code has been done by Chris Hemedinger and Joseph Henry explaining connection of SAS to Microsoft 365 (OneDrive, SharePoint, Teams) [10]. Application of these steps to SharePoint specifically has been expanded by Simon Todd [11]. Without repeating the set-up steps, I will focus on the interaction once we have obtained an access token to use as our authentication.

Within SharePoint, we will assume our Study Tracker has already been loaded to a SharePoint Site, that site will have a Drive ID associated with it. As explained in the references above, it is an iterative process to drill down to a specific file of interest but we can look at an example of how this is done with the following queries:

[i] This returns a Site ID for the given site name replacing {query}
[ii] This returns Drives IDs for the given folders within the site when replacing {site-id} with response from **[i]**
[iii] This returns download URLs for files within each folder when replacing {drive-id} with response from **[ii]**

[i] `https://graph.microsoft.com/v1.0/sites?search={query}`
[ii] `https://graph.microsoft.com/v1.0/sites/{site-id}/drives`
[iii] `https://graph.microsoft.com/v1.0/drives/{drive-id}/items/root/children`

By querying sequentially in this way, we can programmatically move through SharePoint Site -> Site Folders -> Folder Contents. In practice the JSON response from each query is consumed and the required value passed to the next query as a macro variable. This is an open code representation of step **[1]** where {query} has been replaced with actual Site name 'qc-trackers'. Our Access Token obtained during set-up is being passed as a macro variable to the oauth_bearer parameter:

```
filename resp temp;
proc http url="https://graph.microsoft.com/v1.0/sites?search=qc-trackers"
     method="GET"
     oauth_bearer="&access_token."
     out = resp;
run;

libname tracker json fileref=resp;

data site;
  set site.value;
run;
```

Looking at the work dataset SITE we can partially see the Site ID value which will feed into step **[ii]**

id	lastModifiedDateTime	name
phastar1.sharepoint.com,5dd2e29a-b88a-4a	0001-01-01T08:00:00Z	qc-trackers

When we run the query in **[iii]**, we can view contents of the Site folder and filter on the item of interest as we'll know the name associated with the file, in this case TEST TRACKER.xlsx.

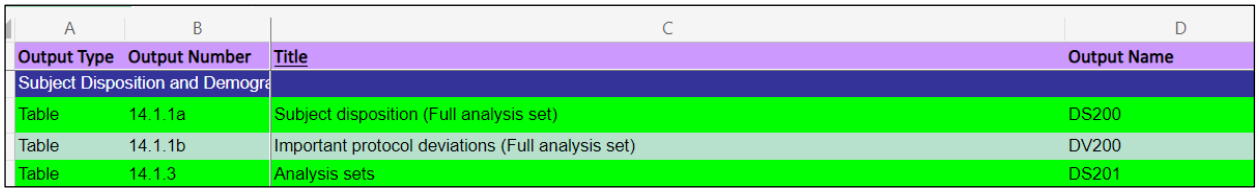
	name	webUrl	cTag	size	_microsoft_graph_downloadUrl
	TEST TRACKER.xlsx	https://phastar1.sharepoint.com/sites/qc	"c:{FC0BE42A-EF12-452F-AB49-F4BCE	198620	https://phastar1.sharepoint.com/sites/qc

Finally we have a download URL which again we will pass to PROC HTTP and retrieve the required file for further processing.

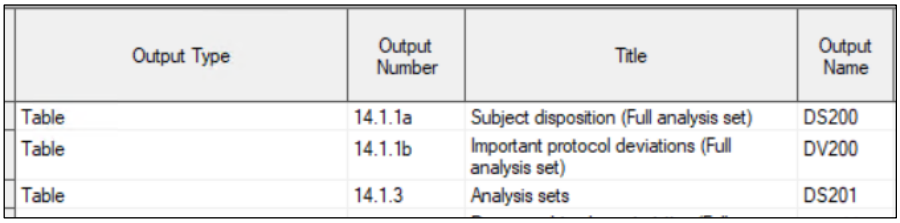
```
*--- Excel Online sheet to import;
%let sheet_name=outputs;

*--- mask download URL before passing to PROC HTTP;
data _null_;
  set tracker.value;
  call symputx("trackerurl",_microsoft_graph_downloadUrl);
run;

%let download=%superq(trackerurl);
```

```
*--- write XLSX to temporary work space;
filename fileout "%sysfunc(pathname(work))/tracker.xlsx";
proc http url="&download."
  method="GET"
  oauth_bearer="&access_token."
  out = fileout;
run;

*--- import outputs tab with title / footnotes;
proc import file=fileout
  out=__&sheet_name
  dbms=xlsx replace;
  getnames=YES;
  sheet="&sheet_name.";
run;
```

The Excel Online file has now been successfully imported and the title and footnotes are readily available for the output programs. The first image here is within the interactive Excel Online file and the second is our dataset created in the proc import step above.



	A	B	C	D
	Output Type	Output Number	Title	Output Name
	Subject Disposition and Demogra			
	Table	14.1.1a	Subject disposition (Full analysis set)	DS200
	Table	14.1.1b	Important protocol deviations (Full analysis set)	DV200
	Table	14.1.3	Analysis sets	DS201



Output Type	Output Number	Title	Output Name
Table	14.1.1a	Subject disposition (Full analysis set)	DS200
Table	14.1.1b	Important protocol deviations (Full analysis set)	DV200
Table	14.1.3	Analysis sets	DS201

UPLOADING FILE TO SHAREPOINT

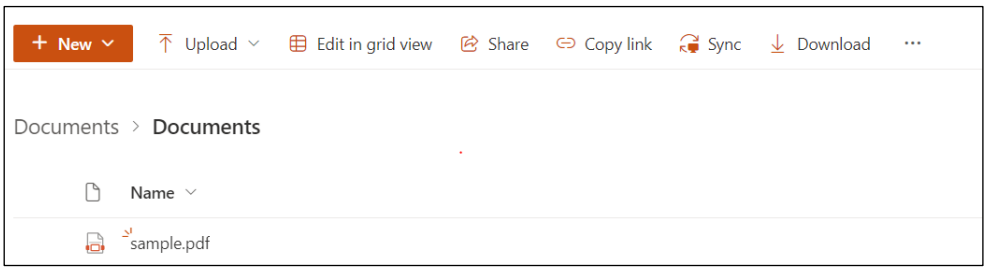
So far we've looked at the connection to extract information from SharePoint in the form of an Excel Online file. All the code so far has employed the GET method within PROC HTTP to download information. We can also do the reverse when desired, either writing entire files to a specified folder or directly within a specified file in the case of Excel Online.

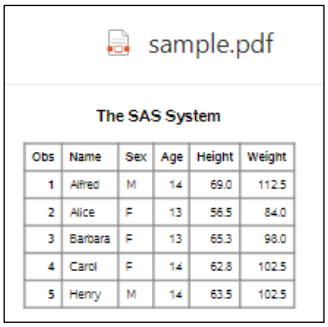
We will look at the more straightforward case of writing an entire file to a specified folder first. In this example the first section of code is creating a simple PDF example file and storing it temporarily in the SAS work area ready for transfer. The second part now uses PROC HTTP with the PUT method, which uploads the file to a given drive (&driveid) and folder (&folderid) within SharePoint. The IN parameter references our PDF filename.

```
*--- create a simple file to upload;
%let targetFile=sample.pdf;
filename tosave "%sysfunc(getoption(WORK))/&targetFile.";

ods pdf file=tosave;
proc print data=sashelp.class (obs=5);
run;
ods pdf close;
```

```
*--- upload PDF with PUT method;
filename resp temp;
proc http
url="https://graph.microsoft.com/v1.0/drives/&driveid./items/&folderid.:/&
targetFile.:/content"
  method="PUT"
  oauth_bearer="&access_token."
  in=tosave
  out=resp;
run;
```

We can inspect the log and this confirms everything has ran as expected. On viewing the particular folder within SharePoint, we can see 'sample.pdf' in the directory and on opening confirms the first 5 observations from SASHELP.CLASS.





Obs	Name	Sex	Age	Height	Weight
1	Alfred	M	14	69.0	112.5
2	Alice	F	13	56.5	84.0
3	Barbara	F	13	65.3	98.0
4	Carol	F	14	62.8	102.5
5	Henry	M	14	63.5	102.5

WRITING TO AN EXISTING FILE WITHIN SHAREPOINT

If you have an Excel workbook which already has Conditional Formatting or Data Validation possibly spanning several Worksheets, it can be desirable to write to an existing sheet rather than create from scratch.

In this case we are using the PATCH method to directly update cells within an existing Excel Online workbook. We are creating some sample JSON data to pass to the PROC HTTP, in practice this would be done with PROC JSON from a SAS dataset when passing a larger sample of data. Full code for this is given in Appendix A as an active session must first be created when updating with PATCH.

```
*--- Put the body of the JSON content in external file;
filename json_in temp;
data _null_;
  file json_in;
  input;
  put _infile_;
datalines;
{
  "index": null,
  "values": [
    ["Table", "14.1.1", "Test titles 1", "DM201"],
    ["Table", "14.1.2", "Test titles 2", "DM202"]
```

```
  ]
}
run;

*--- HTTP Patch request to update cell grouping based on JSON;
filename resp temp;
proc http
  method="PATCH"
  url="https://graph.microsoft.com/v1.0/me/drives/&driveid./
items/&fileid./workbook/worksheets/Outputs/range(address='A3:D4')"
  ct="application/json"
  oauth_bearer="&access_token"
  in=json_in
  headerin=header
  out=resp;
run;
```

In this way, automation of study tracker documents can be achieved by passing relevant output metadata in the form of a JSON file, we can see the sample data has been successfully uploaded to our workbook. This saves manual and error prone tasks like copying title and footnote information between documents. Items like production and validation batch run dates can also be passed dynamically at runtime to the tracker document and again saves the study team manual updates.

A	B	C	D
Output Type	Output Number	Title	Output Name
Subject Disposition and Demogra			
Table	14.1.1	Test titles 1	DM201
Table	14.1.2	Test titles 2	DM202

SECURITY

Most of the sample code in this paper has used security in the form of an API Key or an Access Token providing OAuth 2.0 authentication. While the code is picking these values up in the form of macro variables, it is a good idea to keep any authentication information in files separate from programming code and in a location only accessible to yourself. This information can be used to mimic your login credentials and therefore essential to store securely, a default area can be within your SASUSER path which is accessible only per user.

CONCLUSION

Using traditional desktop files for collaboration like a shared Excel workbook has often been a familiar tool for programmers and simple to read from SAS. With collaboration moved to the Cloud, Excel Online provides an enhanced experience for sharing but with additional technical steps required to maintain a link with SAS. The author believes this is worth the investment, opening capability to read and write to documents in the Cloud allows use of single source documents for collaboration and reduce manual and error-prone tasks.

This use case along with the others discussed provide an insight into the application of the hugely versatile PROC HTTP in an ever-expanding area.

REFERENCES

[1] PROC HTTP SAS Documentation
https://documentation.sas.com/doc/en/pgmsascdc/9.4_3.5/proc/n0bdg5vmrpyi7jn1pbgbje2atoov.htm

[2] REST APIs https://www.ibm.com/cloud/learn/rest-apis

[3] REST API Model https://www.c-sharpcorner.com/article/restful-api-in-net-core-using-ef-core-and-postgres/

[4] National Cancer Institute (NCI) website https://evs.nci.nih.gov/ftp1/CDISC/SDTM/Archive/

[5] CDISC Library https://www.cdisc.org/cdisc-library

[6] JSON Libname Engine https://blogs.sas.com/content/sasdummy/2016/12/02/json-libname-engine-sas/

[7] Lex Jansen, Extracting Data Standards Metadata and Controlled Terminology from the CDISC Library using SAS© with PROC LUA https://www.lexjansen.com/pharmasug/2021/AD/PharmaSUG-2021-AD-168.pdf

[8] CDISC Webinar April 2021: Ideas for using the CDISC Library and a Look at What's Coming Next (Anthony Chow, Sam Hume)
https://www.cdisc.org/events/webinar/cdisc-library-ideas-using-cdisc-library-and-look-whats-coming-next

[9] Spencer Renyard, Using Power BI to present the QC status for studies, PHUSE 2022, TT13

[10] Chris Hemedinger and Joseph Henry, Using SAS with Microsoft 365 (OneDrive, Teams, and SharePoint)
https://blogs.sas.com/content/sasdummy/2020/07/09/sas-programming-office-365-onedrive/

[11] Simon Todd, Efficient Implementation and Applications of PROC HTTP in Analysis and Reporting:
https://www.lexjansen.com/phuse/2019/ad/AD10.pdf

CONTACT INFORMATION

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

John McDade
PHASTAR
2D Bollo Ln,
Chiswick,
London, W4 5LE,
United Kingdom

Email: john.mcdade@phastar.com
Web: phastar.com

APPENDIX: SAMPLE CODE TO OPEN WORKBOOK SESSION WITH PATCH METHOD

```
*--- these variables should be generated dynamically;
*--- based on site / folder name and file name;
%let driveid=xxxxxxxxxxxx;
%let fileid=xxxxxxxxxxxxx;

*--- create work book session;
filename resp temp;
proc http
  method="POST"
  url="https://graph.microsoft.com/v1.0/me/drives/&driveid./items/
       &fileid./  workbook/CreateSession"
  ct="application/json"
  oauth_bearer="&access_token"
  out=resp;
run;

*--- read in session ID;
libname tracker json fileref=resp;
data _null_;
 set tracker.root;
 call symputx('sessid',strip(id));
run;

*--- create header file;
filename header "%sysfunc(pathname(work))/session.txt";
data _null_;
 file header;
 put "workbook-session-id: &sessid.";
 file print;
run;

*--- create JSON file;
filename json_in temp;
data _null_;
file json_in;
input;
put _infile_;
datalines;
{
  "index": null,
  "values": [
    ["Table", "14.1.1", "Test titles 1", "DM201"],
    ["Table", "14.1.2", "Test titles 2", "DM202"]
  ]
}
run;

*--- pass header file to open session and JSON content;
filename resp TEMP;
proc http
  method="PATCH"
  url="https://graph.microsoft.com/v1.0/me/drives/&driveid./items/
       &fileid./workbook/worksheets/Outputs/range(address='A3:D4')"
  ct="application/json"
  oauth_bearer="&access_token"
```

```
  in=json_in
  headerin=header
  out=resp;
run;
```