Paper SAS4439-2020

SAS® and Microsoft Office 365: A Programming Approach to Integration

Chris Hemedinger, SAS Institute Inc., Cary, NC

ABSTRACT

Many of us are now using cloud-based productivity tools like Microsoft Office 365. And some of us are using SAS® software in the cloud, too. For those of us who use SAS to read and create Microsoft Excel documents, cloud-based files can add an extra wrinkle when we automate the process. It also adds some exciting possibilities!

The Microsoft Office 365 suite offers APIs to discover, fetch, and update our documents using code. In this paper, I show you how to use SAS programs to reach into your Microsoft OneDrive and SharePoint cloud to read and update your files. And I show you how to leverage the collaborative features of Microsoft Teams to publish useful updates to your colleagues and yourself. The approach relies on the REST APIs in Microsoft Office 365 and on the HTTP procedure in SAS.

The paper covers each step, including:

- registering a new application in your Microsoft Office 365 account
- authentication and access using OAuth2
- using SAS to explore your document folders in OneDrive and SharePoint and import into SAS data sets
- using SAS to create new documents in OneDrive and SharePoint
- sending rich messages to your teammates in Microsoft Teams

In addition to the detailed steps, this paper references a GitHub repository with code that you can adapt for your own use.

INTRODUCTION

If your work environment is like ours here at SAS, you are seeing more of your data and applications move to the cloud. It's not yet a complete replacement for having local files on your desktop machine, but with cloud storage and apps -- like Microsoft SharePoint Online and OneDrive -- I can now access my work documents from any browser and any device, including my smartphone. I can update now my spreadsheets while waiting in the dentist office. *Oh joy.*

However, now that we look to the cloud for document storage and collaboration, our local SAS processes no longer have direct access to these files. They are no longer on our C: drive or in a shared network folder. We can still access and update these documents with interactive, point-and-click methods – but automated, unattended batch processing requires different tools.

In the first part of this paper, you will learn to use SAS code and the Microsoft Graph APIs to automate the steps of retrieving and updating documents in the Microsoft Office 365 cloud. These documents might include Microsoft Excel files with data your SAS jobs must read or Microsoft Excel files or Word documents that your SAS jobs produce.

To maintain this SAS code, you will need to understand OAuth2 authentication, how REST APIs work, and how to read and interpret JSON responses. I have shared SAS macros that

hide much of the complexity to help you to get started, so you can be productive immediately and learn more as needed over time.

In the second part of this paper, you will learn how to automate the push of messages and other content from your SAS process to a Microsoft Teams channel. The API for this technique is simple, and it's easy to get started. You can invest more time in building a JSON payload to create more sophisticated messages.

USING SAS WITH MICROSOFT OFFICE 365: AN OVERVIEW

Microsoft Office 365 uses an OAuth2-style authentication flow to grant access and permissions to third-party apps. If you are accustomed to the simpler style of user and password authentication (ah, those were the days), OAuth2 can be intimidating. Joseph Henry does a great job of deconstructing OAuth2 -- with code samples -- in his 2017 SAS Global Forum paper, "Show Off Your OAuth."

When we are writing SAS programs to access Microsoft OneDrive or SharePoint, we are actually writing a third-party app. Writing an app requires several setup steps, a few of which cannot be automated. Fortunately, these need to be done only once, or at least infrequently. Here is an outline of the steps:

1. Register a new client application at the Microsoft Azure Portal (https://portal.azure.com/) in the App registrations section. You need to sign in with your Office 365 credentials, which might be your primary organization credentials if you have single-signon with Active Directory.
2. While you are signed into Office 365, use your browser to navigate to a special web address to obtain an authorization code for your application.
3. Plug this authorization code into a SAS program (PROC HTTP step) to retrieve an OAuth2 access token (and a refresh token).
4. With the access token, you can now use PROC HTTP and the Microsoft Office 365 APIs to retrieve your OneDrive or SharePoint Online folders and files, download files, upload files, and replace files.

You will have to complete Step 1 only once for your application or project. Steps 2 and 3 typically need to be done only once or at least only occasionally. The access token is valid for a limited time (usually 1 hour), but you can always exchange the refresh token for a new valid access token. This refresh token step can be automated in your program, so you usually run this step only once per session. Occasionally that refresh token can be revoked (and thus made invalid) when certain events occur, such as changing your account password. When that happens, you need to repeat steps 2 and 3 to get a new set of access and refresh tokens.

Although the examples in this paper are specific to OneDrive, the exact same authentication flow and steps can be used for all Microsoft Office 365 APIs. Have fun with SharePoint Online, Outlook, Teams, Excel, and all your favorite cloud-based Microsoft apps.

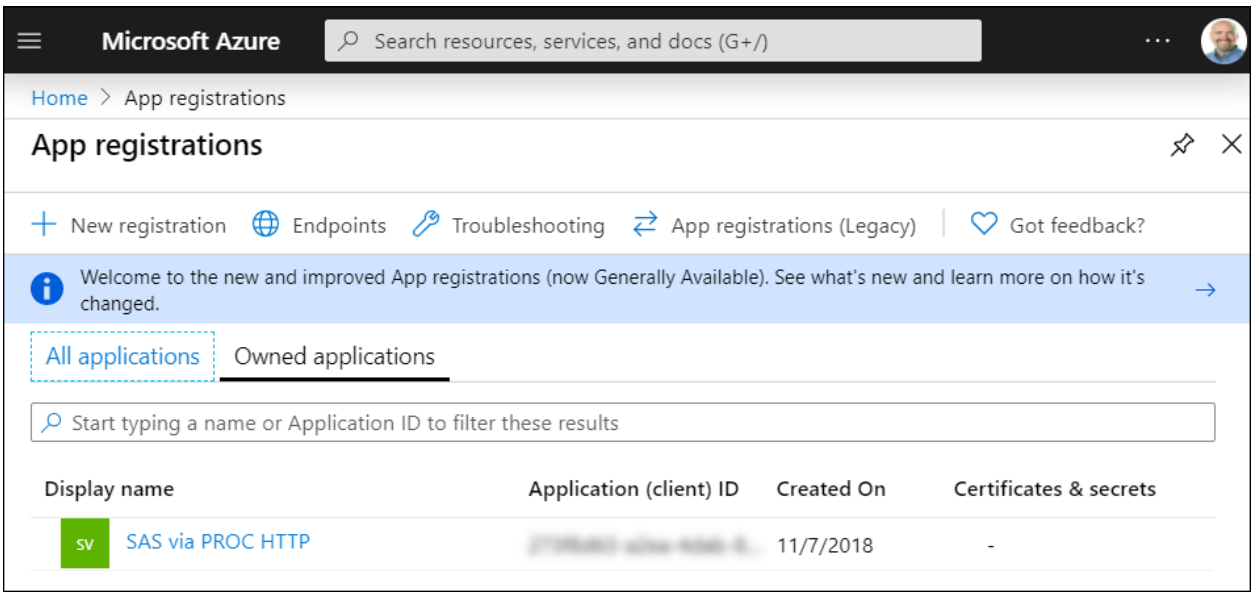
STEP 1: REGISTER YOUR APPLICATION

Before you can use the Microsoft Graph API, you need a way to identify your application and grant it the necessary permissions to do its work. At the end of this step, you will have the following information:

- Your Tenant ID, which is specific to your organization and would be the same for any application that is registered within your company.
- Your application's Client ID, which is unique to your application.
- Optionally, a client "secret" phrase, needed only for applications that do their work without an authenticated user. The examples in this paper assume that you are

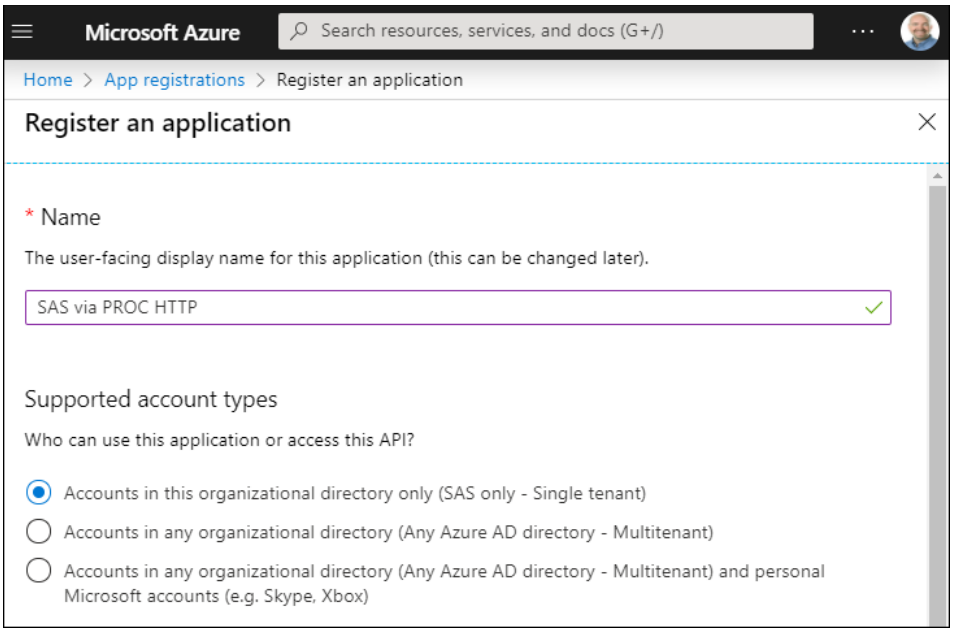
obtaining your authorizations by logging in to your account to get the authorizations, but it's possible to set up a service account to do the work instead.

To register your application, begin by visiting the Microsoft Azure portal (https://portal.azure.com). Sign in with your Microsoft Office 365 credentials. Within the Azure services section, select App registrations.



Display 1. App Registrations in the Azure Portal

Click **New Registration** to get started. The "Register an application page" presents you with a form where you can complete the details that define your app. Mainly, you are giving it a name and defining its scope. You will probably want to limit its use to just your organization (your company) unless you are collaborating with colleagues who work elsewhere.

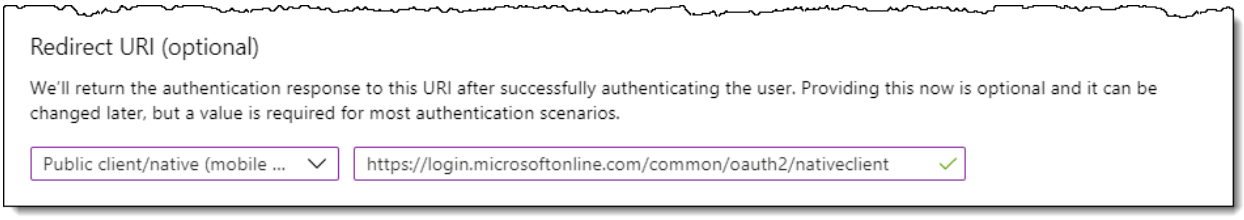


Display 2. Register Your Application

As you register your application, you also need to provide a redirect URL for the authorization flow. In our example, our app is considered "Public client/native (mobile/desktop)." The standard URL to indicate this is:

```
https://login.microsoftonline.com/common/oauth2/nativeclient
```

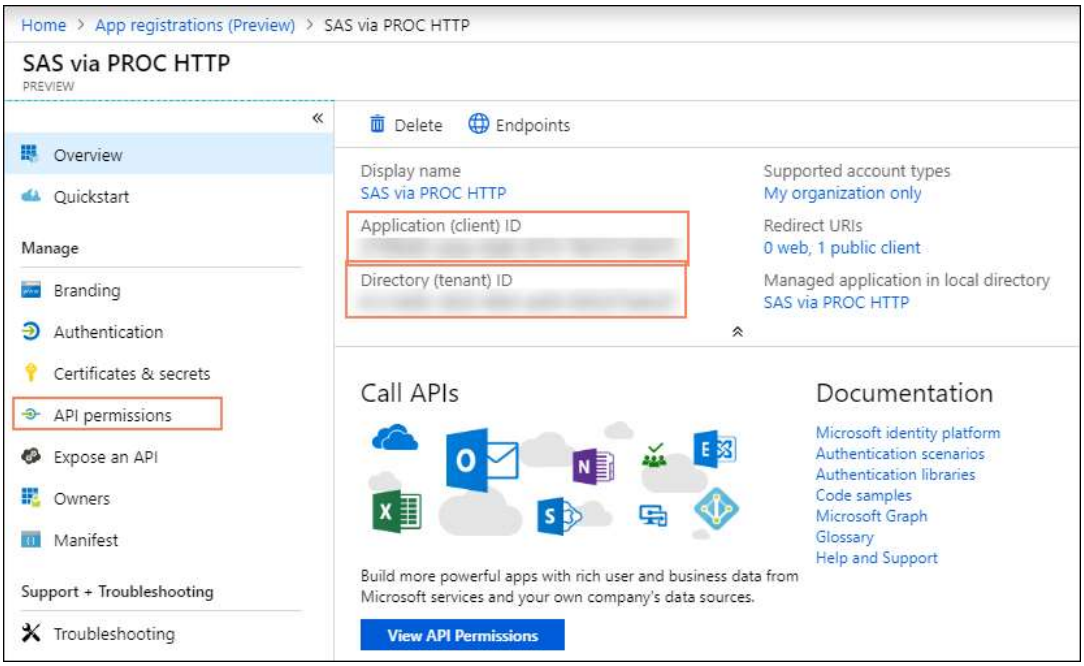
In the Redirect URI section, select this option and specify this URL value.



Display 3. Redirect URI Selections

Finish by clicking Register at the bottom of the page.

When you create an app, you receive a client ID (unique to your app) and tenant ID (unique to your organization). You need these values to obtain your authorization code and tokens later. The application portal provides a sort of control center for all aspects of your app. (Note: I masked out my client ID and tenant ID in this screenshot.)



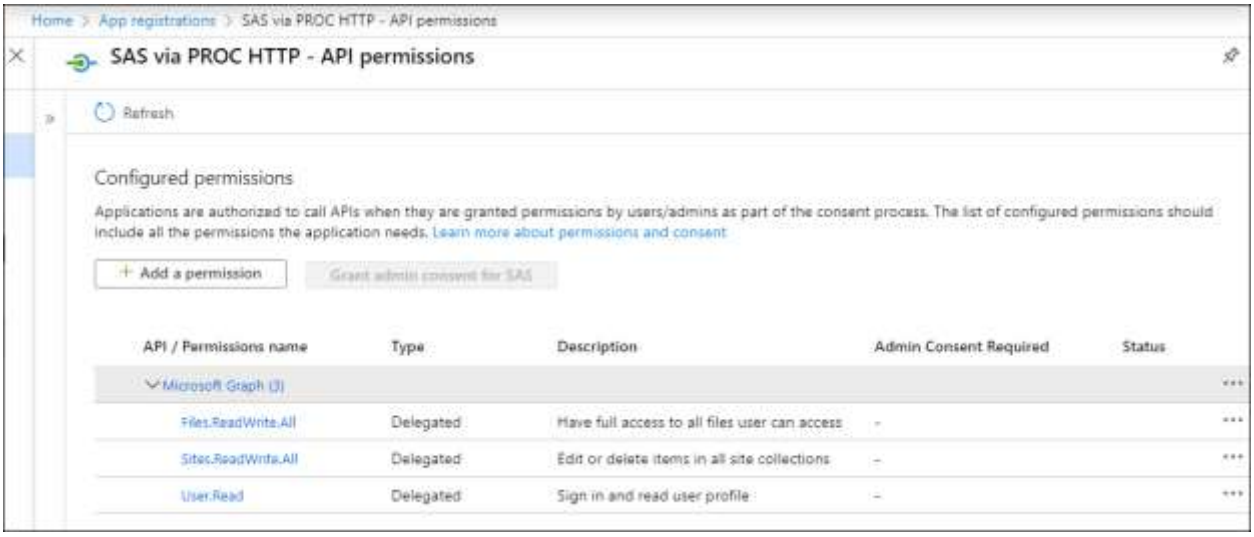
Display 4. Sample Application, Registered

Specifying Your App Permissions

Your app needs specific permissions in order to function. In my example, I want my SAS program to read documents from my OneDrive, add new docs, and update existing docs. Here are the permissions that I need:

- Files.ReadWrite.All: allows the app to read, create, update and delete all OneDrive files that you can access.
- User.Read: allows you to sign into the app with your organizational account and lets the app read your profile.
- Sites.ReadWrite.All (if using SharePoint): allows the app to read, create, update, and delete SharePoint Online files for sites that you can access.

To add these permissions to your app, open the API Permissions tab in the control center. To be clear, these are not permissions that your app will automatically *have*. These are the permissions that will be *requested* when you "sign in to" the app for the first time, and that you will have to agree to before the app can call these APIs on your behalf.



Display 5. Required Permissions for My Sample App

Permission types have their own terminology that is important to understand:

- *Delegated versus Application Permissions*: In our example, we are sticking to Delegated permissions, which allow the application to take actions on behalf of the signed-in user and provides access to the user's data. However, some use cases require use of Application permissions, which allow the application to take actions without a signed-in user and potentially access data across the system and different users.

- *Admin Consent Required*: Some permissions cannot be delegated or granted without the approval of an administrator. This restriction permits the organization to maintain oversight of the important resources that might be accessed by the application and to prevent unauthorized uses. The Microsoft Azure Portal provides an easy way for you to submit a request to an admin, so you can get the permissions that you need. However, I recommend that you follow up (or better yet, precede this) with a formal request to your IT support staff to state what you need and your business case. In my experience, this helps to expedite the process. A good working relationship with IT is important for any SAS user!

The documentation for the Microsoft Graph API provides a comprehensive list of the permission names, whether they are Delegated or Application level, and whether Admin Consent is required. This documentation also includes a helpful 4-minute video on the topic.

Creating a Configuration File

There are a few app-specific values that we need to reference throughout the SAS programs we are writing. I decided to create a configuration file for these settings rather than hardcode them into my SAS statements. Storing these settings in an external file makes it easier for other people to reuse my code in their own applications.

I created a file named config.json that looks like this (but with different tenant_id and client_id values):

```
{
  "tenant_id": "206db638-6adb-41b9-b20c-95d8d04abcbe",
  "client_id": "8fb7804a-8dfd-40d8-bf5b-d02c2cbc56f3",
  "redirect_uri": "https://login.microsoftonline.com/common/oauth2/nativeclient",
  "resource" : "https://graph.microsoft.com"
}
```

By "externalizing" the IDs specific to my account or instance, I can use SAS code to read the values at run time. Note: This code, like all of the code in this paper, uses features from SAS 9.4M5.

```
/*
  File: onedrive_config.sas
  Set the variables that will be needed through the code.
  We will need these for authorization and also for runtime
  use of the service.

  Reading these from a config.json file so that the values
  are easy to adapt for different users or projects.
*/

%if %symexist(config_root) %then %do;
  filename config "&config_root./config.json";
  libname config json fileref=config;
  data _null_;
   set config.root;
   call symputx('tenant_id',tenant_id,'G');
   call symputx('client_id',client_id,'G');
   call symputx('redirect_uri',redirect_uri,'G');
   call symputx('resource',resource,'G');
  run;
%end;
%else %do;
  %put ERROR: You must define the CONFIG_ROOT macro variable.;
%end;
```

STEP 2: OBTAIN AN AUTHORIZATION CODE

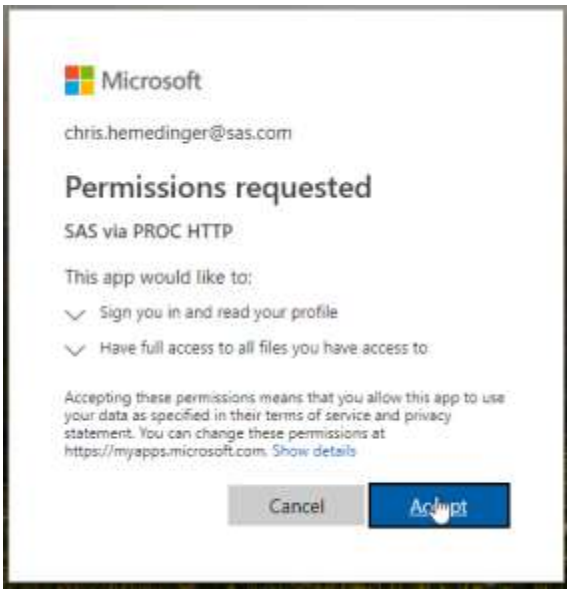
Now that I have defined the application, it's time to "sign into it" and grant it the permission to read and manage content in OneDrive or SharePoint. This step needs to be completed from a web browser while I am signed into my Microsoft Office 365 account. The web address is very long, but we can use a SAS program to generate it for us.

```
/* location of my config file */
%let config_root=/folders/myfolders/onedrive;

%include "&config_root./onedrive_config.sas";

/* Run this line to build the authorization URL -- all one line */
%let
authorize_url=https://login.microsoftonline.com/&tenant_id./oauth2/authoriz
e?client_id=&client_id.%nrstr(&response_type)=code%nrstr(&redirect_uri)=&re
direct_uri.%nrstr(&resource)=&resource.;
options nosource;
%put Paste this URL into your web browser:;
%put -- START -------;
%put &authorize_url;
%put ---END ---------;
options source;
```

This code produces these output lines in the SAS log:

```
 Paste this URL into your web browser:
 -- START -------
https://login.microsoftonline.com/206db638-6adb-41b9-b20c-
95d8d04abcbe/oauth2/authorize?client_id=8fb7804a-8dfd-40d8-bf5b-d02c2cbc56
f3&response_type=code&redirect_uri=https://login.microsoftonline.com/common
/oauth2/nativeclient&resource=https://graph.microsoft.com
 ---END ---------
```

Copy and paste the URL (all on one line, no spaces) into the address bar of your web browser. When you press Enter, you are prompted to grant the required permissions.



Display 6. Prompt for Application Permissions

After you click Accept, the browser redirects to what *looks like* a blank page, but the URL in the browser's address bar contains the authorization code that we need:

Display 7. Access Code in the Address Bar. It's Long.

Copy the value that appears after the `code=` parameter in the URL, but only up to the `&session=` part. It's going to be a very long string -- over 700 characters. We will need that value for the next step.

STEP 3: OBTAIN AN ACCESS TOKEN

My colleague Joseph wrote a few convenient utility macros that can help manage the access token and refresh token within your SAS session. These macros include:

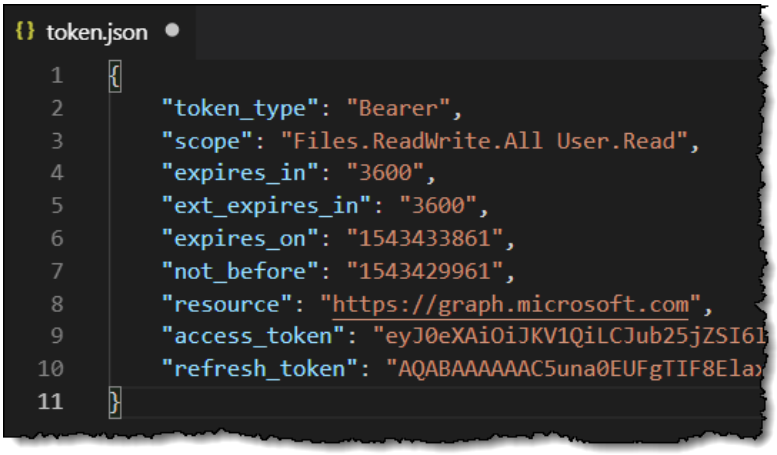
- `%get_token` - get the initial access and refresh tokens, given an authorization code. Remember, an access token expires in about 60 minutes, but the refresh token can be used to get a renewed access token.
- `%refresh` - exchange a valid refresh token for a new access token.
- `%process_token_file` – read or update an external token file so that these values persist beyond your current SAS session.

I am not going to walk through the macro code in this paper, but the SAS programs are straightforward and well-documented. See the section "How to Get This Example Code" at the end of this paper.

With these macros in place, we can paste the (very long) authorization code that we retrieved in the previous step into a macro variable. Then we can run the %get_token macro to generate the tokens and store them in a local file.

```
%let config_root=/folders/myfolders/onedrive;

%include "&config_root./onedrive_config.sas";
%include "&config_root./onedrive_macros.sas";

filename token "&config_root./token.json";
%let auth_code=AQABAAIAAAC5una0EUFgTIF8ElaxtWjTqwohjyfG; * and much more;

/*
  Now that we have an authorization code, we can get the access token.
  This step writes the tokens.json file that we can use in our
  production programs.
*/
%get_token(&client_id.,&auth_code,&resource.,token,tenant=&tenant_id);
```

Running this step creates a new file, token.json, in your designated config folder. Here is a screenshot of what my version of the token.json looks like right now:



Display 8. Example of a Generated token.json File

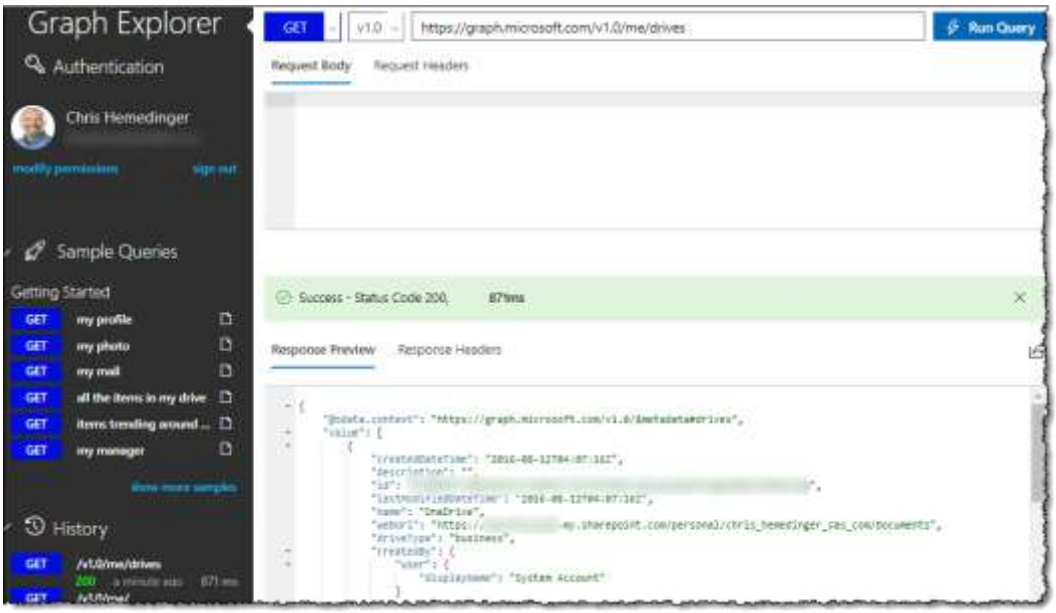
It's *very important* that you keep this file secure. With the information in this file (your refresh token) and your conf.json file (with your client ID and tenant ID), anyone can use these code techniques to impersonate you and access your Microsoft Office 365 data. The best practice is to store this file in a directory that only your user account can read.

STEP 4: CALL THE MICROSOFT GRAPH API

By now you have probably noticed that we are working on Microsoft Azure, which is the Microsoft cloud platform for applications. For the remainder of this article, I will be using methods from the Microsoft Graph API. This REST-based API provides access to almost all of the Microsoft hosted services. For my examples, I will be using methods within the Files component of the API: Drives and Drive Items (folders and files).

You can explore and try the Microsoft Office 365 APIs with the Graph Explorer application from Microsoft. If you sign in with your own account, you can use the APIs with

your own data. This is a great way to try these APIs and discover the correct methods to use before implementing them in your SAS code.



Display 9. The Graph API Explorer Is a Useful Tool for Testing Calls

Initializing and Refreshing the Access Token in a New Session

Now that we have the access and refresh tokens, we can get down to business with some actual OneDrive interactions. Here is how to initialize your SAS session with the tokens.

```
%let config_root=/folders/myfolders/onedrive;

%include "&config_root./onedrive_config.sas";
%include "&config_root./onedrive_macros.sas";

/*
  Path to json file that contains the oauth token information
*/
filename token "&config_root./token.json";

%process_token_file(token);

/* If this is first use for the session, we will likely need to refresh  */
/* the token.  This refresh will also call process_token_file again and update
*/
/* our token.json file.                                                  */
%refresh(&client_id.,&refresh_token.,&resource.,token,tenant=&tenant_id.);

/*
  At this point, we have a valid access token, and we can start using the API.
*/
```

If all goes well, we will have our access token, and it will be stored in a macro variable named &access_token. It's going to be another long and illegible (>700 characters) value.

(Ever hear of the "infinite monkey theorem?" That a monkey hitting a typewriter for an infinite amount of time is certain to produce a certain text, such as the complete works of Shakespeare? Well, that monkey is not going to produce this access token. Plus, who has a typewriter anymore?)

Retrieving the Top-level Drive Identifier (OneDrive)

We will need to explore the OneDrive system from the top-down, using code. First, we need the identifier for the root drive. It's possible for you to have multiple root drives, and if that is the case for you, you will need to modify this code a bit. This code queries the service for your drives and stores the identifier for the first drive in a macro variable. We will need that identifier later to retrieve a list of top-level items.

```
/*
First we need the ID of the "drive" that we are going to use
to list the drives the current user has access to you can do this
*/
filename resp TEMP;
/* Note: oauth_bearer option added in 9.4M5 */
proc http url="https://graph.microsoft.com/v1.0/me/drives/"
     oauth_bearer="&access_token"
     out = resp;
    run;

libname jresp json fileref=resp;

/*
 I have access to only one drive, but if you have multiple drives, you can
filter the set with a WHERE clause on the name value.

 This code creates a data set with the one record for the drive.
*/
data drive;
 set jresp.value;
run;

/* Store the ID value for the drive in a macro variable. */
proc sql noprint;
 select id into: driveId from drive;
quit;
```

Note that this code uses the new OAUTH_BEARER option in PROC HTTP -- a convenient addition when working with OAuth2-compliant APIs. This is shorthand -- and more intuitive syntax -- for placing "Authorization: Bearer TOKEN-VALUE" in the HTTP headers.

Retrieving the Top-level Drive Identifier (SharePoint Online)

The steps for SharePoint Online are nearly the same as for OneDrive, except that we need to reference the site host name (for example, yoursite.sharepoint.com) and the /sites resource (instead of the /me/drives resource).

```
/* Note: oauth_bearer option added in SAS 9.4M5.                        */
/* This code uses the /sites methods in the Microsoft Graph API.            */
/* May require the Sites.ReadWrite.All permission for your app.    */
/* Set these values per your SharePoint Online site.
   Ex: https://yourcompany.sharepoint.com/sites/YourSite
    breaks down to:
       yourcompany.sharepoint.com -> hostname
       /sites/YourSite -> sitepath
   This example uses the /drive method to access the files on the
   Sharepoint site and works just like OneDrive.
   API also supports a /lists method for SharePoint lists.
   Use the Graph Explorer app to find the correct APIs for your purpose.
    https://developer.microsoft.com/en-us/graph/graph-explorer
```

```
*/
%let hostname = yourcompany.sharepoint.com;
%let sitepath = /sites/YourSite;
proc http
url="https://graph.microsoft.com/v1.0/sites/&hostname.:&sitepath.:/drive"
     oauth_bearer="&access_token"
     out = resp;
     run;

libname jresp json fileref=resp;

/*
 This code creates a data set with the one record for the drive.
 You need this object to get the Drive ID.
*/
data drive;
 set jresp.root;
run;

/* Store the ID value for the drive in a macro variable. */
proc sql noprint;
 select id into: driveId from drive;
quit;
```

Retrieve a List of Top-level Folders or Files

With the drive identifier in hand (whether OneDrive or SharePoint), I can use the /children verb on the Microsoft Graph API to get a list of all of the top-level objects in that drive. These objects represent the folders and files that are at the root.

```
/*
 To list the items in the drive, use the /children verb with the drive ID.
*/
filename resp TEMP;
proc http url="https://graph.microsoft.com/v1.0/me/drives/&driveId./items/root/children"
     oauth_bearer="&access_token"
     out = resp;
     run;

libname jresp json fileref=resp;

/* Create a data set with the top-level paths and files in the drive. */
data paths;
 set jresp.value;
run;
```

Here is what I am keeping in my OneDrive right now. It's not too disorganized, is it?

id	lastModifiedDateTime	name	webUrl	cTag	size
01XNKR6V4V2...	2018-03-06T12:58:55Z	Archive	https://sasoffic...	"c:{272DD495-...	2678495972
01XNKR6V77...	2017-11-29T21:51:58Z	ATnT	https://sasoffic...	"c:{43F03CFF-...	474800
01XNKR6V4N...	2017-01-04T17:46:39Z	Attachments	https://sasoffic...	"c:{E04FE48D-...	6515063
01XNKR6V7W...	2017-08-17T19:22:32Z	Ax2017	https://sasoffic...	"c:{478F7CF6-...	927625
01XNKR6V32V...	2016-09-16T13:06:45Z	BCTEG_001	https://sasoffic...	"c:{8B58A97A-...	101375979
01XNKR6VYA...	2017-06-28T14:16:48Z	Communities	https://sasoffic...	"c:{6AB40B00-...	4844382578
01XNKR6VZIZ...	2018-08-16T16:54:00Z	CustomerPresentations	https://sasoffic...	"c:{D747CC28-...	781297579
01XNKR6V4N...	2018-05-30T15:27:14Z	Documents	https://sasoffic...	"c:{6545578D-...	0
01XNKR6V3Y...	2017-07-21T17:40:06Z	EG	https://sasoffic...	"c:{CDB6B678-...	57801929
01XNKR6V4P...	2018-02-23T12:34:18Z	LithiumSDK	https://sasoffic...	"c:{12B4048F-...	2930273688
01XNKR6V3V...	2017-06-29T11:36:26Z	Notebooks	https://sasoffic...	"c:{2AC28F75-...	7581
01XNKR6V7Y...	2018-08-27T19:59:12Z	personal	https://sasoffic...	"c:{E98986F8-...	30338726
01XNKR6V6PL...	2018-03-26T11:19:39Z	Profiles for L77841	https://sasoffic...	"c:{C88058CF-...	69934430
01XNKR6V3O...	2018-08-16T16:53:12Z	Projects	https://sasoffic...	"c:{9CB4366E-...	448542289
01XNKR6V544...	2018-08-29T13:51:32Z	sasCommunity	https://sasoffic...	"c:{796AE0BC-...	4699770115
01XNKR6V4JS...	2016-11-03T17:29:08Z	SASForumDenmark	https://sasoffic...	"c:{2DFD9189-...	13325894
01XNKR6V5M...	2018-04-06T19:28:08Z	SASGF	https://sasoffic...	"c:{08AC4FAC-...	97049865
01XNKR6V4O...	2018-07-03T16:36:45Z	UserCommunications	https://sasoffic...	"c:{B4E0758E-...	62755905
01XNKR6V4G...	2018-05-22T16:40:42Z	CustomerExperience.docx	https://sasoffic...	"c:{3D302986-...	33786
01XNKR6V3RJ...	2018-10-19T20:28:54Z	OOO.docx	https://sasoffic...	"c:{418D4971-...	15339

Display 10. Example of My top-level Folder in OneDrive

List the Files in a Particular Folder

If I am interested in exploring a specific folder, I need to find the folder identifier as it's known to OneDrive. Using PROC SQL and SELECT INTO, I can find the folder by its name and store its ID in another macro variable. Then, I use the /children verb again, but this time with the folder ID instead of the "root" constant.

```
/*
 At this point, if you want to act on any of the items, you just replace "root"
 with the ID of the item. So to list the items in the "SASGF" folder I have:
  - find the ID for that folder
  - list the items within by using the "/children" verb
*/

/* Find the ID of the folder I want. */
proc sql noprint;
 select id into: folderId from paths
  where name="SASGF";
quit;

filename resp TEMP;
proc http
url="https://graph.microsoft.com/v1.0/me/drives/&driveId./items/&folderId./children"
     oauth_bearer="&access_token"
     out = resp;
     run;

/* This code creates a data set of the items in that folder,
   which might include other folders.
*/
libname jresp json fileref=resp;
data folderItems;
 set jresp.value;
run;
```

Here are the items from my SASGF folder. Can you tell that I never throw anything away?

lastModifiedDateTime	name	webUrl	cTag	size	_microsoft_graph_downlo...
2018-09-15T21:58:22Z		https://sasoffic...	"c:{7B25FC0E-...	20453324	
2018-04-08T15:35:22Z	papers	https://sasoffic...	"c:{D6743966-...	26587787	
2018-04-16T20:15:41Z	registration	https://sasoffic...	"c:{1AAE784A-...	1439152	
2018-04-10T13:22:42Z	TechTalks2018	https://sasoffic...	"c:{3E770AA6-...	11201222	
2017-03-21T18:37:40Z	2017_sasgfwebcast_schedule.xlsx	https://sasoffic...	"c:{36F36ACE-...	26005	https://sasoffic...
2016-04-12T17:18:08Z	About Communities_2016.pptx	https://sasoffic...	"c:{07F3A671-...	3054098	https://sasoffic...
2016-04-20T16:45:43Z	applewriteins.xlsx	https://sasoffic...	"c:{31FB0B36-...	8100	https://sasoffic...
2017-03-27T20:55:43Z	Demo_Tech_Talk.egp	https://sasoffic...	"c:{C526042E-...	51467	https://sasoffic...
2018-11-27T19:29:09Z	iris.xlsx	https://sasoffic...	"c:{E7364448-...	10156	https://sasoffic...
2016-04-20T14:13:14Z	Online Communities Linkup.pptx	https://sasoffic...	"c:{070C5F1C-...	1125151	https://sasoffic...
2015-04-24T14:41:50Z	Prompter script for SAS Tech Talks 20...	https://sasoffic...	"c:{B0A6B29D-...	14921	https://sasoffic...
2016-04-20T04:51:35Z	Prompter script for SAS Tech Talks 20...	https://sasoffic...	"c:{14FA45D8-...	15894	https://sasoffic...
2017-04-01T22:11:11Z	Prompter script for SAS Tech Talks 20...	https://sasoffic...	"c:{1AD64608-...	15633	https://sasoffic...
2018-04-05T19:13:50Z	Prompter script for SAS Tech Talks 20...	https://sasoffic...	"c:{87C193D2-...	14872	https://sasoffic...
2016-04-18T15:22:39Z	RankExamples.pptx	https://sasoffic...	"c:{1A9B3D9F-...	80055	https://sasoffic...
2017-02-16T14:43:49Z	SAS Global Forum 2017 - Webcast Sc...	https://sasoffic...	"c:{DF6AEAE4...	19685	https://sasoffic...
2016-04-21T07:19:52Z	SAS Professional Forum Google+ SA...	https://sasoffic...	"c:{A88FE45A-...	130081	https://sasoffic...
2016-04-21T07:06:20Z	SAS Professional Forum LinkedIn SA...	https://sasoffic...	"c:{C6780FF1-...	395335	https://sasoffic...
2015-04-29T13:19:57Z	SAS SECURITY Vision and Roadmap...	https://sasoffic...	"c:{BD60037A-...	5997902	https://sasoffic...
2014-04-09T17:16:11Z	sas_tech_talks_14.xlsx	https://sasoffic...	"c:{D16B4A8D-...	14809	https://sasoffic...
2015-05-05T17:08:37Z	sas_tech_talks_15.xlsx	https://sasoffic...	"c:{3992C9D9-...	13868	https://sasoffic...
2016-04-22T19:08:08Z	sas_tech_talks_16.xlsx	https://sasoffic...	"c:{308877E9-...	15941	https://sasoffic...
2017-04-10T18:48:16Z	sas_tech_talks_17.xlsx	https://sasoffic...	"c:{75EC4FC2-...	19321	https://sasoffic...
2018-04-06T19:28:01Z	sas_tech_talks_18.xlsx	https://sasoffic...	"c:{37A0EC0C-...	17566	https://sasoffic...

Display 11. Drilling Down to a Particular Folder

Download a File from OneDrive and Import into SAS

I know that I keep a spreadsheet named "sas_tech_talks_18.xlsx" in this SASGF folder. With the `/content` verb, I can download the file from OneDrive and store it in the file system that is local to my SAS session. Then, I can use PROC IMPORT to read it into a SAS data set.

```
/*
 With a list of the items in this folder, we can download
 any item of interest by using the /content verb.
*/

/* Find the item with a certain name. */
proc sql noprint;
 select id into: fileId from folderItems
  where name="sas_tech_talks_18.xlsx";
quit;

filename fileout "&config_root./sas_tech_talks_18.xlsx";
proc http
url="https://graph.microsoft.com/v1.0/me/drives/&driveId./items/&fileId./content"
     oauth_bearer="&access_token"
     out = fileout;
     run;

/* Import the first sheet into a SAS data set. */
proc import file=fileout
 out=sasgf
 dbms=xlsx replace;
run;
```

Boom! I have just downloaded my data from the cloud and brought it into my SAS session.

Add a New File to OneDrive

We can build wonderful documents from SAS too, and it's important to be able to share those. By using the PUT method with the /content verb, we can copy a file from the local SAS session into a target folder on OneDrive. Most often, this file will probably be an Excel spreadsheet or maybe a PDF report.

```
/*
  We can upload a new file to that same folder with the PUT method and /content verb.
  Notice the : after the folderId and the target filename.
*/

/* Create a simple Excel file to upload. */
%let targetFile=iris.xlsx;
filename tosave "%sysfunc(getoption(WORK))/&targetFile.";
ods excel(id=upload) file=tosave;
proc print data=sashelp.iris;
run;
ods excel(id=upload) close;

filename details temp;
proc http
url=
"https://graph.microsoft.com/v1.0/me/drives/&driveId./items/&folderId.:/&targetFile.:/content"
  method="PUT"
  in=tosave
  out=details
  oauth_bearer="&access_token";
run;

/*
  This code returns a json response that describes the item uploaded.
  This step pulls out the main file attributes from that response.
*/
libname attrs json fileref=details;
data newfileDetails (keep=filename createdDate modifiedDate filesize);
 length filename $ 100 createdDate 8 modifiedDate 8 filesize 8;
 set attrs.root;
 filename = name;
 modifiedDate = input(lastModifiedDateTime,anydtdtm.);
 createdDate  = input(createdDateTime,anydtdtm.);
 format createdDate datetime20. modifiedDate datetime20.;
 filesize = size;
run;
```

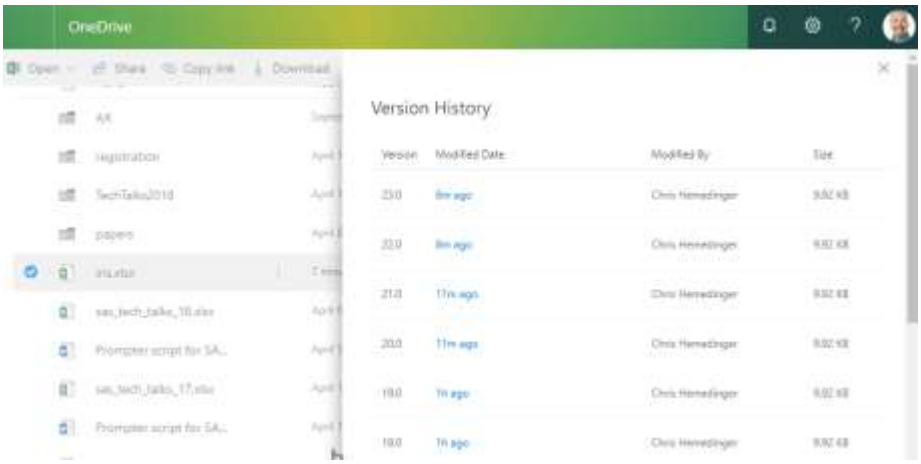
Replace or Update a File in OneDrive

If you want to replace an existing file, then you will want to perform the additional step of retrieving the unique ID for that file from OneDrive. When you PUT the new version of the file into place, its history and sharing properties should remain intact. Here is my code for navigating the folder and file structure in my OneDrive and finally replacing an existing file.

```
/*
  If you want to replace a file instead of making a new file,
  then you need to upload it with the existing file ID.  If you
  do not replace it with the existing ID, some sharing properties
  and history could be lost.
*/
/* Create a simple Excel file to upload. */
%let targetFile=iris.xlsx;
filename tosave "%sysfunc(getoption(WORK))/&targetFile.";
ods excel(id=upload) file=tosave;
```

```
proc print data=sashelp.iris;
run;
ods excel(id=upload) close;

/* Navigate the folder and file IDs from my OneDrive. */
proc sql noprint;
 select id into: folderId from paths
  where name="SASGF";
quit;

proc http
url="https://graph.microsoft.com/v1.0/me/drives/&driveId./items/&folderId./children"
     oauth_bearer="&access_token"
     out = resp;
     run;

libname jresp json fileref=resp;
data folderItems;
 set jresp.value;
run;

/* Find the ID of the existing file. */
proc sql noprint;
 select id into: fileId from folderItems
  where name="iris.xlsx";
quit;

libname attrs json fileref=details;
proc http
url="https://graph.microsoft.com/v1.0/me/drives/&driveId./items/&fileId./content"
 method="PUT"
 in=tosave
 out=details
 oauth_bearer="&access_token";
run;
```

As you can see from my OneDrive history for this file, I have tested this program a few times -- resulting in 23 revisions of this file in its history!



Display 12. Example History for a File Created or Modified by SAS

How to Get This Example Code

You can find the source files for these examples on GitHub.

I have organized this code into 5 different files in order to make it easy to reuse:

- onedrive_config.sas – This file reads the fields from the conf.json including your client_id and tenant_id and sets them as global macro variables,.
- onedrive_setup.sas – This file contains the SAS statements that represent the code that you will need to run to get your authorization code and first access code. You need to run this code only once.
- onedrive_macros.sas – This file contains three utility macros that help you to create, refresh, and manage your access token and refresh token in your token.json file
- onedrive_example_use.sas – This file contains the sample SAS steps that I used in this article. They won't quite work for you as-is because you do not have the same files that I do. (Unless *you do* have the same files, in which case...creepy.) My hope is that you can read and adapt them for your own content.
- onedrive_sharepoint_example.sas – This file contains the sample SAS steps for reading and writing files with SharePoint Online. The basic steps are the same as for OneDrive, except that you use the /sites resource instead of the OneDrive-specific methods.

I also included a template for the conf.json file, with obvious placeholders for the client_id and tenant_id that you will substitute with your own values. You will also need to change the statements that define &CONFIG_LOC -- the location of your configuration directory where you are storing these files. I developed these examples in SAS® University Edition -- yes, this works there! I also ran the code from my full SAS environment via SAS® Enterprise Guide®.

PUBLISHING TO A MICROSOFT TEAMS CHANNEL

Like many organizations, we (SAS) invest a considerable amount of time and energy into gathering metrics and building reports about our operations. However, reports are useful only when the intended audience is tuned in and refers to them regularly. With a small additional step, you can use SAS to bring your most interesting data forward to your team -- automatically.

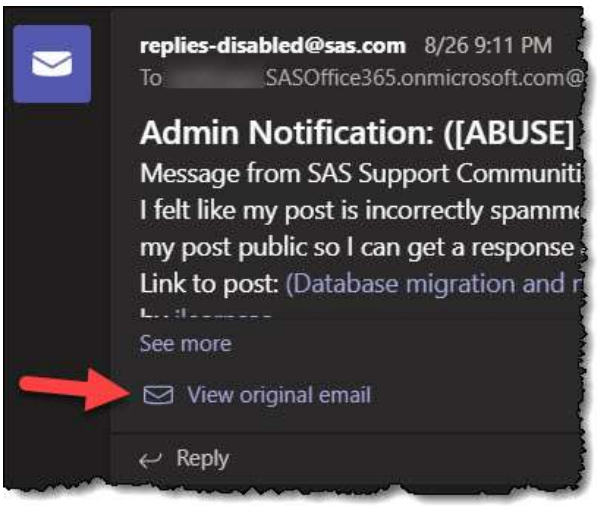
Whether you use Microsoft Teams or Slack, automated alerting and updates are a great opportunity to keep your teams informed. Each of these tools offers fit-for-purpose connectors that can tie in with information from other popular operational systems (Salesforce, GitHub, Yammer, JIRA, and many more). For cases where a built-in connector is not available, the webhook approach enables you to easily create your own.

I will share two methods for automating updates to a Microsoft Teams channel: using email and using a webhook with an API call.

LAZY METHOD: SEND EMAIL TO THE CHANNEL

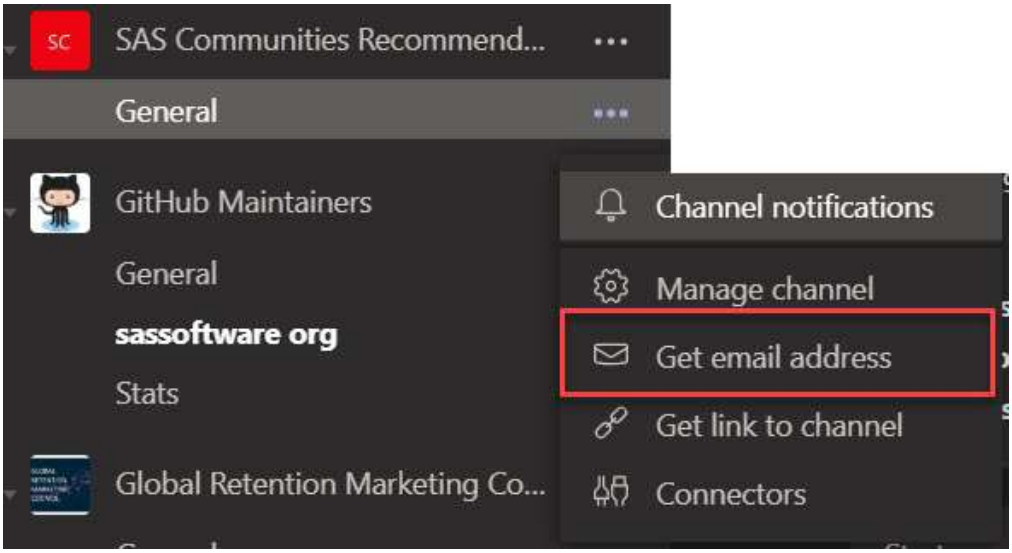
Before I cover the "deluxe" method for sending content to a Microsoft Teams channel, I want to make sure you know that there is a simple method that involves no coding and no need for APIs. The message experience is not as nice, but it does the job. You can simply "send email" to the channel. If you are automating output from SAS, it's a simple, well-documented process to send email from a SAS program with the FILENAME EMAIL method.

When you send email to a Microsoft Teams channel, the message notice includes the message subject line, sender, and the first bit of the message content. To see the entire message, you must click the View original email link in the notice. This "downloads" the message to your device so that you can open it with a local tool (such as your email reader, Microsoft Outlook). My team uses this method to receive certain alerts from our communities.sas.com platform. Here is an example:



Display 13. Example Email Notification in a Channel

To get the unique email address for a channel, right-click on the channel name and select Get email address. Any message that you send to that email address is distributed to the team.



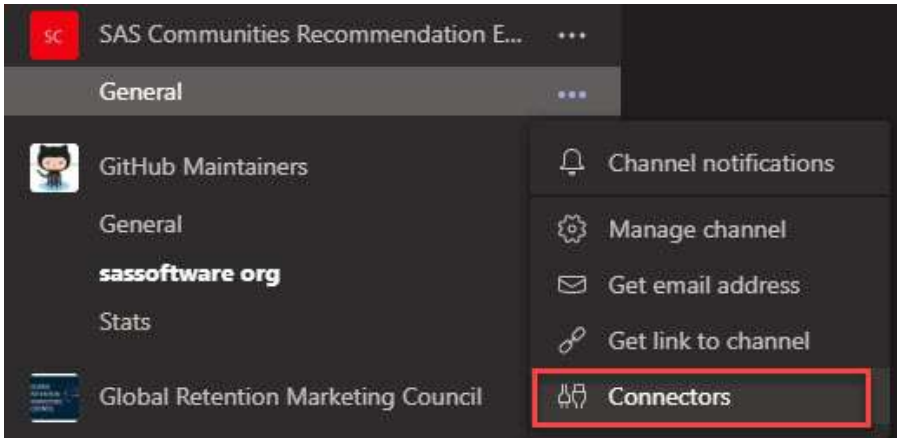
Display 14. Menu Option to Find the Channel Email Address

USING A MICROSOFT TEAMS WEBHOOK

In order to provide a richer, more integrated experience with Microsoft Teams, you can publish content using a webhook. A webhook is a REST API endpoint that enables you to post messages and notifications with more control over the appearance and interactive options within the messages. In SAS, you can publish to a webhook by using PROC HTTP.

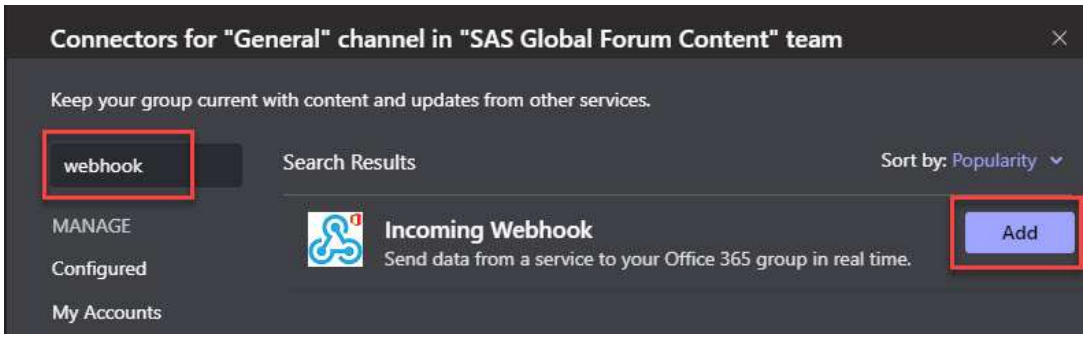
To get started, you need to add and configure a webhook for your Microsoft Teams channel:

1. Right-click on the channel name and select Connectors.



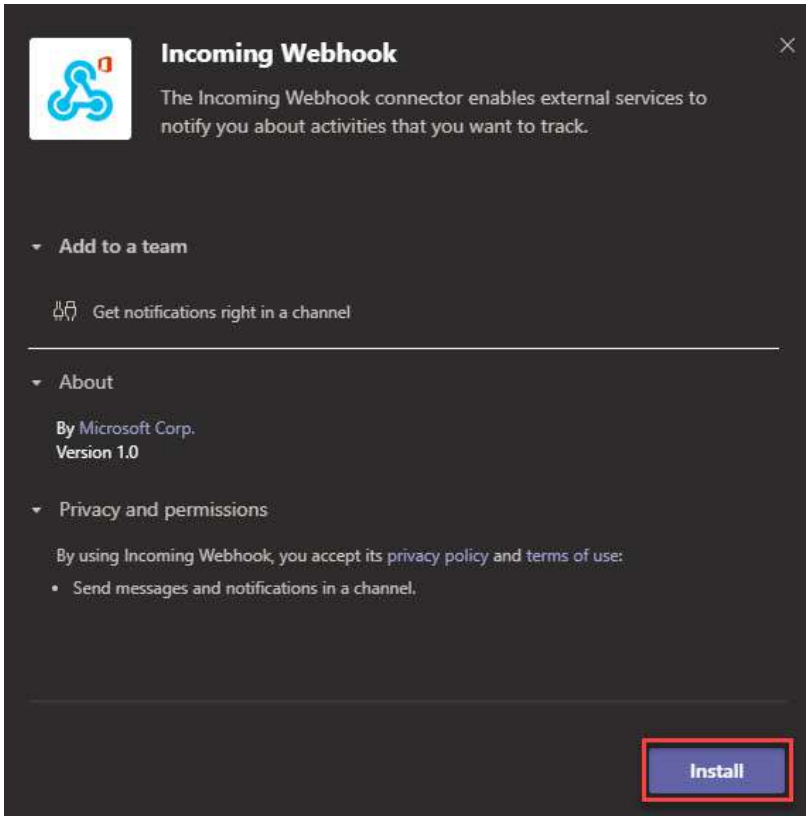
Display 15. Access List of Possible Connectors

2. Microsoft Teams offers built-in connectors for many different applications. To find the connector for Incoming Webhook, use the search field to narrow the list. Then click Add to add the connector to the channel.



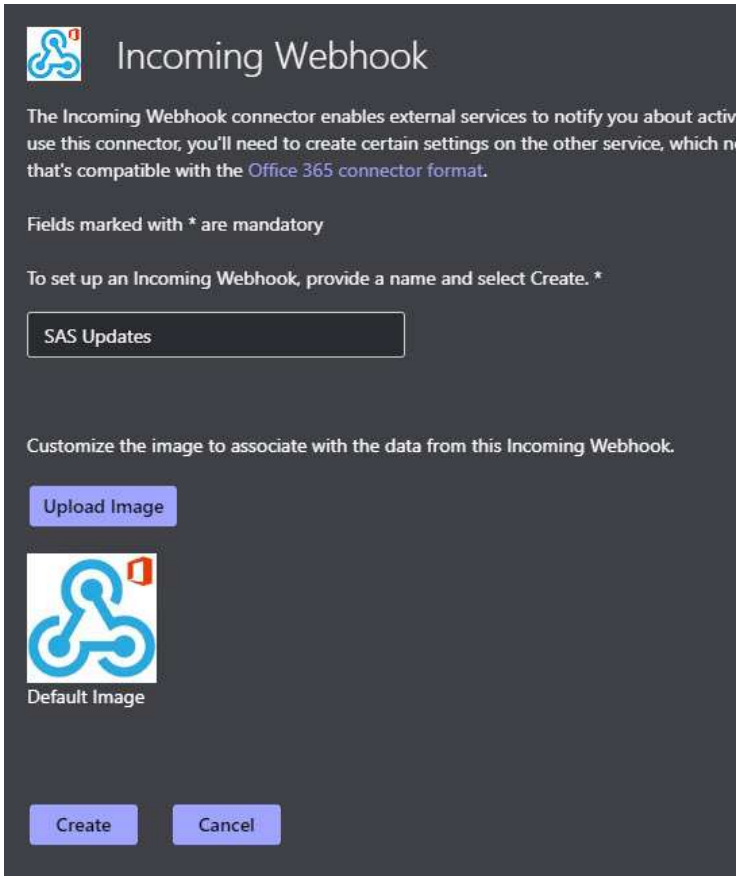
Display 16. Find and Select the Webhook Connector

3. You must grant certain permissions to the connector to interact with your channel. In this case, you need to allow the webhook to send messages and notifications. Review the permissions and click Install.



Display 17. Confirm Permissions and Install

4. On the Configuration page, assign a name to this connector and customize the image. The image is the avatar that is used when the connector posts content to the channel. When you have completed these changes, click Create.



Display 18. Name the Webhook and Change Its Avatar

5. The connector generates a unique (and very long) URL that serves as the REST API endpoint. You can copy the URL from this field -- you will need it later in your SAS program. You can always come back to these configuration settings to change the connector avatar or re-copy the URL.

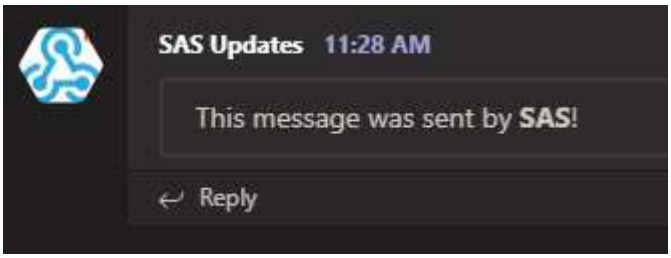
Display 19. Copy the URL from the Webhook Settings

At this point, it's a good idea to test that you can publish a basic message from SAS. The "payload" for a Teams message is a JSON-formatted structure, and you can find examples in the Microsoft Teams reference doc. Here is a SAS program that publishes the simplest message. Add your webhook URL and run the code to verify the connector is working for your channel.

```
filename resp temp;
options noquotelenmax;
proc http
  /* Substitute your webhook URL here */
  url="https://outlook.office.com/webhook/your-unique-webhook-address-it-is-very-
long"
```

```
  method="POST"
  in=
  '{
      "$schema": "http://adaptivecards.io/schemas/adaptive-card.json",
      "type": "AdaptiveCard",
      "version": "1.0",
      "summary": "Test message from SAS",
      "text": "This message was sent by **SAS**!"
  }'
  out=resp;
run;
```

If successful, this step posts a simple message to your Teams channel:



Display 20. Example Message Sent Using a Webhook

DESIGNING A MESSAGE CARD FOR MICROSOFT TEAMS

Now that we have the basic plumbing working, it's time to add some bells and whistles. Microsoft Teams calls these notifications "message cards," which are messages that can include interactive features such as images, data, action buttons, and more.

Designing a Simple Message

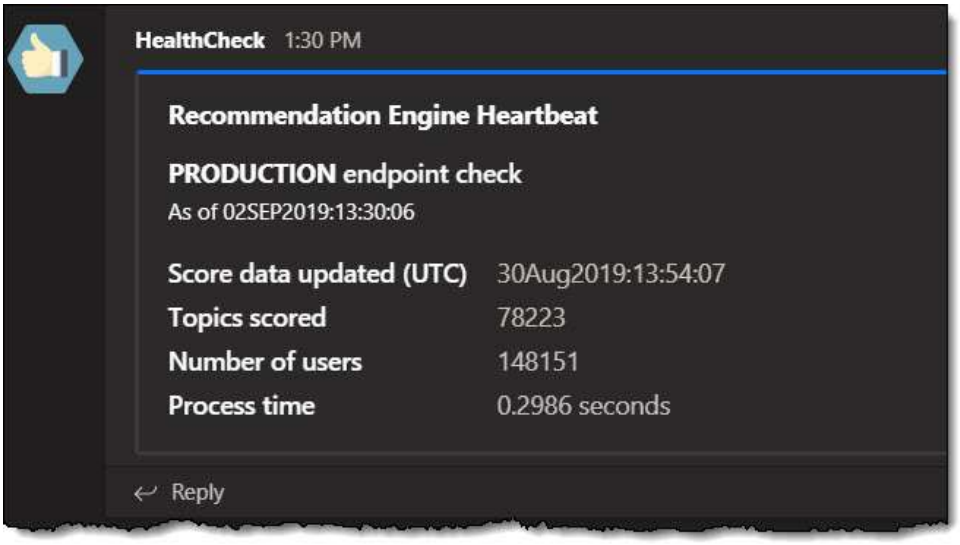
Microsoft Teams supports a large palette of building blocks (expressed in JSON) to create different card experiences. You can experiment with these cards in the MessageCard Playground that Microsoft hosts. The tool provides templates for several card varieties, and you can edit the JSON definitions to modify and design your own.

For one of my use cases, I designed a simple card to show the status of our recommendation engine on SAS Support Communities. (Read "Building a Recommended Engine with SAS" on the SAS Dummy blog for more information about how we built and monitor the recommendation engine.) The engine runs as a service and is accessed with its own API. I wanted a periodic "health check" to post to our internal team that would alert us to any problems. Here is the JSON that I used in the MessageCard Playground to design it.

```
{
    "@type": "MessageCard",
    "@context": "https://schema.org/extensions",
    "summary": "Recommendation Engine Health Check",
    "themeColor": "0075FF",
    "sections": [
        {
            "startGroup": true,
            "title": "**Recommendation Engine Heartbeat**",
            "activityImage": "",
            "activityTitle": "**PRODUCTION** endpoint check",
            "activitySubtitle": "As of 05SEP2019:13:10:30",
            "facts": [
                {
                    "name": "Score data updated (UTC)",
                    "value": "05Sep2019:13:30:06"
                },
                {
                    "name": "Topics scored",
                    "value": "78378"
                },
                {
                    "name": "Number of users",
                    "value": "148686"
                },
                {
                    "name": "Process time",
                    "value": "2.0373 seconds"
                }
            ]
        }
    ]
}
```


Display 21. Example JSON for a Message Card

Much of the JSON is boilerplate for the message. I drew the green blocks to indicate the areas that need to be dynamic -- that is, replaced with values from the real-time API call. Here is what the card looks like when rendered in the Microsoft Teams channel.



Display 22. Example Channel Message with a Simple Message card

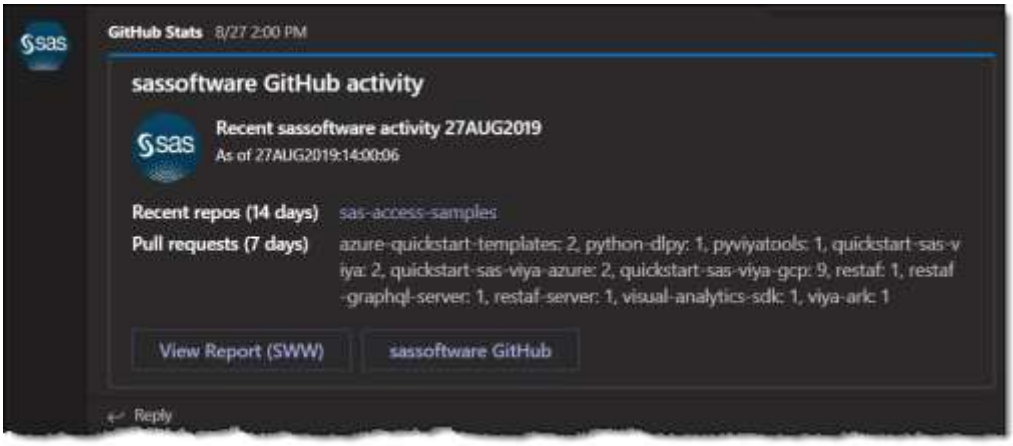
Since my API call to the recommendation engine service creates a data set, I can [run that data through PROC JSON](#) to create the JSON segment I need:

```
/* Reading the results from my API call to the engine. */
libname results json fileref=resp;

/* Prep a simple name-value data set with the results. */
data segment (keep=name value);
 set results.root;
 name="Score data updated (UTC)";
 value= astore_creation;
 output;
 name="Topics scored";
 value=left(num_topics);
 output;
 name="Number of users";
 value= left(num_users);
 output;
 name="Process time";
 value= process_time;
 output;
run;

/* Use PROC JSON to create the segment. */
filename segment temp;
proc json out=segment nosastags pretty;
 export segment;
run;
```

I shared a version of the complete program on GitHub. It should run as is -- but you would need to supply your own webhook endpoint for a channel that you can publish to.

Designing a Message with Actions

I also use Microsoft Teams to share updates about the SAS Software GitHub organization. In "Reporting on GitHub accounts with SAS" on the SAS Dummy blog, I discussed how I use GitHub APIs to gather data from the GitHub service. Each day, my program summarizes the recent activity from github.com/sassoftware and publishes a message card to the team. Here is an example of a daily update:



Display 23. A More Sophisticated Message Card with Action Buttons

This card is fancier than my first example. I added action buttons that can direct the team members to the internal reports for more details and to the GitHub site itself. I used the Microsoft Teams documentation and the MessageCard Playground to design the experience.

Display 24. Example of a Message Prototype in MessageCard Playground

HELPFUL ONLINE RESOURCES

Throughout this paper, I have mentioned several websites for reference information and helpful applications. Here is a list of those resources:

SAS code for OneDrive/SharePoint Online (GitHub)	https://github.com/sascommunities/sas-dummy-blog/tree/master/onedrive
SAS code for Microsoft Teams (GitHub)	https://github.com/sascommunities/sas-dummy-blog/tree/master/msteams
Microsoft Azure Portal	http://portal.azure.com/
Microsoft Graph API reference	https://docs.microsoft.com/en-us/graph/?view=graph-rest-1.0
Microsoft Graph API explorer	https://developer.microsoft.com/en-us/graph/graph-explorer
MessageCard Playground application for Teams	https://messagecardplayground.azurewebsites.net/

CONCLUSION

Many SAS users are also users of Microsoft Office 365. As we use these tools together, there are many opportunities to automate the crossover tasks that we have traditionally handled manually. By learning how the Microsoft Graph APIs can work with PROC HTTP, we can make it easier to share the important results that we generate using SAS – in spreadsheets, documents, and collaborative messaging with the rest of the organization.

REFERENCES

Henry, Joseph. April 2019. "The ABCs of PROC HTTP." *Proceedings of the SAS Global Forum 2019 Conference*. Available at https://www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3232-2019.pdf

Henry, Joseph. April 2017. "Show Off Your OAuth." *Proceedings of the SAS Global Forum 2017 Conference*. Available at http://support.sas.com/resources/papers/proceedings17/SAS0224-2017.pdf

ACKNOWLEDGMENTS

I'd like to give major credit to Joseph Henry, the SAS developer who maintains PROC HTTP. Joseph did the heavy lifting for putting together the code and examples in this article.

He also regularly adds new features to PROC HTTP that make it a more natural fit for calling REST APIs that require special authentication flows, such as OAuth2.

Also, thank you to Angela Hall and her team at SAS for her use case. Angela's team used SAS to access and process email messages from a shared Microsoft Office mailbox. Her use case is a variation that required help from SAS IT for Application Permissions and the Admin Consent options.

And finally, I want to thank two "customers" who used the information from my original blog article to successfully integrate SAS with Microsoft SharePoint Online. Matthew Clarke of Kaiser Permanente helped me to find gaps in my instructions as he worked to implement this in his organization. And Richard Clowes (of the SAS travel department) helped me to understand some special cases with Admin Consent.

CONTACT INFORMATION

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

Chris Hemedinger
SAS Institute Inc.
Chris.Hemedinger@sas.com
Twitter: @cjdinger