

Tales From a Tech Support Guy: The Top Ten Most Impactful Reporting and Data Analytic Features for SAS Programmers

Chevell Parker, SAS Institute Inc.

ABSTRACT

This paper helps you get the most out of SAS software by discussing some features that can make a big impact on your job as a SAS programmer or a data analyst. Some of the areas covered include the following:

- features that highlight the power of SAS reporting, which allow SAS programmers to generate customized reports
- SAS integration with open-source tools, as well as automation
- some of the most impactful features in SAS® Viya®

INTRODUCTION

SAS®9 and the SAS Viya platform have the capability of answering some of the most complex business problems. This paper covers topics and methods that address these convoluted questions. The paper covers key topics within the SAS language and is organized to build on earlier examples. These topics include the following: reporting needs, programming automation, SAS integration with open source, reading files of various formats, SAS Viya features and functionality, and more.

1. USING THE OUTPUT DELIVERY SYSTEM

The most fundamental part of the Output Delivery System (ODS) is the output object. An output object is created when a template definition is combined with data that is generated when a DATA step or procedure is executed. This output object is then sent to the open destination. In the current releases of SAS®9 and SAS Viya, the default destination is the HTML5 destination for SAS® Studio, SAS® Enterprise Guide®, and HTML4 for the SAS display manager. Note that ODS is not limited to generating HTML, but it is the default destination.

Some of the formats that can be requested by the clients or by using the ODS statements include XLSX, PDF, PPTX, DOCX, XML, Output, RTF, CSV, and more. Along with the many ODS destinations that can be generated from this output object, the TEMPLATE procedure provides the styling component with the formatted destinations.

ODS EXCEL

The ODS Excel destination generates output in the Microsoft Office Open XML SpreadsheetML file format (XLSX). This destination automates the generation of Microsoft Excel worksheets directly from SAS®9 and SAS Viya by creating presentation-ready Excel spreadsheets. There are many highlights of this destination, but none are more important than the ability to eliminate manual work. Some of the most used functionality that the options supply includes the ability to name and color worksheet tabs, freeze row and column headers, add filters, start output in specific locations, and more. Other functionality includes the ability to add graphics using both SAS/GPGRAPH® and ODS Graphics, add formulas, and add multiple worksheets per workbook. Example 1 shows how you can use this destination to create customized output similar to Output 1 below.

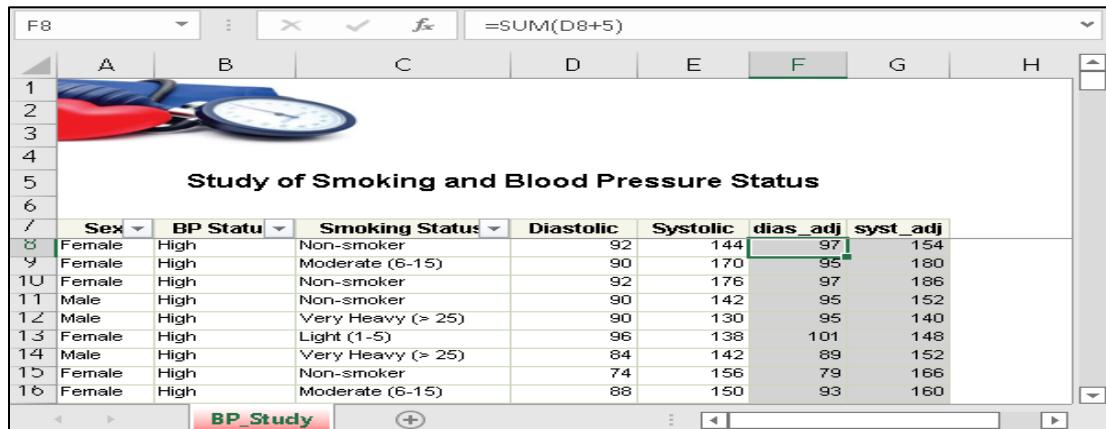
Example 1

```
ods excel options(embedded_titles="yes"
                  frozen_headers="yes" ①
                  autofilter="1-3"
                  sheet_name="BP_Study" sheet_interval="none"
                  tab_color="red") style=styles.meadow;

data _null_; ②
  declare odsout obj();
  obj.image(file:"c:\temp\hbp2.jpg");
run;

title "Study of Smoking and Blood Pressure Status";
proc report data=sashelp.heart(obs=100);
where bp_status="High";
column Sex BP_Status Smoking_Status diastolic Systolic dias_adj syst_adj ;
  define BP_Status / "BP Status";
  define dias_adj / computed style(column)={tagattr="formula:=sum(RC[-
2]+5)" background=lightgray}; ③
  define syst_adj / computed style(column)={tagattr="formula:=sum(RC[-
2]+10)" background=lightgray};
run;
ods excel close;
```

- ① Added Excel options to include titles, freeze column headers, add filters, provide sheet names, modify tab colors, and prevent creation of a new worksheet when a new object is added.
- ② Used the Report Writing Interface to add a logo to the worksheet using the image method.
- ③ Added formulas to the **dias_adj** and **syst_adj** columns using the TAGATTR= attribute.



	Sex	BP Status	Smoking Status	Diastolic	Systolic	dias_adj	syst_adj
8	Female	High	Non-smoker	92	144	97	154
9	Female	High	Moderate (6-15)	90	170	95	180
10	Female	High	Non-smoker	92	176	97	186
11	Male	High	Non-smoker	90	142	95	152
12	Male	High	Very Heavy (> 25)	90	130	95	140
13	Female	High	Light (1-5)	96	138	101	148
14	Male	High	Very Heavy (> 25)	84	142	89	152
15	Female	High	Non-smoker	74	156	79	166
16	Female	High	Moderate (6-15)	88	150	93	160

Output 1. Output from ODS Excel

ODS OUTPUT

The ODS OUTPUT destination can generate output data sets so that you do not have to rely on procedures. You can create data sets with ODS using the ODS OUTPUT destination along with either the object name or label set to a named output data set. Use the ODS TRACE statement to get the name of the object as shown in Example 3, which produces Output 3.

Example 3

```
ods output moments=moments_AP; ①
proc univariate data=sashelp.class;
var age;
run;

proc print data= moments_AP;
run;
```

① Used the ODS OUTPUT statement along with the MOMENTS output object to generate the output data moments_AP.

Obs	VarName	Label1	cValue1	nValue1	Label2	cValue2	nValue2
1	Age	N	19	19.000000	Sum Weights	19	19.000000
2	Age	Mean	13.3157895	13.315789	Sum Observations	253	253.000000
3	Age	Std Deviation	1.49267216	1.492672	Variance	2.22807018	2.228070
4	Age	Skewness	0.06361167	0.063612	Kurtosis	-1.1109255	-1.110926
5	Age	Uncorrected SS	3409	3409.000000	Corrected SS	40.1052632	40.105263
6	Age	Coeff Variation	11.2097909	11.209791	Std Error Mean	0.34244248	0.342442

Output 3. Output from ODS OUTPUT destination

2. GENERATING EMAILS FROM SAS

You might want to generate email as a part of your code to provide customized output or to indicate a successful job completion. The ability to generate emails from SAS is simple, yet powerful. There are a few methods of generating emails, most of which involve interacting with the operating system. None of these methods is as simple as the EMAIL access method of the FILENAME statement.

The EMAIL access method enables you to send an email attachment and write in the body of an email as demonstrated in Example 4. Writing to the body of the email allows you to send the output as text or HTML (SMTP). Sending output as HTML allows flexibility with the style generated.

ATTACHING AN EMAIL

One benefit to creating output as an attached email is that you can specify your file format. For example, you can attach an Excel file that includes formulas to allow the user to further interact with the data in the worksheet. Another example would be to provide a PDF file, which lets the user to print while still preserving the formatting (unlike printing an HTML file). Table 1 lists options that are specifically used with an attached file. These options are used either to attach an email or within the ATTACH= option:

ATTACH=	Specifies the path and name of the attachment
CONTENT_TYPE=	Provides the name and the mime type to use
EXT=	Add the extension of the attached file
LRECL=	Include record length to prevent truncation

Table 1. Options used when attaching files to an email

Example 4

```
Ods pdf file="c:\temp\sample.pdf"; ①
Proc print data=sashelp.class;
Run;
Ods pdf close;
```

```

filename output email to="chevell.parker@aol.com"
  attach=("c:\temp\sample.pdf" ct="application/pdf" ②
  ext="pdf")
  subject="Summary of Accounts";
data _null_; ③
  file output;
  put "Attaching document";
run;

```

- ① Created a sample PDF file using the ODS PDF destination, which is attached in the email.
- ② Attached the PDF file using the ATTACH= option along with additional options.
- ③ Wrote text to the body of the email along with attaching the file to the email.

WRITING TO THE BODY OF THE EMAIL

Writing to the body of an email can be done as text or HTML. The output is sent as HTML when you specify the EMAILSYS=SMTP system option. Set the CONTENT_TYPE= option to text/html to display the output as HTML rather than simply showing HTML tag. If you want to generate HTML output in the body of the email, use the MSOffice2K ODS destination because it uses styles that work well with Outlook and other popular web mail applications.

Some of the benefits of writing to the body of the email include the ability to add styles, hyperlinks, images, and additional functionality available to the browser. Example 5 shows how to send an email (shown in Output 4) to each winner of the North Carolina education lottery of taxes owed using the EMAIL access method.

Example 5

```

options emailsyssmtp emailhost=mailhost.dca.com; ①

data one;
  input name $17. amount_won type $14. email $25.;
format amount_won dollar.;
cards;
Bill Hemmer      22000 Mega Millions b.hemmer@yahoo.com
Gloria Mackenzie 59000 Powerball      g.mack@gmail.com
Ira Curry        64800 Mega Millions ira.curry@gmail.com
Mavis Wanczyk    75800 Powerball      mwanczky@gmail.com
John Robinson    40000 Powerball      jrobinson@gmail.com
Chevell Parker   50000 Pick 4        chevell.parker@aol.com
Chevell Parker   10000 Cash 5        chevell.parker@aol.com
;
run;

%macro mailit(name,email);
filename mailer email to="&email"
  content_type="text/html"
  subject="North Carolina Lottery Winners"
  attach=("c:\temp\nclottery.png" inline="image1"); ②

ods msoffice2k file=mailer style=analysis rs=none;
title1 j=1 '<img src=cid:image1>';
title2 "^{style[url=""https://nclottery.com/"" North Carolina Lottery}"; ③

```

```

proc report data=one(where=(name="&name" and email="&email"));
  column name amount_won type tax;
  compute tax;
    tax_rate=.545;
    tax=amount_won.sum*tax_rate; ④
  endcomp;
  run;
  ods msoffice2k close;
  %mend;

proc sort data=one;
  by name;
  run;

data _null_;
  set one;
  by name;
  if first.name then
    call execute('%mailit('|| name ||','|| email ||')'); ⑤
run;

```

- ① Set the option EMAILSYS=SMTP, which indicates that the output is sent HTML.
- ② Used the ATTACH= option included with the INLINE= option to specify that the image is embedded.
- ③ Referenced the image in the title statement using the CID.
- ④ Used the compute block to calculate the numbers that are shared.
- ⑤ Used the CALL Execute statement in the DATA step to pass each unique name and email to the macro as a parameter, which is used to subset and mail each unique user.

North Carolina Lottery Winners

To: Chevell Parker

Retention Policy: EX-3YR-DELETE (3 years)

Expires: 10/5/2026

9:57 AM

NC Education Lottery

name	amount_won	type	tax
Chevell Parker	\$50000	Pick 4	27250
Chevell Parker	\$10000	Cash 5	5450

Output 4. Email generated using the EMAIL access method

3. SAS AND OPEN SOURCE INTEGRATION

The ability to integrate with third-party applications is necessary for any modern software application. SAS provides abundant ability to incorporate open source code. Table 2 displays a partial list of third-party applications that are integrated with the SAS®9 and SAS Viya platforms, focusing on the most-downloaded application, Python:

Open Source Client	Description	Available
SASPy	Provides APIs for connection and analytics	SAS 9.4+
SAS_Kernel for Jupyter Notebook	Kernel used to add SAS to Jupyter notebook	SAS 9.4
PROC FCMP- Python Objects	Executes Python functions in SAS	SAS 9.4+
Python CAS API's	Uses the SAS Scripting Wrapper for Analytics Transfer (SWAT) package for connection and analytics	Viya 3x +
R CAS API	Uses the SWAT package for connection and analytics	Viya 3x +
PROC PYTHON	Executes Python code from SAS Studio	Viya 4
Python Editor In SAS Studio	Executes Python code in the Python Editor	Viya 4
SAS Configurator for Open Source	Installs Python and R within a PVC	Viya 4
Viya Workbench	A lightweight development environment that supports multiple programming languages and access to scalable cloud compute.	Viya 4*
PROC LUA	Provides the ability to add LUA syntax in SAS	SAS 9.4+

Table 2. Open-Source SAS integration

CONNECTING TO SAS® 9.4 AND THE SAS® VIYA® SPRE

The SASPy package provides the ability to connect to SAS 9.4 and a SAS Viya session using the SAS programming runtime environment (SPRE). This package contains various APIs that allow you to run your analysis using object-oriented Python syntax. In addition, APIs are available to show the SAS equivalent statements from the object-oriented Python code. The APIs for the SASPy package provide interfaces for the following:

- starting a SAS session on the same host as Python or a remote host
- exchanging data between SAS data sets and Pandas dataFrames
- using familiar methods of Python to work with data
- exchanging values between Python variables and SAS macros

Before you connect, you must perform a very simple configuration [step](#) and then use the IDE of your choice to connect. Output 5 shows code and output for connecting to a local SAS session and running an analysis.

```

In [1]: import saspy
import pandas as pd
sas=saspy.SASsession(cfgname='winlocal')

SAS Connection established. Subprocess id is 32592

In [2]: cars=sas.read_csv("http://support.sas.com/documentation/onlinedoc/viya/exemplardatasets/cars.csv")

In [3]: type(cars)

Out[3]: saspy.sasdata.SASdata

In [4]: cars.means().head(2)

Out[4]:
  Variable   N  NMiss  Median      Mean    StdDev     Min     P25     P50     P75     Max
  0   MSRP  428      0  27635.0  32774.855140  19431.716674  10280.0  20329.5  27635.0  39215.0  192465.0
  1   Invoice  428      0  25294.5  30014.700935  17642.117750  9875.0  18851.0  25294.5  35732.5  173560.0

In [5]: carsdf=cars.to_df()

In [6]: type(carsdf)

Out[6]: pandas.core.frame.DataFrame

```

Output 5. Using the SASPy package to connect to SAS 9.4

- ① Added the SASPy and Pandas packages as well as creates the **sas** connection object.
- ② Created the **cars** SASdata object from the `read_csv` method, which reads the Cars CSV file.
- ③ The `Type` function is used to view the type of object which is a SASdata object
- ④ The `means` method is combined with the `head()` method to summarize the data and display the first 2 records.
- ⑤ The **carsdf** object was created as a result of using the `to_df` method creating a python DataFrame .

CONNECTING TO THE CAS SERVER

The Python CAS API uses the SWAT package to connect to the CAS server. In addition to providing the connection information, this package contains all the APIs and methods needed for your analysis. The Python CAS API allows you to generate your analysis using object-oriented syntax similar to the SASPy package. When connecting to the CAS server, you can use either native REST or binary protocol. A binary connection is fastest because it uses C extensions. Analysis performed using the Python CAS API uses CAS actions.

An action is a task performed by CAS at the request of the user that returns a response containing results and status. Actions can load data, manage tables, perform analytics or statistics, and more. If you are familiar with SAS®9, you can think of an action as a SAS procedure on steroids. The connection string below specifies a host, protocol, port, and the username and password needed to connect to the CAS server:

```
conn=swat.CAS('Host-name',port,"username","password",protocol)
```

For increased security, you can move the username and password from connection string to an [AUTHINFO file](#) that stores the credentials. You can then point to the file in the connection string with the AUTHINFO= option. Another authentication method is to specify a token with the PASSWORD= option and the username set to **none**.

Output 6 contains code that connects to the CAS server from Python using Jupyter notebook and shows the output that is produced.

```
In [1]: import os
import swat
os.environ['CAS_CLIENT_SSL_CA_LIST'] = 'c:/temp/CAS/trustedcerts.pem'
conn = swat.CAS("https://cas.Foundation.com/cas-shared-default-https", 8777, authinfo="c:/temp", caslib="casuser")

In [2]: cars=conn.read_csv('https://raw.githubusercontent.com/
    'sassoftware/sas-viya-programming/master/data/cars.csv')

NOTE: Cloud Analytic Services made the uploaded file available as table TMP2IV43FAV in caslib CASUSER(sasctp).
NOTE: The table TMP2IV43FAV has been created in caslib CASUSER(sasctp) from binary data uploaded to Cloud Analytic Services.

In [3]: cars[["MSRP", "MPG_Highway", "MPG_City"]].describe()

Out[3]:
   MSRP  MPG_Highway  MPG_City
count  428.000000  428.000000  428.000000
mean   32774.855140   26.843458   20.060748
std    19431.716674    5.741201    5.238218
min    10280.000000   12.000000   10.000000
25%    20329.500000   24.000000   17.000000
50%    27635.000000   26.000000   19.000000
75%    39215.000000   29.000000   21.500000
max    192465.000000   66.000000   60.000000
```

Output 6. Using the Python CAS API

- ❶ Imported the packages, such as the SWAT and OS, that are needed for the example.
- ❷ Applied the CAS_CLIENT_SSL_CA_LIST environment variable using the trustedcerts.pem file.
- ❸ Created a CARSTable Object using the read_csv method.
- ❹ Added the Describe() method to the TBL data object.

PROC PYTHON AND THE SAS STUDIO PYTHON EDITOR

The ability to run PROC PYTHON code directly from SAS Viya is a beneficial enhancement from the integration of Python and SAS. In addition, a Python editor has been incorporated in SAS Studio that adds generated code within PROC PYTHON. This enhancement has quite a few benefits as you run Python code directly from SAS Studio and exchange data between Python dataFrames and SAS data sets. Other enhancements are the ability to exchange text using macro variables and using callback functions and Python integration with SAS flows.

There are two different methods to execute code using the PYTHON procedure. The first method is to use the INFILE= option in the PROC PYTHON statement, which executes the code from SAS. The second method is to use PROC PYTHON with the SUBMIT and ENDSUBMIT statements and add the Python code in between, as shown in Example 7A. For a method and APIs to modify XLSX files, see [openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files](#). After modifying files using PROC PYTHON, you will never see PROC EXPORT the same.

Example 7A demonstrates reading an Excel file and copying the table to SAS using the SAS callback method.

Example 7A

```
proc python;
submit;
import pandas as pd
print("Inside Python") ❶
carsds = pd.read_excel("/srv/nfs/compute/home/sasctp/cars.xlsx") ❷
ds = SAS.df2sd(carsds, 'work.cars') ❸
endsubmit;
run;
```

- ❶ Used the PRINT function with text that prints a message from within Python.
- ❷ Created the carsds dataFrame from the cars.xlsx file using the read_excel method.
- ❸ Used the SAS callback method to copy the carsds dataFrame to SAS.

Example 7B uses the openpyxl package to update an existing XLSX file that was created with PROC EXPORT, add methods to freeze columns and rows, and add outlines, tab colors, images, and protections. Output 7 shows what this code generates.

Example 7B

```
/* Create PROC Export output to update */
proc export data=sashelp.class
  outfile="/srv/nfs/compute/home/sasctp/Sample.xlsx"
  dbms=xlsx
  replace;
  sheet="Sample";
run;

/* Update the XLSX file */
proc python;
submit;
from openpyxl import load_workbook; ❶
from openpyxl.drawing.image import Image
wb=load_workbook('/srv/nfs/compute/home/sasctp/Sample.xlsx') ❷
ws=wb.active
ws.freeze_panes='A2' ❸
ws.freeze_panes='B2'
ws.row_dimensions.group(1,10)
ws.sheet_properties.tabColor = "1072BA"
img=Image("/srv/nfs/compute/home/sasctp/SGPlot2.png")
ws.add_image(img,'G2')
wb.save('/srv/nfs/compute/home/sasctp/Sample_update.xlsx') ❹
endsubmit;
run;
```

- ❶ Adds the openpyxl and other packages that are needed for the example.
- ❷ Creates the wb workbook object and ws worksheet object from the workbook and loads the active worksheet from PROC EXPORT.
- ❸ Applies various methods such as freeze_panes, group, tabColor, add_image to the worksheet object (ws).
- ❹ Adds the save method to the workbook object (wb), saving the updated file.



Output 7. Using PROC Python to update an XLSX file

4. MANAGING DATA IN CAS

SAS Viya is a cloud-enabled, in-memory analytics platform that provides quick, accurate, and reliable analytical insights. SAS Viya is able to run your analysis in the cloud by connecting to the CAS server. Once connected to the CAS server, you can use the CASL programming language, which is a new programming language that is understood by the CAS server. Although you can use the CASL language, some unsupported features might prevent the code from running on the CAS sever and run it in SAS instead. You can use the MESSAGELEVEL=1 option to determine where the code runs.

CLOUD ANALYTIC SERVICE HIGHLIGHTS

CAS exploits the scalability of cluster computing and solves analytical problems with large volumes of data and so is more suited to the cloud than SAS®9. Here are some CAS server highlights:

- Data is transmitted to and from the CAS server and stored data is secure.
- The CAS server is multi-threaded for high-performance analytics.
- The CAS server processes in-memory tables, which can come from a variety of sources.
- Open-source clients (Python, Lua, Java, R, REST API languages) connect to the CAS server.
- You can use the new high-performance SAS Viya procedures, SAS Procedures, the DATA step, and CAS actions to run your analysis in CAS.

Programmers can invoke CAS sessions from the client of choice, such as Jupyter Notebook or SAS Studio. Figure 1 CAS server architecture when running in massively parallel processing (MPP) mode:

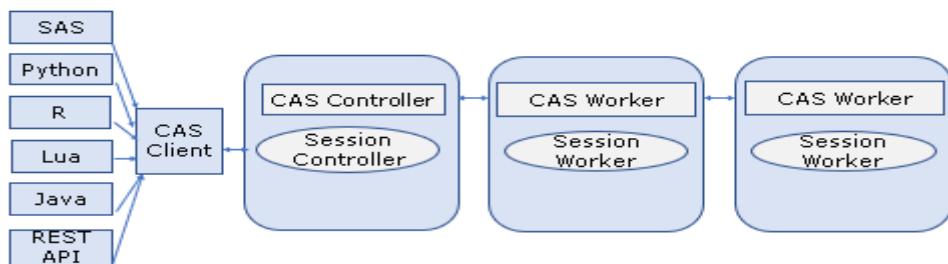


Figure 1. Cloud Analytic Service Concepts

THE CASUTIL PROCEDURE

The first step to running your analysis is to import your data into SAS, which you can do with various clients. In SAS Studio, you can easily manage data in CAS with the CASUTIL procedure, which manages tables in the CAS environment using a lightweight approach that feels familiar to SAS programmers. The CASUTIL procedure can load, drop, save, and describe CAS tables and works with tables in CAS, SAS

data sets in SAS libraries, and external files. You can also use other methods to load tables, such as the LoadTable CAS action with PROC CAS, the CAS Libname engine, and the DATA step, and a log file will be generated similar to Output 8.

Example 8

```
Cas casauto sessionopts=(caslib=casuser timeout=1800 locale="en_US"); ①
proc casutil;
  load data=sashelp.shoes casout="shoes"; ②
  promote casdata="shoes" incaslib="casuser"; ③
  save casdata="shoes" ; ④
  list tables;
quit;
```

- ① Added the CAS statement to start the CAS server with some session options.
- ② Used the CASUTIL statement using the LOAD statement to upload SAS data sets to the CAS server.
- ③ Used the PROMOTE statement to promote the CAS table with a global scope.
- ④ Used the SAVE statement to save the Shoes CAS table to the casuser caslib in the SASHDAT format.

```
81  cas;
NOTE: The session name identified with the SESSREF= SAS option is connected to Cloud Analytic Services. The default value
      SESSREF= is CASAUTO.
82
83  proc casutil;
NOTE: The UUID '7b4246ed-0986-6747-8f4e-d617a94e63bd' is connected using session CASAUTO.
84  /* droptable casdata="cars"; */
85  load data=sashelp.shoes casout="shoes";
NOTE: SASHELP.SHoes was successfully added to the "CASUSER(sasctp)" caslib as "SHOES".
86  PROMOTE CASDATA="shoes"INCASLIB="casuser";
NOTE: The meaning of an identifier after a quoted string might change in a future SAS release. Inserting white space between
      quoted string and the succeeding identifier is recommended.
NOTE: Cloud Analytic Services promoted table SHOES in caslib CASUSER(sasctp) to table shoes in caslib CASUSER(sasctp).
NOTE: The Cloud Analytic Services server processed the request in 0.001555 seconds.
87  SAVE CASDATA="shoes" ;
NOTE: Cloud Analytic Services saved the file shoes.sashdat in caslib CASUSER(sasctp).
NOTE: The Cloud Analytic Services server processed the request in 0.01128 seconds.
88  list tables;
```

Output 8. Log file that is generated

5. PROC REPORT

The REPORT procedure is the most powerful of the reporting procedures within SAS that contains a lot of functionality. Table 3 lists some of the main statements in PROC REPORT:

Statement	Description
Column	Specifies which columns are added to the report
Define	Defines how to use and display report items
Compute	Computes new columns or modify existing columns
Break	Adds a summary before/after an order or group variable to the left of the break variable
Rbreak	Adds a summary either at the beginning or end of a report or by-group

Table 3. PROC REPORT statements

THE COLUMN AND DEFINE STATEMENTS

The COLUMN statement defines which fields are included in the report as well as how they are arranged. The statement can include computed variables that are not a part of the input data. The COLUMN statement also defines the layout of the headers, such as adding spanning headers that span one or more columns. If no columns are added to the COLUMN statement, all columns are used by default.

The DEFINE statement describes how to use a column, specifies style attributes for a column, and supplies options for a column. Some usages that you can add for a column are Display, Order, Group, Computed, and Analysis. The usage applied to a column does everything from collapsing the values of a group to transposing or applying statistics to a column. The usages of Order and Group can also be used with the BREAK statement to generate summaries before or after similar values, or after the entire report. Example 9 demonstrates how to use these statements to generate output similar to Output 9.

Example 9

```
proc report data=sashelp.cars(where=(mpg_city > 25 and
Make in("Chevrolet", "Dodge"))) style(header)={backgroundcolor=black
Color=white};
column Make ('Luxury Car Prices' (MSRP Invoice )); ①
define make / order style(column)={backgroundcolor=lightgray}; ②
define model / left order style(column)={color=gray bordercolor=red};
define msrp / mean;
define MPG_Highway / noprint; ③
define mpg_city / mean format=8.0;
break after make / summarize style={bordertopwidth=5pt
borderbottomwidth=5pt bordercolor=black}; ④
rbreak after / summarize style={borderbottomwidth=5pt bordercolor=black};
run;
```

- ① The COLUMN statement includes the **Make**, **MSRP**, and **Invoice** fields along with a spanned header.
- ② The Color= and Backgroundcolor= style attributes are added for the **Make** and **Model** fields.
- ③ The NOPRINT option is added for the MPG_Highway column, which prevents it from being displayed.
- ④ The BREAK statement and RBREAK statement add a summary after each group and overall.

Luxury Car Prices		
Make	MSRP	Invoice
Chevrolet	\$11,690	\$10,965
	\$12,585	\$11,802
	\$14,610	\$13,697
	\$14,810	\$13,884
	\$16,385	\$15,357
Chevrolet	\$14,016	\$65,705
Dodge	\$13,670	\$12,849
	\$15,040	\$14,086
Dodge	\$14,355	\$26,935
	\$14,113	\$92,640

Output 9. COLUMN and DEFINE statements of PROC REPORT

COMPUTE BLOCK

Although extremely powerful, the compute block is also complex. The compute block can create new columns, inset lines of text, control placement of text, and use many of the same SAS Language statements that are used in the DATA step. The style generation capabilities of the compute block are limitless. Another powerful feature of the compute block is the CALL DEFINE statement, which can be

used to set or modify the style, format, or URL of a row, column, or individual cell. Example 10 shows using the compute block to add traffic lighting and add text to the summary line, as shown in Output 10.

Example 10

```
proc format;
  value price_mpg 0-<28="#3399FF"
    28-<30="#3333FF"
    34-<37="#9933FF"
    37-<40="#FF33FF"
    10000-<11000="#FF5733"
    11000-<13000="#FFBD33"
    13000-<15000="#DBFF33"
    15000-<18000="#75FF33";
  run;

proc report data=sashelp.cars(where=(mpg_city > 25 and
Make in("Chevrolet", "Dodge")) style(header)={backgroundcolor=black
Color=white};
column Make model ('Luxury Car Prices' (MSRP Invoice )) ('Miles Per
Gallon'(MPG_City MPG_Highway)) ;
define make / order style(column)={backgroundcolor=lightgray};
define msrp / mean;
define invoice / mean;
define MPG_Highway / mean;
define MPG_City / mean;
break after make / summarize style={bordertopwidth=5pt borderbottomwidth=5pt
};
rbreak after / summarize style={borderbottomwidth=5pt };

compute MPG_Highway; ①
array temp{*} msrp invoice MPG_City MPG_Highway; ②
  do i=1 to dim(temp);
    call define(cat(vname(temp{i}),"._mean"),"style",
"style={backgroundcolor=price.}"); ③
  end;
endcomp;

compute after;
  make="Total";
endcomp;
run;
```

① Added the COMPUTE statement for the MPG_Highway variable.

② Created an array for the numeric columns in the report.

③ Used the CALL DEFINE statement with the VNAME function to get the column name of the columns in the array concatenated with the Mean statistic. The BackgroundColor= style is added along with the format to add the traffic lighting.

		Luxury Car Prices		Miles Per Gallon	
Make	Model	MSRP	Invoice	MPG (City)	MPG (Highway)
Chevrolet	Aveo 4dr	\$11,690	\$10,965	28	34
	Aveo LS 4dr hatch	\$12,585	\$11,802	28	34
	Cavalier 2dr	\$14,610	\$13,697	26	37
	Cavalier 4dr	\$14,810	\$13,884	26	37
	Cavalier LS 2dr	\$16,385	\$15,357	26	37
Chevrolet		\$14,016	\$13,141	26.8	35.8
Dodge	Neon SE 4dr	\$13,670	\$12,849	29	36
	Neon SXT 4dr	\$15,040	\$14,086	29	36
Dodge		\$14,355	\$13,468	29	36
Total		\$14,113	\$13,234	27.428571	35.857143

Output 10. Compute block of PROC REPORT

6. GENERATING REST API REQUEST FROM SAS

The most popular type of API currently is REST API because of its flexibility. REST stands for Representational State Transfer and defines a set of functions like GET, PUT, DELETE, POST, etc. that clients can use to access server data. With Rest APIs, both the clients and servers exchange data using HTTP. Some of the benefits of a REST API include the following:

1. They are fast, and lightweight which is necessary for mobile apps, Internet of Things devices.
2. They are independent because the client and the server are separate.
3. They are both scalable and flexible.
4. They are stateless, which means that servers do not save client data between requests.

The method is the type of request that you send to the server. The four main resource methods that are associated with REST APIs are shown in Table 4:

GET	This method allows the server to find the data you requested and send it back to you.
PUT	This method updates an entry on the server
POST	This method permits the server to create a new entry on the server.
DELETE	This method allows the server to delete an entry.

Table 4. Common REST resource methods

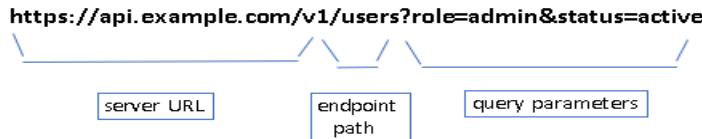
PROC HTTP AND REST

As mentioned earlier, HTTP is simply a way for clients and servers to communicate. Every time you enter a URL into a browser, a GET request runs and returns the information to the browser (client). When using PROC HTTP, the procedure is the client rather than the browser. This process is the same for any other client that uses REST, such as the cURL application, Postman, and Swagger, which are all good tools to test your API.

REST APIs that are available to the public always have accompanying documentation, which specifies how to implement the API and its components. The documentation also includes information about how to consume the API endpoints and authentication requirements. See the [HTTP Procedure](#) for the authentication methods available to PROC HTTP. Communicating with SharePoint via an API is a common task for SAS programmers which can be done using the PROC HTTP procedure.

THE LOCATION URL

The location of the server is identified with the URL, which is composed of the base endpoint, path query, and fragments as shown in Display 1:



Display 1. Location URL components

Example 11 uses the [Query parameters](#) API documentation for the OpenFDA API to determine how to structure the query and generates output similar to Output 11.

Example 11

```
filename resp temp;
proc http
  url="https://api.fda.gov/food/enforcement.json?search=state:'FL' ②
  &limit=10"
  method="get" ①
  out=resp;
run;

libname resp json; ③
proc print data=resp.results;
run;
```

① Specified the METHOD= option with the default GET.

② Specified the URL= with the endpoint, which is required to query the information along with the query parameters.

③ Used the JSON engine to read the data into SAS.

Obs	ordinal_root	ordinal_results	country	city	address_1	reason_for_recall	address_2	product_quantity	code_info	cent
1		1	1	United States	Davie	4131 SW 47th Ave Ste 1403	Recall initiated as a precautionary measure due to potential risk of product contamination with Burkholderia cepacia.	1,990 bottles	UPC No. 632687615989; Lot No. 30661601, Exp. Date 05/2018.	2016

Output 11. SAS data set generated from the response file

7. READING POPULAR FILE FORMATS WITH SAS

Every SAS programmer must read files of various formats and from an endless number of data sources. Data sources could involve database tables such as Oracle that have access engines, or those that do not but might require the use of an ODBC driver. Other data sources, such as Excel or SPSS files, are read using PROC IMPORT. This section focuses on some popular file formats and how to read or import them into SAS.

MICROSOFT EXCEL

The Excel XLSX file format is read into SAS using either the Excel LIBNAME engine, PROC IMPORT, or the Import Wizard with the SAS Display Manager (DMS) and SAS Enterprise Guide. This format requires a license for the SAS/ACCESS® Interface to PC files product. Example 12 demonstrates using the PROC IMPORT procedure to import an XLSX file and create SAS data set from it. The DBMS= option specifies the application format, such as XLSX, CSV, or SPSS.

Example 12

```
proc import datafile="C:\temp\heart.xlsx" ①
  out=work.heart
  dbms=xlsx ②
  replace;
  sheet="Sheet1"; ③
  getnames=yes;
run;
```

- ① Used PROC IMPORT to import the XLSX file using the DATAFILE= option
- ② Used the DBMS=XLSX option to import Excel 2007 and later files in the XLSX format.
- ③ The sheet name has been specified to read.

THE JSON AND XML FORMATS

Both JSON and XML are popular formats used for web-based exchanges. These file formats are used in a variety of ways, such as formats returned from a web service, used as configuration files, or simply as a method of transporting data. These files are more descriptive than some of the previous file formats used for transporting data, such as CSV. Display 2 shows an example of both a JSON and an XML file. The JSON format is made up of value pairs, whereas the XML is more structured and made up of tags:

JSON Example	XML Example
<pre>{ "SASJSONExport": "1.0 PRETTY", "SASTableData+CLASS": [{ "Name": "Alfred", "Sex": "M", "Age": 14, "Height": 69, "Weight": 112.5 }, { "Name": "William", "Sex": "M", "Age": 15, "Height": 66.5, "Weight": 112 }] }</pre>	<pre><?xml version="1.0" encoding="windows-1252" ?> <TABLE> <CLASS> <Name>Alfred</Name> <Sex>M</Sex> <Age>14</Age> <Height>69</Height> <Weight>112.5</Weight> </CLASS> <CLASS> <Name>Alice</Name> <Sex>F</Sex> <Age>13</Age> <Height>56.5</Height> <Weight>84</Weight> </CLASS> </TABLE></pre>

Display 2. Example of a JSON file and an XML file

The JSON file format can be read using the JSON LIBNAME engine in SAS, whereas the XML file format can be read using the XMLV2 engine. The JSON and XMLV2 LIBNAME engines use some defaults, such as numbers are always read into SAS as a numeric data type. To modify any of the metadata such as the data types, column lengths, formats/informats, you can generate a map file (a SAS specific file). You can generate the map file dynamically using the XMLMAP= option with the AUTOMAP=REPLACE. You can also use the XML Mapper, which is a standalone Java application, to interactively generate map files for XML. The map file can describe how to make rows and columns from the data. Example 13 demonstrates reading data using the JSON LIBNAME engine.

Example 13

```
proc json out="c:\temp\class.json" pretty; ①
  export sashelp.class;
run;
Libname input json "c:\temp\class.json" map="c:\temp\class.map"
  automap=create; ②
proc copy in=input out=work;
run;
```

- ① Used PROC JSON to generate a JSON file from the SASHELP.CARS data set.
- ② Used the JSON LIBNAME engine to read the JSON file and also generate a map file dynamically.

SAS TRANSPORT FILES

Several types of SAS transport files, such as the V5 and V8 formatted files and transport files, are created with PROC CPOR. The V5 transport files are created and read using the XPORT LIBNAME engine. V8 transport files are created and read using the %LOC2XPT and %XPT2LOC macros respectively. The V5 and V8 transport file format only move SAS tables; however, PROC CPOR can move entire libraries and is used frequently to migrate or backup libraries. PROC CIMPORT restores the transport files created with PROC CPOR.

If you want to know which format was used, you can view the file in a text editor and see how the layout is different, as illustrated in Example 14. The left side shows Version 8 transport files created using the %LOC2XPT macro and read using the %xpt2loc macro. The right side shows PROC CPOR and PROC CIMPORT being used to create and restore SAS transport files.

Example 14

```
%loc2xpt(libref=sashelp,
      memlist=class,
      filespec='c:\trans.v9xpt')

%xpt2loc(libref=work,
      memlist=class,
      filespec='c:\trans.v9xpt');

filename trans 'c:\transport.stc';
proc cport file=trans lib=sashelp;
select class;
run;

proc cimport
infile="c:\transport.stc" lib=work;
run;
```

USING THE ZIP ACCESS METHOD IN SAS

The ability to read ZIP files has been simplified by the addition of the ZIP access method. Prior to the ZIP access method, reading ZIP files typically involved interacting with the operating system. The ZIP access method reads and writes only files created with the WinZip file compression tool. Example 15 shows a practical use of the ZIP access method to extract files using the DATA step:

Example 15

```
ods package(newzip) open nopf; ①
ods package(newzip) add file="c:\temp.csv";
ods package(newzip) publish archive
  properties(
    archive_name="cars.zip"
    archive_path="c:\output"
  );
ods package(newzip) close;

filename foo "c:\output\cars.zip";
data _null_; ②
infile foo(temp.csv);
input a $80. ;
put _infile_;
run;
```

- ① Used the ODS Package to create a ZIP file from the temp.csv file.
- ② Used the DATA step to read the temp.csv file from the ZIP archive.

8. AUTOMATING SAS CODE USING MACROS

SAS macros are a tool used for text substitution, which is used to automate SAS code. Macros are useful when you need to generate repetitive syntax. Macros are a broad topic, so this paper focuses on key concepts. Some features in the macro facility include creating macro variables, generating reusable code, dynamic macro variable generation, and macro variable scope.

CREATING MACRO VARIABLES

Table 5 shows a few methods for creating macro variables using the SAS language. It is important to know the scope of the macro variables, such as when and where they are available. Macro variables are added to either the global or local symbol tables. In general, macro variables that are generated inside of a macro are local to the macro. Use the %PUT _GLOBAL_ to see what the global symbol table contains.

%LET Statement	The %LET statement assigns text to a macro variable
CALL SYMPUT Statement	Assigns a DATA step variable values to a macro variable
INTO: Clause	Creates one or more macro variables based on a SELECT statement
Macro Parameters	Creates local macro variables whose values are specified when invoked
%GLOBAL statement	Creates global scope macro variables available for the duration of the session
%LOCAL Statement	Creates local scope macro variables available for the duration of the macro

Table 5. Creating macro variables

Example 16 demonstrates how to generate reusable code that can create output such as Output 12.

Example 16

```
%let make=Audi; ①
%let type=Sports;

proc print data=sashelp.cars;
where make="&make" and ②
      type="&type";
title "Report Generated
&sysdate";
run;
```

Report Generated 13SEP23					
Obs	Make	Model	MSRP	Invoice	
21	Audi	RS 6 4dr	\$84,600	\$76,417	
22	Audi	TT 1.8 convertible 2dr (coupe)	\$35,940	\$32,512	
23	Audi	TT 1.8 Quattro 2dr (convertible)	\$37,390	\$33,891	
24	Audi	TT 3.2 coupe 2dr (convertible)	\$40,590	\$36,739	

Output 12. SAS macro and reusable code

- ① Created reusable code by setting macro variables at the beginning of the program using the %LET statement.
- ② Used the WHERE statement with the Make and Model macro variables to subset the code.

GENERATING REPETITIVE CODE

You can generate repetitive code several ways using the macro language. One method uses the INTO clause within the PROC SQL procedure by creating a series of macro variables and resolving them within a macro %DO loop.

Example 17 creates a macro variable list from the distinct values of the variable TYPE in PROC SQL using the INTO clause. The TOTAL macro variable is created getting the count of unique values of the variable count.

Example 17

```
proc sql;
  select count(distinct type) into:total ①
```

```

from sashelp.cars;
select distinct(type) into:typel-:type%trim(%left(&total)) ②
from sashelp.cars;
quit;

%macro cars;
%do i=1 %to &total; ③
  proc print data=sashelp.cars;
  where type= "&&type&i";
  run;
%end;
%mend;
%cars

```

- ① Macro variable Total generated from the distinct values of the variable type using the INTO clause.
- ② A macro variable list with all of the distinct values of the variable type is generated dynamically using the INTO clause of PROC SQL.
- ③ Used the Macro %DO loop to loop through all the macro variables, creating a new table for each distinct type.

THE CALL EXECUTE ROUTINE

Using the CALL EXECUTE routine provides the ability to pass values directly from the DATA step to the macro as a parameter. This technique is valuable and is used in multiple examples in this document.

Example 18

```

%macro cars(type); ①
  proc print data=cars;
  where type="&type";
  run;
%mend;

proc sort data=sashelp.cars out=cars;
by type;
run;

data _null_; ②
  set cars;
  by type;
  if first.type then call execute('%cars('||type||')');
run;

```

- ① The macro is defined with a single positional parameter named TYPE.
- ② The invocation of the macro CARS passes each distinct value of the variable TYPE as a parameter.

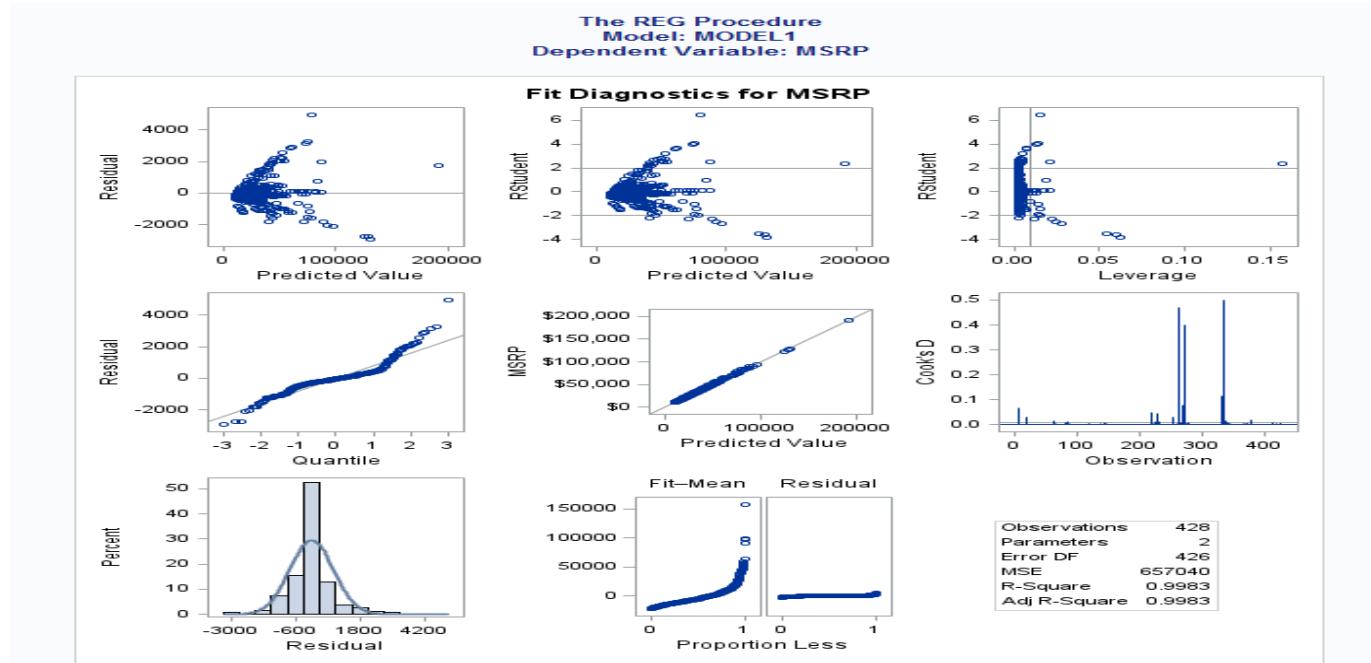
9. CREATING GRAPHICS IN SAS

ODS Graphics are template-based graphics that were introduced in SAS®9. Before ODS Graphics, you had to use SAS/GRAPH® to generate graphics within SAS, which was device based. Some of the highlights of ODS Graphics are the innate ability to interpret the data and to generate graphics on the fly from most statistical procedures. Graphics can be generated from statistical procedures if the ODS Graphics preference is turned on, which is the default. ODS Graphics are made up of the Graphics Template Language (GTL) and the Statistical Graphics (SG) procedures. Example 19 demonstrates using

PROC REG to generate the graphics shown in Output 13.

Example 19

```
Ods select FitPlot;
proc reg data = sashelp.cars;
  model msrp = invoice;
run;
```



Output 13. Graphics created from PROC REG

THE STATISTICAL GRAPHICS PROCEDURES

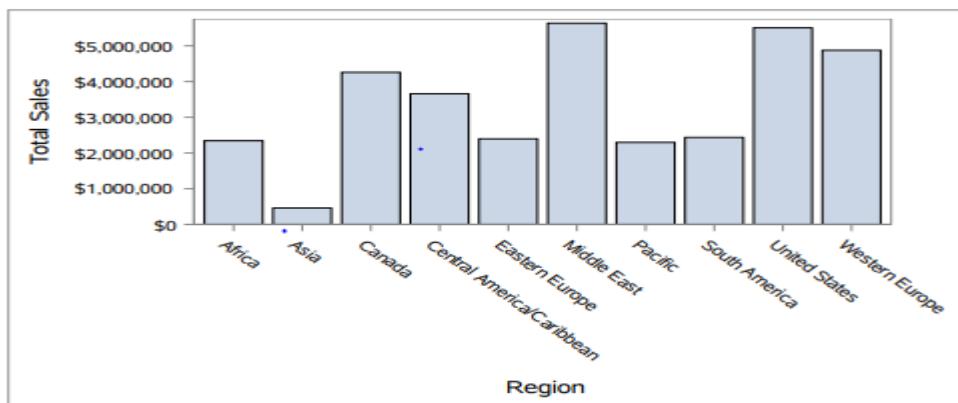
The SG procedures are an extension of the ODS Graphics framework, providing access to the GTL in the familiar syntax of the SAS/GRAPH procedures. The SG procedures include PROC SGLOT, PROC SGANEL, and PROC SGSCATTER. There is also PROC SGRENDER, which renders graphics generated using the GTL. This paper examines some basic functionality of ODS Graphics.

The SGLOT procedure contains many statements that can be used based on the type of graph required. For example, the SCATTER statement is used to create a scatter plot. There are other statements, such as the HISTOGRAM statement, which generates histograms; the DENSITY statement, which generates density plots; and the VBAR and HBAR statements generate vertical and horizontal bar charts. A bar chart plots numeric values for levels of a categorical feature as bars. Bar charts are used when you want to show a distribution of data points or perform a comparison of metric values across different subgroups of your data. The primary variable of a bar chart is its categorical variable. Example 19 uses bar charts to view the SASHELP.SHOES table, as shown in Output 14.

Example 19

```
ods graphics / height=3in width=5in imagefmt=jpeg ; ①
PROC SGLOT DATA = sashelp.shoes ;
  VBAR Region / stat=sum response=sales; ②
run;
```

- ① Added the height, width, image format, and image name using the ODS Graphics statements.
- ② Used the VBAR statement to look at sales by region within the SASHELP.SHOES table.



Output 14. PROC SGPlot using the VBAR statement.

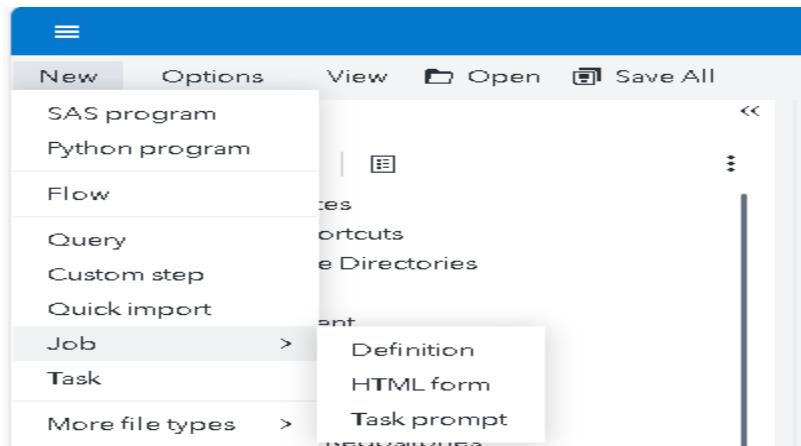
10. SCHEDULING SAS VIYA JOBS IN SAS STUDIO

One of the many powerful features of the SAS Viya platform is the ability to create and schedule SAS Viya jobs, which consist of a program and a definition that executes on the compute server. The definition includes items such as the author, the job name, a creation date, and time. SAS Viya jobs are similar to stored processes on SAS®9; however, SAS Viya does not have a metadata server. Some commonalities are that both enable you to stream output and create amazing web applications.

CREATING SAS VIYA JOBS

To create a SAS Viya job, from the menu items at the top of the window, select **New > Job**, which expands to display the values Definition, HTML form, and Task prompt as seen in Display 3.

- *Definition* is the SAS code which is executed on the compute server.
- *HTML form* builds dynamic web content that gives you the ability to select parameters that are used to determine the results.
- *Task prompt* takes advantage of the SAS Studio Common Task Model and is flexible in the type of prompts generated, including the dynamic content using data value.



Display 3. Creating Viya Jobs

See the [Create a SAS Viya Job with a prompt using HTML Prompts](#) article for more detail. There are many benefits to adding a dynamic front end to the job. When the job is executed, the form is displayed first, which uses the selected parameters to add to the job.

SAS Viya jobs can be executed using a variety of methods, such as SAS Studio, command-line interface (CLI), REST API, Job Execution Web app, and a few others. Once the job is created, a URL can be sent to others who can also execute the job if they are in the authenticated group. The SAS Viya platform adds the ability to add guest access similar to a feature in the SAS®9 stored processes. To get the link for the job, go to job properties on the right side of the window and click the item to show the URL on the bottom right.

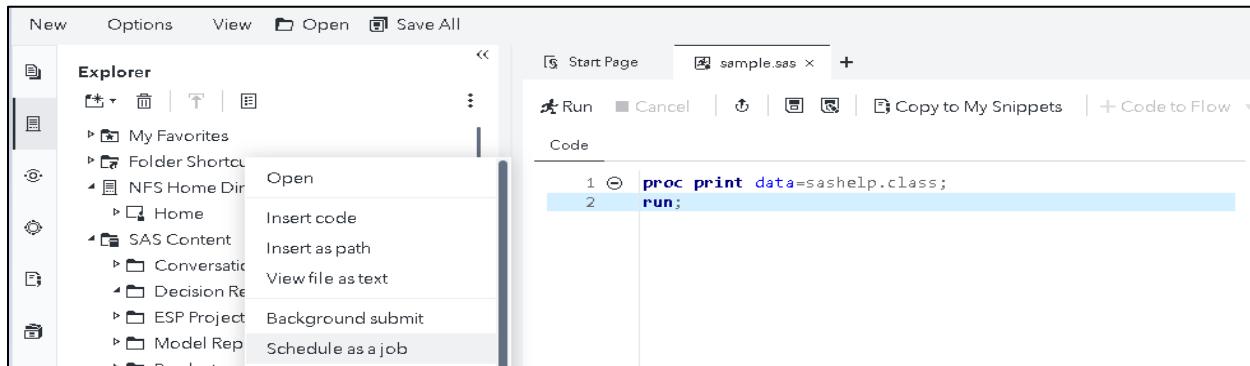
SCHEDULING SAS JOBS

There are a few methods to schedule capabilities for SAS Viya jobs, such as SAS Studio, SAS Job Execution Web App, SAS Environment Manager, and the CLI. You can choose different methods based on your needs. Each of the methods mentioned uses the Job Execution Microservice or the scheduling microservice. This paper describes using SAS Studio for scheduling.

SAS Studio

When you schedule a job, SAS Studio creates a snapshot of the program. Any subsequent changes to the program are not reflected in the scheduled job. To incorporate any program updates, you must schedule a new job by completing these steps:

1. In Explorer, right-click the name of the job definition, and select **Schedule as a job** as shown in Display 4.
2. In the New Trigger window, specify how frequently to run the job. You can also specify the time at which the job should run and the start and end dates for running the job.



Display 4. Scheduling SAS Viya Jobs using SAS Studio

CONCLUSION

You have finally made it to the end of the top ten features from the perspective of a Technical Support guy! The list includes impactful reporting features that allow you to generate presentational quality Excel reports and email them using SAS. It also includes automating SAS code using the SAS macro language in addition to the integration of open-source languages. Finally, managing data in the CAS server or scheduling SAS Viya jobs adds additional impact when working within the SAS Viya platform. Hopefully, some of the features mentioned in this paper will be helpful in your daily work as an impactful SAS programmer.

REFERENCES

OpenFDA. 2023. "Query parameters." Available at open.fda.gov/apis/query-parameters/.

OpenPyXL. 2023. "openpyxl - A Python library to read/write Excel 2010 xlsx/xlsm files." Available at openpyxl.readthedocs.io/en/stable/.

SAS Communities Library. 2022. "Create a SAS Viya job with a prompt using HTML Prompts." Available at communities.sas.com/t5/SAS-Communities-Library/Create-a-SAS-Viya-Job-with-a-prompt-using-HTML-Prompts/ta-p/805091.

SAS Institute Inc. 2022. "Client Authentication Using an Authinfo File." *SAS® Viya® Programming Documentation*. Cary, NC: SAS Institute Inc. Available at go.documentation.sas.com/doc/en/pgmsascdc/v_017/authinfo/n0xo6z7e98y63dn1fj0g912j7oyq.htm.

SAS Institute Inc. 2023. "HTTP Procedure." *SAS® 9.4 and SAS® Viya® 3.5 Programming Documentation*. Cary, NC: SAS Institute Inc. Available at go.documentation.sas.com/doc/en/pgmsascdc/v_017/authinfo/n0xo6z7e98y63dn1fj0g912j7oyq.htm.

SAS Institute Inc. 2018. "SASPy Configuration." Available at sassoftware.github.io/saspy/configuration.html.

CONTACT INFORMATION

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

Chevell Parker
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Email: support@sas.com
Web: support.sas.com/en/support-home.html

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.