

SAS4594-2020**Building an Expert's Toolbox: Essential Tools for Generating the Perfect Microsoft Excel Worksheet**

Parker, Chevell, SAS Institute Inc.

ABSTRACT

When you have a home building or renovation project to accomplish, you need expert tools for the job. The same is true when you want to build (create) or modify (renovate) Microsoft Excel worksheets. You need a variety of expert tools in your SAS® software toolbox to accomplish these tasks. You have a choice of many tools that enable you to create and fully customize your worksheets. For example, you can use the SAS® Output Delivery System (ODS) Excel destination and the SAS EXPORT procedure. But you can also complement the standard tools with more specialized ones (for example SAS macros and the Python open-source language) to further extend the capabilities of your worksheets.

This paper explains how to use all of these tools to create fully functional Microsoft Excel worksheets. The discussion is divided into two main sections. The first section explains how to generate Excel worksheets and perform various tasks in SAS and SAS® Viya®. For each task in this section, the paper demonstrates how to accomplish the task by using current functionality (for example, the ODS Excel destination, PROC REPORT, and so on) that is available in SAS and SAS Viya. This section also explains how you can enhance that functionality by using the custom %Excel_Enhance macro. The second section illustrates how you can further extend worksheet functionality in all environments by using the open-source tools Python and Java.

INTRODUCTION

Microsoft Excel is a spreadsheet application that is used universally in all industries and job categories. People use Excel for tasks ranging from adding formulas to a worksheet and recording expenditures and income to planning budgets. You can use SAS® software to fully automate the process of generating and customizing Excel worksheets. This paper discusses methods for generating and enhancing worksheets from both the SAS and SAS Viya platforms. For both platforms, the paper discusses in detail how you can use the SAS Output Delivery System (ODS) Excel destination and PROC EXPORT to generate worksheets. In addition, the discussion covers tools that you can use to enhance the output that the ODS Excel destination and PROC EXPORT generate. This paper also discusses ways to automate worksheets and extend worksheet functionality by using Python and Java open-source software

GENERATING EXCEL FILES USING SAS® 9.4 AND SAS® VIYA®

You can generate Excel workbooks and worksheets in various ways with both SAS 9.4 and SAS Viya. This paper discusses these methods in detail. It also explains the correct tools to use to generate and enhance worksheets and workbooks. The next section discusses how to generate worksheets using SAS Viya. Then, the next major section explains methods that work with both SAS 9.4 and SAS Viya.

CREATING WORKSHEETS USING SAS® VIYA®

SAS Viya is a cloud-enabled, in-memory analytics engine that provides quick, accurate, and reliable analytical insights. One of the huge benefits of SAS Viya is that you can run your analysis in the cloud using the Cloud Analytic Server (CAS). The CAS server uses high-

performance, multi-threaded analytic code to rapidly process requests against data of any size. SAS Viya also enables you to run your analysis using these open-source programming languages: Python, R, Java, Lua, and the REST API. Programming languages such Python and R connect to the CAS server to run analysis. However, SAS® Studio has the ability to run in SAS or to connect to the CAS server. Tables that you create in CAS are stored in-memory (in CAS libraries, or caslib names) in the SASHDAT format.

To save an in-memory CAS table to another format (for example, XLSX or XLS), you can use either CASL programming language (via the CAS procedure), the CASUTIL procedure, or PROC EXPORT. Both PROC CAS and PROC CASUTIL enable you to run actions on the CAS server. PROC CASUTIL enables you to load, drop, save, and describe tables. PROC EXPORT uses the CAS engine when you qualify it with a caslib. By doing so, you can use PROC EXPORT to export CAS tables. In addition, you can use PROC EXPORT with a SAS data set that uses the V9 engine to read the data that is exported or saved to an XLSX file. The following example illustrates this ability by loading the SASHELP.ORSALES data set into memory, using a caslib that is named CASUSER.

Note: You need SAS® Data Connector to PC Files in order to export data to Excel.

Example 1

```
cas mySess sessopts=(caslib=casuser locale="en_US");

proc casutil outcaslib="casuser";
  load data=sashelp.orsales casout="sales";
  save casdata="sales" casout="sales.xlsx";
  list files;
quit;
```

In This Example

- The CAS session is established with the session that is named CASAUTOS.
- The OUTCASLIB=option determines the caslib to which to write tables.
- The LOAD statement in PROC CASUTIL loads data from the client to the CASUSER caslib.
- The SAVE statement saves the in-memory table to an Excel file that is named SALES.XLSX

Output

	A	B	C	D	E	F	G	H	I
1	Year	Quarter	Product_Line	Product_Category	Product_Group	Quantity	Profit	Total_Retail_Price	
2	1999	1999Q1	Children	Children Sports	A-Team, Kids	286	4980.15	8990.9	
3	1999	1999Q1	Children	Children Sports	Bathing Suits, Kids	98	1479.95	2560.4	
4	1999	1999Q1	Children	Children Sports	Eclipse, Kid's Clothes	588	9348.95	18768.8	
5	1999	1999Q1	Children	Children Sports	Eclipse, Kid's Shoes	334	7136.8	14337.2	
6	1999	1999Q1	Children	Children Sports	Lucky Guy, Kids	303	7163	12996.2	
7	1999	1999Q1	Children	Children Sports	N.D. Gear, Kids	755	19153.05	34250.5	

Output 1. Output Created by Using the CASUTIL Procedure and the SAVE Statement

CREATING EXCEL WORKSHEETS WITH METHODS THAT WORK IN BOTH SAS® 9.4 AND SAS® VIYA

With SAS 9.4 and SAS Viya, you can use various methods to create Excel worksheets that can generate Excel files. Those methods include the use of the following tools as well as others:

- ODS Excel destination: This destination generates presentation-ready output.
- EXPORT procedure: This procedure enables you to update worksheets. PROC EXPORT also handles larger files very efficiently. This procedure runs either on the CAS server or in SAS, depending on whether you use a CAS table or a SAS data set.

Note: You can use both the ODS Excel destination and PROC EXPORT in SAS®9 and SAS Viya. Currently, the Excel destination is not CAS enabled. Therefore, the ODS destination and your data are run in SAS rather than on the CAS server.

You do not use a Philips screwdriver to remove a hollow-point nail because you would be using the wrong tool for the job. The task would be inefficient, tedious, and frustrating. In the same way, you need to use the correct tool for the job when you construct a worksheet. Table 1 lists several tools that are right for the job when you want to generate and modify Excel worksheets. The table correlates these tools with their strengths and weaknesses.

Tools	Description	Limitations
ODS Excel	Creates XLSX files from SAS DATA steps and procedures.	Does not have update ability. It generates a new worksheet each time.
PROC EXPORT	Creates XLSX files from SAS data sets.	Does not have styling capabilities.
DDE (does not run on SAS Viya)	Uses the DDE-Triplet specification, which writes to Excel worksheets.	Only works in Microsoft Windows environments, and DDE is no longer supported by Microsoft.
%Excel_Enhance Macro	Adds functionality to and post-processes Excel files.	Requires Microsoft Excel in order to execute.
TableEditor Tagset	Adds pivot tables and Excel output to a worksheet.	Requires Microsoft Excel in order to execute.
Open Source (Python)	Modifies or creates XLSX files.	Adds an extra step.

Table 1. Tools That Enable You to Create or Enhance Excel Worksheets

MODIFYING WORKSHEETS USING THE ODS EXCEL DESTINATION

This section demonstrates how to create a worksheet from start to finish by using both the ODS Excel destination and the %Excel_Enhance macro. The macro enables you to include additional functionality that generates Microsoft VBScript.

Note: The %Excel_Enhance macro is not shipped with SAS. Therefore, you need to include it or call it from a macro library. To download the macro, click [here](#).

The following sections explain how to perform the following enhancements to a worksheet:

- include a banner
- add summary totals
- add a summary worksheet
- filtering within worksheets
- adding text with the Excel destination

INCLUDING A BANNER

You can add a banner image to a worksheet with the Excel destination by using the SAS Report Writing Interface (RWI). The RWI can add a logo only to cells beginning in column A1, and it cannot control the size of the image.. Tools such as the %Excel_Enhance macro and Python open-source language (discussed later in this paper) enable you to place images

anywhere on the worksheet, but you must post-process such files. For more information, see the blog [Tips for Using the ODS Excel Destination](#). (Parker, 2017)

The following example illustrates how to add a banner image to a worksheet.

Example 2

```
ods excel file="c:\temp\image.xlsx";
data _null_;
  declare odsout obj();
  obj.image(file:"C:\Users\sasctp\pictures\banner_bi.jpg");

run;
proc print data=sashelp.heart(obs=5);
  var Sex Height Weight Diastolic Systolic Smoking Cholesterol;
run;
ods excel close;
```

In This Example

- The DECLARE statement declares the object OBJ().
- The image method (OBJ.IMAGE) is used within RWI to add the banner.
- PROC PRINT displays the SASHELP.HEART data sets.

Output

Obs	Sex	Height	Weight	Diastolic	Systolic	Smoking	Cholesterol
1	Female	62.50	140	78	124	0	
2	Female	59.75	194	92	144	0	181
3	Female	62.25	132	90	170	10	250
4	Female	65.75	158	80	108	0	249

Output 2. Adding a Banner by Using the Report Writing Interface

ADDING SUMMARY TOTALS TO THE WORKSHEET

Adding summary totals to a worksheet enables decisionmakers to view an overall snapshot of the data initially. But it also offers them the ability to drill down into the data dynamically, much like a pivot table. Summary totals (which include the total sales, average sales, and sales count [number of items visible]) are displayed at the top of a report, and you can modify them based on filtered values in the table.

In the following example, filters are added to all of the categorical columns. This program uses the Excel application's [SUBTOTAL\(\) function](#) in the Report Writing Interface.

Example 3

```
ods escapechar="^";
ods excel file="filter.xlsx" options(sheet_interval="none"
                                   autofilter="1-4"
                                   start_at="3,3");

proc odstext;
  p 'Total Sales' / style={color=red};;
  p "=Text(SUBTOTAL(9,G10:G9999),"$#,###.00")";
  p " ";
  p 'Average Sales' / style={color=purple};
  p "=Text(SUBTOTAL(1,G10:G9999),"$#,###.00")";
  p " ";
```

(code continued)

```

p 'Items Visible' / style={color=blue};
p "=SUBTOTAL(2,G10:G9999)";
run;

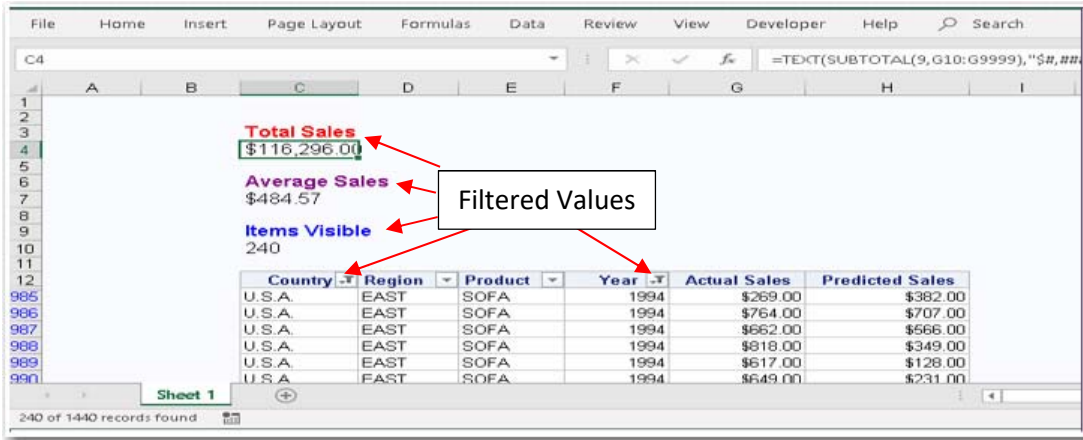
proc report data=sashelp.prdsale;
  column COUNTRY REGION PRODUCT YEAR ACTUAL PREDICT;
run;
ods excel close;

```

In This Example

- The Excel option AUTOFILTER= is applied to the first four columns of the report.
- The Excel SUBTOTAL() function, used within PROC ODSTEXT, defines the statistics to use and the ranges for those statistics.
- The TEXT() function nests the SUBTOTAL() function within the PROC ODSTEXT, which provides the formatting (the names of the function or statistic, along with the actual statistic value).

Output



Output 3. Adding a Dynamic Summary Table to a Worksheet

ADDING A SUMMARY WORKSHEET

Similar to the previous example, summary worksheets provide a quick snapshot of the data. They also enable you to drill down to supporting details or other data. The supporting details or other data can be in the same workbook, in an external workbook, on a web page, in a PDF file, or in a Word file.

The following table illustrates example syntax of the various types of links you can use in a worksheet for drill-down information.

Syntax for Drill-Down Links That You Can Use in Excel	Results
#sheet1!A1	Link to a worksheet within a workbook that begins in cell A1
= 'Sales Report'!A10	Link to a worksheet that includes spaces starting at cell A10
D:\Reports\[Sales.xlsx]Jan!B2: B5	Link to an external worksheet
[Sales.xlsx]Jan!jan	Link to a specific named range
filename.html# name	Link to a named destination (anchor) of on web page
Mailto: john.doe@sas.com	Link to a particular email address

Table 2. Syntax for Generating Links in a Microsoft Excel Worksheet

Adding Hyperlinks and Drill-Down Capability in Microsoft Excel Worksheets

Drill-down capability in worksheets enables you to access more detailed data than is available visibly on your worksheet. As mentioned previously, that information can be internal to the worksheet or in external locations. You can also include hyperlinks directly in your worksheet to access supporting data.

Example 4 demonstrates adding hyperlinks and drill-down functionality. You can add hyperlinks using several methods:

- the LINK= option in TITLE and FOOTNOTE statements
- the Excel =HYPERLINK function within SAS statements (for example, ODS TEXT=, PROC ODSTEXT, TITLE, LABEL, and FOOTNOTE), in the Report Writing Interface, or in a DATA step
- either the URL style attribute or the URL method within the SAS REPORT procedure's CALL DEFINE statement (dynamically builds links to various files)

The example demonstrates how to include hyperlinks via the =HYPERLINK Excel function by using the ODS TEXT= statement because the URL= attribute is not supported in the ODS TEXT= statement.

In addition, Example 4 uses PROC REPORT with a compute block to generate drill-down functionality for each unique value of the products.

Example 4

```
ods escapechar="^";
title "Summary of Financials for year 2019";
ods excel file="c:\temp\drill.xlsx" options(sheet_interval="none"
                                           sheet_name="Summary"
                                           embedded_titles="yes"
                                           start_at="2,2");

ods text='=hyperlink("mailto:joe.doe@sas.com","Hyperlink to Email")';
ods text='=hyperlink("#Beds!a1","Hyperlink to Embedded Worksheet")';
ods text='=hyperlink("c:\[drill.xlsx]#Beds!a1","Hyperlink to External
Worksheet")';
ods text='=hyperlink("http://www.sas.com","Hyperlink to URL")';
ods text='=hyperlink("file://c:\temp.pdf","Hyperlink to PDF File")';
title "Summary Worksheet";
proc report data=sashelp.prdsale spanrows
           style(report)={pretext=" "};
  column product country actual predict;
  define country / group;
  define product / group style(column)={textdecoration=underline
                                       color=blue};
  compute product;
    if product="SOFA" then call define(_col_,"url","Sofa");
    else if product="CHAIR" then call
      define(_col_,"url","#Chair!a1");
    else if product="DESK" then call define(_col_,"url","#Desk!a1");
    else if product="TABLE" then call define(_col_,"url","#Table!a1");
    else if product="BED" then call define(_col_,"url","#bed!a1");
  endcomp;
run;
```

(code continued)


```
ods excel options(sheet_name="#byval(product)"
                 sheet_interval="bygroup" );
title link="#summary!a1" "Back to summary";
proc report data=sashelp.prdsale;
  column product country region actual predict;
  by product;
run;
ods excel close;
```

In This Example

- The ODS TEXT= statement adds hyperlinks to the various file formats. The links that are generated open an email message, link to internal and external worksheets, open web pages, and open a PDF file.
- PROC REPORT is used with the CALL DEFINE statement and the URL method to create drill-downs to the various worksheets.
- The second PROC REPORT step (which includes the SHEETNAME="#BYVAL(*product*)" option) generates comparable like-named worksheets in the same workbook to which you can drill down. This PROC REPORT step generates detail data from the BY-group, which is the target of the drill-down information. The data set is already sorted by product.

Output

Product	Country	Actual Sales	Predicted Sales
BED	CANADA	\$47,729.00	\$44,215.00
	GERMANY	\$46,134.00	\$43,796.00
	U.S.A.	\$48,174.00	\$49,856.00
CHAIR	CANADA	\$50,239.00	\$46,796.00
	GERMANY	\$47,105.00	\$44,069.00
	U.S.A.	\$50,936.00	\$45,245.00
	CANADA	\$52,187.00	\$49,393.00
	GERMANY	\$46,503.00	\$44,539.00

Output 4. Adding a Summary Sheet to a Workbook

FILTERING WITHIN EXCEL WORKSHEETS

Adding filters to table enables you to subset on(or, filter) values within a column. To apply filters to columns with the ODS Excel destination, you need to use the AUTOFILTER= suboption. You can add filters to all columns, to the default column, to a specific column, or to a range of columns in sequence (for example, columns 1-3), as illustrated in [Output 4](#).

Example 5

```
ods escapechar="^";
ods excel file="filter.xlsx" options(autofilter="all");
proc report data=sashelp.orsales(obs=3);
  column Year Product_Line Profit;
run;

proc report data=sashelp.orsales(obs=3)
  style(header)={posttext="^{nbspspace 4}" asis=on};
  column Year Product_Line Profit;
run;
ods excel close;
```

In This Example

- The AUTOFILTER= option adds filters to the table headers.
- Notice that the filter icon (the down arrow circled in red in Report 1 of Output 5) might hide part of the header text. You can avoid that situation by using the POSTTEXT= attribute with the NBSpace function, as is done in Example 5. In this example, that combination adds four spaces to the right of the value. As a result, the header text is no longer hidden (as shown in the second report of Output 5).

Output

1	A	B	C
2	1999	Children	4980.15
3	1999	Children	1479.95
4	1999	Children	9348.95
5			

1	A	B	C
2	1999	Children	4980.15
3	1999	Children	1479.95
4	1999	Children	9348.95
5			

Output 5. Adding Filters with the AUTOFILTER= Option and Fixing Hidden Header Text with the POSTTEXT= Attribute

ADVANCED FILTERING

Currently, the ODS Excel destination provides basic filtering by using the AUTOFILTER= options. However, you can achieve some advanced filtering capabilities by using the ODS Excel destination in conjunction with the %Excel_Enhance macro. Doing so offers the following features:

- the ability to add preselected filters. *Preselected filters* are values that you supply and that are filtered when the table is displayed.
- the ability to select the top 10% of values in a column.
- the ability to select the bottom 10% of values in a column.

Example 6 adds preselected filters by passing the FILTER= parameter to the macro along with the sheet name, the column to which you want to add the filter. The FILTER= parameter has a three-part argument. Those parts are the sheet name, the column name, and value or expression on which to filter the column.

```
FILTER="sheet-name, column-name, value-or-expression"
```

The values that are available for this parameter are documented in [this %Excel_Enhance macro download](#).

Example 6

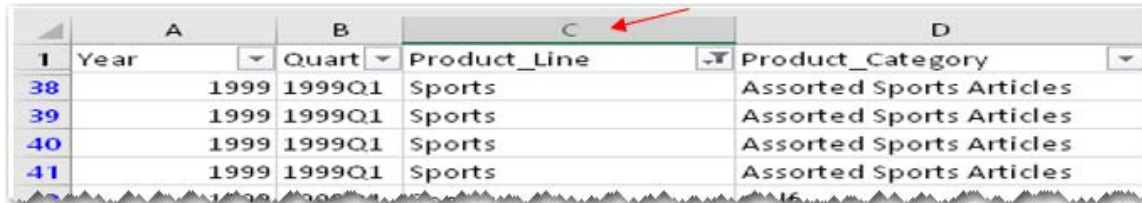
```
ods excel file="c:\temp\summary.xlsx" options(sheet_name="sales");
  proc print data=sashelp.orsales;
    run;
ods excel close;

/* Modify the file to create the predefined filters. */
%excel_enhance(open_workbook=c:\temp\summary.xlsx,
  filter=%str(sales,4,Sports),
  create_workbook=c:\temp\summary_update1.xlsx);
```


In This Example

- The ODS Excel destination creates a sample workbook named `summay.xlsx`.
- The `%Excel_Enhance` macro passes the worksheet name, the column name that should be added to the filter, and the value or expression on which to filter.

Output



	A	B	C	D
1	Year	Quart	Product_Line	Product_Category
38	1999	1999Q1	Sports	Assorted Sports Articles
39	1999	1999Q1	Sports	Assorted Sports Articles
40	1999	1999Q1	Sports	Assorted Sports Articles
41	1999	1999Q1	Sports	Assorted Sports Articles

Output 6. Adding a Predefined Filter to the Product-Line Column of a Worksheet

USING TEXT WITH THE ODS EXCEL DESTINATION

You can use the ODS Excel destination with the following methods to add text to output:

- The `ODS TEXT=` statement enables you to add text quickly. However, it does not generate an output object, nor does it have the advanced capabilities of `PROC ODS TEXT`.
- `PROC ODSTEXT` or `PROC ODSLST`, along with the Report Writing Interface, generates output objects. When you combine these tools with the `SHEET_INTERVAL=` option, they control whether text is added to a new worksheet.
- The `PRETEXT=` and `POSTTEXT=` style attributes enable you to add text strings before or after cell or table values.

This section illustrates how to use these tools to customize your output.

Using PROC ODSTEXT and the ODS Excel Destination to Generate a Customized Table of Contents

`PROC ODSTEXT` can generate custom lists and text, among other things. You can also use this procedure to add Excel functions to a worksheet.

In Example 7, `PROC ODSTEXT` generates a customized table of contents by creating a hyperlink with the unique data value from the variable `Product_line` from the data set.

Example 7

```
proc sort data=sashelp.orsales out=temp nodupkey;
  by product_line;
run;
ods escapechar="^";
ods excel file="c:\temp\contents.xlsx" options(sheet_name="Summary"
                                              embedded_titles="yes");

proc odstext data=temp;
  p "=hyperlink('||'[contents.xlsx]|| product_line ||'!A1'||',' ||
      quote("Product: " ||product_line)||")";
run;

ods excel options(sheet_name="#byval(product_line)");
proc print data=temp;
  by product_line;
run;
ods excel close;
```

In This Example

- Dynamic hyperlinks with unique values are added to the Product_line variable.
- The ODS Excel option SHEET_NAME= is used with the #BYVAL argument to name worksheets that are the targets of the hyperlinks.

Output



	A	B	C	D	E	F
1	Product: Children					
2	Product: Clothes & Shoes					
3	Product: Outdoors					
4	Product: Sports					
5						
6						

Output 7. Using PROC ODSTEXT and the Text added with the PROC ODSTEXT

Using the PRETEXT= and POSTTEXT= Attributes to Add Text to a Variable Value

You can add text before or after character variable values in a cell or before or after a table by using the PRETEXT= or POSTTEXT= attributes. However, you cannot use these attributes with numeric values because Microsoft Excel (unlike SAS ODS formatted destinations that are not based on Excel) enforces data types.

If you want to add text before or after a numeric field, you must first set the cell type to STRING in the TAGATTR= attribute (as shown in Example 8).

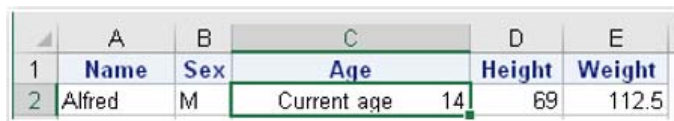
Example 8

```
ods excel;
proc report data=sashelp.class;
  define age /style(column)={tagattr="type:String" pretext="Current
                             age"};
run;
ods excel close;
```

In This Example

- This example uses the TAGATTR="TYPE:String" attribute to change the cell type to STRING.
- The text **Current age** is added by the PRETEXT="Current age" attribute so that the text is placed before the numeric value (which was changed to type STRING) 14 in the **Age** cell.

Output



	A	B	C	D	E
1	Name	Sex	Age	Height	Weight
2	Alfred	M	Current age 14	69	112.5

Output 8. Using the PRETEXT= and the TAGATTR= Attributes to Add Text before a Variable Value

Displaying Text with the Best Fit for Text within Cells

With the ODS Excel destination, column width in a worksheet might not always fit text the best in the display, especially if you have longer strings. To display this column with the optimal width for text, you need to set the cell width explicitly with the WIDTH= style

attribute. You use this attribute in the TEMPLATE procedure or in other SAS procedures that enable you to override style. The WIDTH= attribute overrides the default measurement. As another option, you can use the ODS Excel suboption ABSOLUTE_COLUMN_WIDTH=. This option sets the width for the cell after the text for that cell is optimally sized.

	A	B	C	D	E
1	Var1	Year	Quarter	Product_Line	Product_Group
2	This is a long string which is used to demonstrate how values display	1999	1999Q1	Children	A-Team, Kids
3	This is a long string which is used to demonstrate how values display	1999	1999Q1	Children	Bathing Suits, Kids

Output 9(a). Column Var1 As It Appears before You Use the ABSOLUTE_COLUMN_WIDTH= Option

Example 9 illustrates how to remove extra blank spaces in columns in a worksheet.

Example 9

```
data one;
  length Var1 $75;
  set sashelp.orsales;
  Var1="This is a long value that is used to demonstrate how values
  display";
run;

ods excel file="c:\temp\wrap.xlsx" options(flow="tables" sheet_name="test"
  absolute_column_width="4.1in,5,7,15,18");

proc print data=one noobs;
  var Var1 Year Quarter Product_Line Product_Group;
run;
ods excel close;
```

In This Example

The ABSOLUTE_COLUMN_WIDTH= option in this example controls the column width, giving you a method for removing extra blank spaces.

Output

	A	B	C	D	E
1	Var1	Year	Quarter	Product_Line	Product_Group
2	This is a long value that is used to demonstrate how values display	1999	1999Q1	Children	A-Team, Kids
3	This is a long value that is used to demonstrate how values display	1999	1999Q1	Children	Bathing Suits, Kids

Output 9(b). Extra Space in Column Var1 Is Removed When You Specify the ABSOLUTE_COLUMN_WIDTH= Option

Automatically Fitting Columns in the Table

To automatically fit cells in a worksheet, select the **Autofit Column Width** option (under **Home ► Format** in Excel). You can also achieve the same result in an automated manner by using the AUTOFIT_COLUMNS= option in the %Excel_Enhance macro. Doing so ensures the best space of the cells using the macro or with Excel.

The following example autofits the output that is shown in Output 10(a). This output shows blank spaces at the end of the Var1 column.

	A	B	C	D	E
1	Var1	Year	Quarter	Product_Line	Product_Group
2	This is a long value that is used to demonstrate how values display	1999	1999Q1	Children	A-Team, Kids
3	This is a long value that is used to demonstrate how values display	1999	1999Q1	Children	Bathing Suits, Kids

Output 10(a). Var1 Column before You Use the AUTOFIT_COLUMNS= Option

Example 10 shows you how to use the AUTOFIT_COLUMNS= option in the macro. Note that the original file is generated in Example 9. So, only the %Excel_Enhance macro statement is shown in the following example.

Example 10

```
%excel_enhance(open_workbook=c:\temp\wrap.xlsx,
               autofit_columns="test");
```

Output

	A	B	C	D	E
1	Var1	Year	Quarter	Product Line	Product Group
2	This is a long value that is used to demonstrate how values display	1999	1999Q1	Children	A-Team, Kids
3	This is a long value that is used to demonstrate how values display	1999	1999Q1	Children	Bathing Suits, Kids

Output 10(b). Var1 Column after You Use the AUTOFIT_COLUMNS= Option

USING PROC EXPORT TO GENERATE AND UPDATE WORKSHEETS

The previous section demonstrated the power of the Excel destination in generating multiple worksheets per workbook. However, you can also use PROC EXPORT to generate multiple worksheets per workbook (using the DBMS=XLXS option) and to update worksheets.

This procedure handles large files very efficiently. However, PROC EXPORT cannot add styles (for example, adding colors) or add formulas.

Example 11 shows a basic EXPORT procedure that generates an Excel file.

Example 11

```
proc export data=sashelp.orsales outfile="c:\temp\table.xlsx"
           dbms=xlsx replace;
run;
```

Output

This procedure generates the following output:

	A	B	C	D	E	F	G	H
1	Year	Quarter	Product Line	Product Category	Product Group	Quantity	Profit	Total Retail Price
2	1999	1999Q1	Children	Children Sports	A-Team, Kids	286	4980.15	8990.9
3	1999	1999Q1	Children	Children Sports	Bathing Suits, Kids	98	1479.95	2560.4
4	1999	1999Q1	Children	Children Sports	Eclipse, Kid's Clothes	588	9348.95	18768.8
5	1999	1999Q1	Children	Children Sports	Eclipse, Kid's Shoes	224	7136.8	14337.2

Output 11. Output for the SASHELP.ORSALES Data Set That Is Generated with PROC EXPORT

EXTENDING THE FUNCTIONALITY OF THE EXPORT PROCEDURE

PROC EXPORT can be enhanced to provide some common features in Excel worksheets (for example, adding filters, freezing headers, or adding Excel styles, and so on).

The next example demonstrates how to add some enhanced functionality by using PROC EXPORT with the %Excel_Enhance macro. The program uses the TABLE_STYLE= attribute to generate an Excel style that includes column banding (highlighting), column totals, and the application of the TableStyleDark10 style.

Note: This example uses the [same PROC EXPORT step](#) that is shown in Example 11.

Example 12

```
%excel_enhance(open_workbook=c:\temp\table.xlsx,  
              table_style=%str(ShowTableStyleColumnStripes=1  
                               ShowTotals=1  
                               TableStyle="TableStyleDark10");
```

In This Example

The TABLE_STYLE= attribute generates an Excel style that includes alternating column banding, column totals, and the application of the TableStyleDark10 style.

Output

	A	B	C	D	E	F	G	H
1	Year	Quarter	Product Line	Product Category	Product Group	Quantity	Profit	Total Retail Price
2	1999	1999Q1	Children	Children Sports	A-Team, Kids	286	4980.15	8990.9
3	1999	1999Q1	Children	Children Sports	Bathing Suits, Kids	98	1479.95	2560.4
4	1999	1999Q1	Children	Children Sports	Eclipse, Kid's Clothes	588	9348.95	18768.8
5	1999	1999Q1	Children	Children Sports	Eclipse, Kid's Shoes	334	7136.8	14337.2
6	1999	1999Q1	Children	Children Sports	Lucky Guy, Kids	303	7163	12996.2
7	1999	1999Q1	Children	Children Sports	N.D. Gear, Kids	755	19153.05	34250.5
8	2002	2002Q4	Sports	Winter Sports	Winter Sports	2577	138617.06	246968.66
9	Total							340872.86

Output 12. Using the TABLE_STYLE= Attribute to Apply Style Features

GENERATING PIVOT TABLES USING THE TABLEEDITOR TAGSET

A *pivot table* in Excel summarizes large amounts of data (from other data sources) easily, tracks trends, and enables you to view the data dynamically in different views. Microsoft Excel 2016 also has new functionality that permits you to select multiple tabs in a slicer (as illustrated in in [Output 13](#)).

You can also generate pivot tables with the SAS TableEditor tagset. The tagset provides various options (for example, PIVOTROW=, PIVOTCOL=, and PIVOTDATA=) that enable you to place variables in the Excel layout areas of a report that are available from the **PivotTable Fields** options list. The tagset also enables you to create one or more *slicers* (visual filters) by using the PIVOT_SLICER= option. The TableEditor tagset also contains new style options (as illustrated below in Example 13). To find information about the tagset options, submit DOC="help" as an option in the ODS TAGSETS.TABLEEDITOR statement.

Example 13 generates a workbook to which a pivot table is added. The TableEditor tagset generates a JavaScript file (with the .js extension) that contains a script to create the pivot table.

Example 13.

```
/* Reference the tagset from support.sas.com. */  
filename tpl url  
      "http://support.sas.com/rnd/base/ods/odsmarkup/tableeditor/tableeditor.tpl";  
  
/* Insert the tagset into the search path for ODS templates. */  
ods path(prepend) work.templat(update);  
%include tpl;  
  
options noxsync noxwait;  
/* Create sample data to which to add pivot data. */  
ods excel file="c:\temp\report.xlsx" options(sheet_name="Sales");  
proc print data=sashelp.prdsale;  
run;  
ods excel close;
```

(code continued)


```
ods tagsets.tableeditor file="c:\temp\PivotTable.js"
  options(update_target="c:\\temp\\report.xlsx"
    output_type="script"
    sheet_name="Sales"
    pivot_sheet_name="Profit Analysis"
    pivotrow="month, country"
    pivotcol="product"
    pivotdata="actual"
    pivot_slicer="MONTH" /* Has the match case in the source. */
    header_bgcolor="black"
    rowheader_bgcolor="black"
    rowheader_fgcolor="white"
    data_bgcolor="18"
    datalabel_bgcolor="magenta"
    pivotdata_fmt="$#,###.##"
  );

data _null_;
  file print;
  put _all_;
run;
ods tagsets.tableeditor close;
x "c:\temp\PivotTable.js";
```

In This Example

- The Excel destination creates a workbook to which a pivot table is added.
- The FILE= option generates the JavaScript that, when executed, generates the pivot table.
- The UPDATE_TARGET= option loads the file with the data from which the pivot table is created.
- The PIVOTROW=, PIVOTCOL=, AND PIVOTDATA= options add fields to the various Excel layouts that are available in the Pivotable Fields list.
- The PIVOT_SLICER= option generates the MONTH slicer.
- The style options (ROWHEADER_BGCOLOR=, ROWHEADER_FGCOLOR=, DATA_BGCOLOR=, and DATALABEL_BGCOLOR=) modify the row, data, columns, and labels.
- The DATA_NULL_ step triggers the execution of the .js file, and the X statement executes the pivot table.

Output

Row Labels	BED	CHAIR	DESK	SOFA	TABLE	Grand Total
Jan	\$4,085.	\$5,174.	\$7,303.	\$6,802.	\$6,449.	\$29,813.
CANADA	\$856.	\$979.	\$2,492.	\$1,953.	\$1,835.	\$8,115.
GERMANY	\$1,875.	\$2,107.	\$2,044.	\$3,723.	\$1,490.	\$11,239.
U.S.A.	\$1,354.	\$2,088.	\$2,767.	\$1,126.	\$3,124.	\$10,459.
Feb	\$5,025.	\$6,306.	\$4,855.	\$7,290.	\$6,108.	\$29,584.
CANADA	\$1,581.	\$2,323.	\$2,460.	\$2,483.	\$1,584.	\$10,431.
GERMANY	\$1,929.	\$1,802.	\$956.	\$2,393.	\$2,217.	\$9,297.
U.S.A.	\$1,515.	\$2,181.	\$1,439.	\$2,414.	\$2,307.	\$9,856.
Mar	\$4,918.	\$5,620.	\$5,819.	\$7,317.	\$6,199.	\$29,873.
CANADA	\$1,900.	\$1,813.	\$1,692.	\$2,495.	\$2,558.	\$10,458.
GERMANY	\$1,222.	\$2,367.	\$1,330.	\$2,799.	\$2,054.	\$9,772.
U.S.A.	\$1,796.	\$1,440.	\$2,797.	\$2,023.	\$1,587.	\$9,643.
Grand Total	\$14,028.	\$17,100.	\$17,977.	\$21,409.	\$18,756.	\$89,270.

Output 13. Generating a Pivot Table from an Excel File

USING OPEN-SOURCE SOFTWARE TO UPDATE WORKSHEETS

Individuals and companies have begun using open-source software (for example, Python and Java) because it provides additional functionality and enables customization for specific needs. Open-source software also provides methods for reading, modifying, and generating XLSX files. The Python language performs these tasks by using the `openpyxl` package and Java uses the [Apache Poi library](#).

Using the combination of open-source software (Python or Java) and SAS software (reporting tools such as the Excel destination or PROC EXPORT) enables you to take advantage of automating the reporting process. The next sections demonstrate how to use the Python `openpyxl` package, which modifies XLSX files in Python. You can run and call Python functions (for example, to include the `openpyxl` package) with the SAS FCMP procedure.

ENHANCING WORKSHEETS USING THE PYTHON OPENPYXL PACKAGE

The good news is that you do not need to be a Python programmer to take advantage of the Python language for enhancing Excel output. [Excellent openpyxl documentation](#) is available that demonstrates the object model and methods that you need to enhance your output.

First, you need to install Python which you can do using the [Anaconda distribution package](#). After you install Python, then you need to install the `openpyxl` package by submitting the following command from a command line:

```
$ pip install openpyxl
```

Based on information in the `openpyxl` documentation, you can add functionality that you want in worksheets that you generated previously with PROC EXPORT or with the ODS Excel destination. In the following example, the Excel worksheet is created with PROC EXPORT, and additional functionality is added with the Python script that is shown in Display 1. This script imports the packages and loads workbooks.

Example 14

First, you use PROC EXPORT to create a worksheet based on the SASHELP.ORSALES data, as shown below. The workbook `sales.xlsx` is written to the current working directory.

```
proc export data=sashelp.orsales
  outfile="sales.xlsx" dbms=xlsx;
run;
```

To locate your current working directory (where Python searches for relative files), submit the following code:

```
import os
print os.getcwd()
```

Then, use the following Python script to modify the worksheet. In this case, you modify tab name (by making the value uppercase) and tab color.

```
import openpyxl
import load_workbook
#Create workbook object
wb=load_workbook('sales.xlsx')
#Create worksheet object
ws=wb.active
#Modify worksheet tab name
ws.title='sales'
#Color the tab name
ws.sheet_properties.tabColor="red"
```

(code continued)

```
#Save workbook with a new name
wb.save('sales_update.xlsx')
```

In This Example

Once you import the `openpyxl` and `load_workbook` packages, the Python script uses methods to modify the tab name and change the tab color.

USING PROC FCMP TO CREATE PYTHON FUNCTIONS IN SAS®

As mentioned earlier, you also can modify Excel worksheets that are created in SAS by using Python objects. You do this by including a Python function, which includes the `openpyxl` package, and using Python objects within the FCMP procedure.

The first step in using PROC FCMP and Python objects is to declare the objects in a DECLARE statement, as shown in Example 15.

```
DECLARE OBJECT object-name(PYTHON);
```

You include the Python functions between the SAS SUBMIT and ENDSUBMIT statements.

```
SUBMIT INTO PY;
```

```
    python-syntax
```

```
ENDSUBMIT;
```

Other methods and techniques (for example, using the INFILE method) are available with PROC FCMP for including external code. For more information, see [What's New in FCMP for SAS 9.4 and SAS Viya](#).

After you submit your code, you then must publish the code that is stored in the Python object to the Python interpreter. The interpreter validates the syntax of the code. Finally, after the Python function is published, you can call it from either PROC FCMP or from a DATA step. You can also pass the value back to SAS using the RESULTS method. However, the next example publishes and calls the Python function without returning a value.

Example 15 adds a Python function in PROC FCMP to calculate the change the worksheet name and color.

Example 15

```
proc fcmp;
  declare object py(python);
    /* Create an embedded Python block to write your Python function */
  submit into py;
  def MyFunc(arg1,arg2):
    "Output: MyKey"
    from openpyxl import load_workbook
    wb=load_workbook(filename=arg1)
    ws=wb.active
    ws.title="Sales"
    ws.sheet_properties.tabColor="FF0000"
  endsubmit;

    /* Publish the code to the Python interpreter */
  rc=py.publish();

    /* Call the Python function from SAS */
  rc=py.call("MyFunc","sales.xlsx","sales_update.xlsx");
run;
```

In This Example

- The Python object `py` is created in the DECLARE statement.
- The Python function and two arguments (Arg1 and Arg2) are added in the SUBMIT block.
- The "Output: MyKey" argument is required for denoting the variable type.
- The `publish` and `call` methods compile and execute the Python function from SAS. They also return a value to the variable RC if the compile and execution are successful.

Output



	A	B	C	D	E	F
1	Year	Quarter	Product_Line	Product_Category	Product_Group	Quantity
2	1999	1999Q1	Children	Children Sports	A-Team, Kids	286
3	1999	1999Q1	Children	Children Sports	Bathing Suits, Kids	98
4	1999	1999Q1	Children	Children Sports	Eclipse, Kid's Clothes	588
5	1999	1999Q1	Children	Children Sports	Eclipse, Kid's Shoes	334
6	1999	1999Q1	Children	Children Sports	Lucky Guy, Kids	303

Output 15. Changing a Worksheet Name and Tab Color Using Python and PROC FCMP

This output is the same as the output that is generated by the code in Example 14.

Modifying Existing Excel Files by Using PROC FCMP and a Python Function

The next series of examples use a Python function within PROC FCMP to update worksheets that are created with PROC EXPORT. These examples apply an Excel style, conditional formatting, and a graph dashboard to an Excel table.

To make managing and analyzing a group of related data easier, you can turn a range of cells into an Excel table. (Excel tables were previously called Excel lists.)

Excel tables enable you to modify various sections of a table individually or as a group. By default, tables are generated with headers in the first row and with column filters. Styles are managed by using the `TableStyleInfo` object.

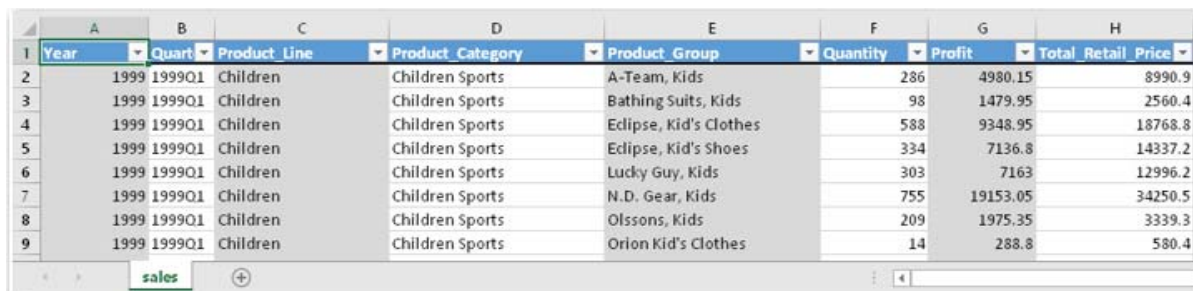
Example 17

```
proc fcmp;
  declare object py(python);
  submit into py;
  def MyFunc(arg1,arg2):
    "Output: MyKey"
    from openpyxl import load_workbook
    from openpyxl.worksheet.table import Table, TableStyleInfo
    wb=load_workbook(filename=arg1)
    ws=wb.active
    # define a table style
    med=TableStyleInfo(name='TableStyleMedium16',showColumnStripes=True)
    # Create a table
    tab=Table(displayName="Table1",ref="A1:J1441",tableStyleInfo=med)
    # Add the table to the worksheet
    ws.add_table(tab)
    wb.save(filename=arg2)
  endsubmit;
  rc=py.publish();
  rc=py.call("MyFunc","sales.xlsx","excel_table.xlsx");
run;
```

In This Example

- The packages that are needed for this example are added to the Python function.
- The Python variable `med` specifies style information (the table style column color-stripping attributes) for the table.
The Python variable `tab` specifies all the information that is required to create an Excel table. That information includes the table name (`Table1`), the range (via the `ref=` variable) , and the style (via the `TableStyleInfo=` option).
- The `ws.add_table` method adds the Excel table to the worksheet.

Output



Year	Quart	Product Line	Product Category	Product Group	Quantity	Profit	Total Retail Price
1999	1999Q1	Children	Children Sports	A-Team, Kids	286	4980.15	8990.9
1999	1999Q1	Children	Children Sports	Bathing Suits, Kids	98	1479.95	2560.4
1999	1999Q1	Children	Children Sports	Eclipse, Kid's Clothes	588	9348.95	18768.8
1999	1999Q1	Children	Children Sports	Eclipse, Kid's Shoes	334	7136.8	14337.2
1999	1999Q1	Children	Children Sports	Lucky Guy, Kids	303	7163	12996.2
1999	1999Q1	Children	Children Sports	N.D. Gear, Kids	755	19153.05	34250.5
1999	1999Q1	Children	Children Sports	Olssons, Kids	209	1975.35	3339.3
1999	1999Q1	Children	Children Sports	Orion Kid's Clothes	14	288.8	580.4

Output 16. Generating an Excel Table from PROC EXPORT Output by Using a Python Function

Conditional Formatting

Conditional formatting enables you to add a format either to a cell or a range of cells in an Excel file. You can use this type of formatting to add data bars, icon sets, and a color scale to elements in your Excel file. A *color scale* adds color to cells in a range of cells based on the values for those cells. This feature enables you to trends as well as minimum and maximum values.

Example 17 demonstrates how to add data bars as well as a color scale. For data bars, the below are the type of values in the cell that the data bars can be generated. The appropriate package is the first thing which is modified, before adding the rule.

Example 17

```
proc fcmp;
  declare object py(python);
  submit into py;
  def MyFunc(arg1,arg2):
    "Output: MyKey"
    from openpyxl import load_workbook
    from openpyxl.formatting.rule import DataBarRule, ColorScaleRule
    wb=load_workbook(filename=arg1)
    ws=wb.active
    # Add the rule for the data bars.
    rule=DataBarRule(start_type='num',start_value=10,
end_type='num', end_value='800',
color="FF638EC6",showValue="None")
    # Add data bars to the worksheet.
    ws.conditional_formatting.add('F2:F10',rule)
    # Add a color scale to the worksheet.
    ws.conditional_formatting.add('G2:F20',
ColorScaleRule(start_type='min',start_color='FF638EC6',
end_type='max', end_color='FF638EC6'))
    wb.save(filename=arg2)
```

(code continued)

```

endsubmit;
rc=py.publish();
rc=py.call("MyFunc","sales.xlsx","Condition.xlsx");
run;

```

In This Example

- Several packages are included in this example. These packages load the workbook and create data bars and color scales.
- The **wb** workbook object is created from the workbook; the **ws** object is created from the active worksheet.
- The **rules** variable is created with the rules for the data bars. The data bars are created with information such as the starting and ending colors and values.
- The **ws** object and the **conditional_formatting.add** method create rules for creating data bars and a color scale.
- The **.save** method creates a new workbook with the updates.

Output

	A	B	C	E	F	G
1	Year	Quarter	Product_Line	Product_Group	Quantity	Profit
2	1999	1999Q1	Children	A-Team, Kids	286	4980.15
3	1999	1999Q1	Children	Bathing Suits, Kids	98	1479.95
4	1999	1999Q1	Children	Eclipse, Kid's Clothes	588	9348.95
5	1999	1999Q1	Children	Eclipse, Kid's Shoes	334	7136.8
6	1999	1999Q1	Children	Lucky Guy, Kids	303	7163
7	1999	1999Q1	Children	N.D. Gear, Kids	755	19153.05
8	1999	1999Q1	Children	Olssons, Kids	209	1975.35

Output 18. Adding Data Bars and a Color Scale

Creating Graphs

Using code similar to that in the previous example, you can add graphs to your Excel worksheet. With the Python openpyxl package, you can add various types of graphs: bar charts, VBAR charts, line charts, bubble plots, stock charts, and more. You can create a single graph or multiple graphs in a dashboard.

Example 18 creates a summary table along with a bar chart. This code generates a summary table that was created with the SQL procedure.

Example 18

```

proc sql;
  create table temp as
  select distinct(product) as Product, sum(actual) as Actual, sum(predict)
  as Predicted
  from sashelp.prdsale;
quit;

proc export data=temp;
  outfile="graph.xlsx" dbms=xlsx;
run;

proc fcmp;
  declare object py(python);
  submit into py;
  def Myfunc(arg1,arg2):
    "Output: MyKey"

```

(code continued)

```

from openpyxl import load_workbook
from openpyxl.chart import BarChart, Series, Reference
wb=load_workbook(filename=arg1)
ws=wb.active
chart=BarChart()
chart.title="Sales Report"
data=Reference(ws, min_row=1, max_row=2, min_col=2,
              max_col=3)
cats=Reference(ws, min_col=1, min_row=2, max_row=6)
chart.add_data(data, titles_from_data=True)
chart.set_categories(cats)
ws.add_chart(chart, "E2")
wb.save("arg2")

endsubmit;
rc=py.publish();
rc=py.call("Myfunc", graph.xlsx, "output.xlsx");
run;

```

In This Example

- PROC EXPORT generates a worksheet that contains a summary.
- The **chart=** variable, which holds the chart type, is set to **BarChart()**, which creates a bar chart.
- The **data=** variable references the data that is used to create the bar chart.
- The actual chart is added by using the **chart.add** method.
- The **.save** method creates a new, second worksheet.

Output



Output 18. Creating a Bar Chart by Using the Python openpyxl Package

CONCLUSION

Using the SAS ODS Excel destination in combination with Microsoft Excel leverages the power of SAS and automates the generation of presentation-ready spreadsheet. This paper discussed several tools that you can use to generate worksheets with SAS 9.4 (ODS Excel destination, PROC EXPORT, and DDE) and SAS Viya (PROC CASUTIL). The discussion also explained how you can use the %Excel_Enhance macro, PROC EXPORT, and open-source software (Python and Java) to build the worksheet that you want. All of these tools can be added to your toolbox for creating perfect, presentational worksheets.

REFERENCES

- Gazoni, Eric and Charlie Clark. 2020. "openpyxl: A Python library to read/write Excel 2010 xlsx/xlsm files." OpenPyXL website. Available at openpyxl.readthedocs.io/en/stable/. Accessed on February 5, 2020.

- Parker, Chevell. 2018. "Insights from a SAS Technical Support Guy: A Deep Dive into the SAS® ODS Excel Destination." *Proceedings of the SAS Global Forum 2018 Conference*. Cary, NC: SAS Institute Inc. Available at support.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2174-2018.pdf .
- Whitcher, Michael, et al. 2019. "What's New in FCMP for SAS 9.4 and SAS Viya." ."
Proceedings of the SAS Global Forum 2019 Conference. Cary, NC: SAS Institute Inc. Available at www.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2019/3480-2019.pdf.

ACKNOWLEDGEMENTS

I would like to thanks Susan Berry for her editorial help with this paper.

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

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.