Paper 3388-2019

Power Up Your Reporting Using the SAS® Output Delivery System

Chevell Parker, SAS Institute Inc.

ABSTRACT

Making sense of a large amount of data is one of the most important aspects of a reporting system. Reporting helps you and others in your organization discover important insights into trends, business strengths and weaknesses, and the overall health of a company. Therefore, report output should be in a format that anyone can understand easily. To create such output, you need to use the correct reporting tools. This paper, written for data analysts, discusses techniques to power up (amplify) the effectiveness of your reporting. These techniques use SAS® Output Deliver System (ODS) destinations (especially the ODS Excel destination) to generate functional, presentation-ready Microsoft Excel worksheets. The discussion also explains how to use the ODS destinations to enhance web pages and other types of documents. Finally, the paper explains how you can use Python open-source software with the SAS® System and ODS destinations to further enhance your reporting.

INTRODUCTION

The SAS Output Delivery System has an ODS destination to fit almost every formatting need, including Excel, CSV, HTML, PowerPoint, EPUB, Word, and others. So, it's easy to understand why ODS is so popular. You can do so much with the standard functionality of these destinations. However, this paper also discusses how you can enhance some of the standard functionality to make these destinations even more useful. The discussion covers several of the destinations that you can use to enhance your reports. The paper concludes by explaining how you can use Python open-source software to enhance your reporting.

EXPORTING OUTPUT TO MICROSOFT EXCEL USING THE SAS® OUTPUT DELIVERY SYSTEM

ODS provides multiple methods for exporting output to an Excel file. For example, the ODS Excel destination writes output in the XLSX file format that is native to Excel. This section discusses using the ODS Excel destination.

This section covers the following topics:

- using the ODS Excel destination to generate native Microsoft Excel files
- enhancing Microsoft Excel worksheets using formats
- using the SAS Report Writing Interface with formats and formulas to enhance

USING THE ODS EXCEL DESTINATION TO GENERATE NATIVE MICROSOFT EXCEL FILES

The ODS Excel destination generates native Excel files from SAS® procedures and the DATA step. The ODS Excel destination has three prongs that allow you to plug in to its power for creating functional and highly presentational worksheets.

- The first prong consists of the ODS Excel options. You can choose from over 50 such options in SAS 9.4M6 (TS1M6). These options work alone or in combination with the Excel options to perform tasks such as filtering, freezing headings or rows, or adding printing options. *(list continued)*

- The second prong entails built-in styles that you can use to enhance your presentation. Those styles include global styles that you obtain by using either the TEMPLATE procedure or cascading style sheets. The STYLE= override option for procedures is also a part of this prong.
- The third prong is the TAGATTR= style attribute, which works like an API that you can use with the Excel destination. This attribute enables you to add Excel formats and formulas, rotated text, and more.

You can see these three prongs at work in the following example, which reads a JSON file from an open data site and creates a SAS data set. The example uses ODS Excel options, the TAGATTR= style attribute (used in this case, to rotate headings), and the style CALL DEFINE statement in the PROC REPORT step to create striped rows. The data set from this example is included in the downloads.

Example 1

```
filename temp url
          "https://dashboard.hawaii.gov/resource/wz7p-e5jq.json";
libname temp json;

proc transpose data=temp.root out=budget name=Category;
   id fy;
   var capital_assets_net current_and_other_assets
        deferred_loss_on_refunding
        net_investment_in_capital_assets total_assets
        long_term_liabilities other_liabilities total_liabilities
        total_net_position;
run;

ods excel file="c:\temp\test.xlsx" options(frozen_headers="1"
                                           frozen_rowheaders="1"
                                           row_heights="30"
                                           tab_color="blue");

   proc report data=budget style(header)={tagattr="rotate:45" color=white
                background=#b0b0b0};
      define Category / " ";
      compute Category;
          count+1;
          if mod(count,2) then call
          define(_row_,"style","style={background= #edfbf5}");
   endcomp;
   run;

ods excel close;
```

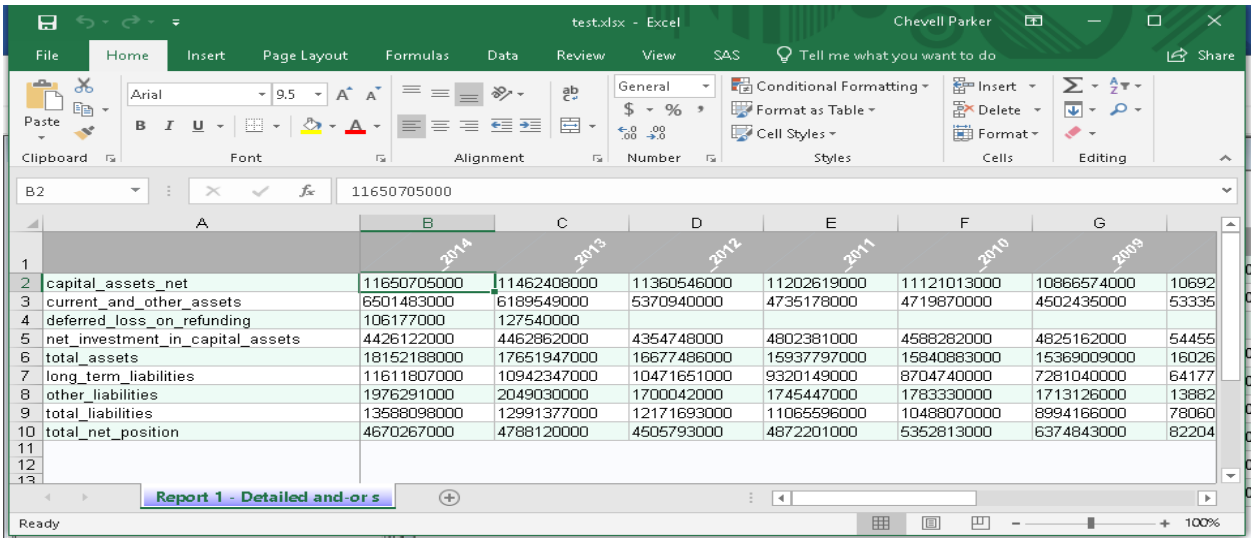
In This Example

- The JSON engine reads open budget data from the state of Hawaii, and the TRANSPOSE procedure reshapes the data by making the FY values the column headings.
- The TAGATTR= attribute in the PROC REPORT statement rotates the headers by 45 degrees.
- In the ODS Excel statement, the options FROZEN_HEADERS= and FROZEN_ROWHEADERS= are included, respectively, to freeze the column headings and the first row. In addition, the ROW_HEIGHTS= option adds height to the rotated header, and the TAB_COLOR= option adds the color blue to the worksheet tab.

(list continued)

- The MOD function in the PROC REPORT compute block, along with the CALL DEFINE statement and the value _ROW_, create rows in alternating colors, as shown in the output below.

Output



		2014	2013	2012	2011	2010	2009	
2	capital_assets_net	11650705000	11462408000	11360546000	11202619000	11121013000	10866574000	10692
3	current_and_other_assets	6501483000	6189549000	5370940000	4735178000	4719870000	4502435000	53335
4	deferred_loss_on_refunding	106177000	127540000					
5	net_investment_in_capital_assets	4426122000	4462862000	4354748000	4802381000	4588282000	4825162000	54455
6	total_assets	18152188000	17651947000	16677486000	15937797000	15840883000	15369009000	16026
7	long_term_liabilities	11611807000	10942347000	10471651000	9320149000	8704740000	7281040000	64177
8	other_liabilities	1976291000	2049030000	1700042000	1745447000	1783330000	1713126000	13882
9	total_liabilities	13588098000	12991377000	12171693000	11065596000	10488070000	8994166000	78060
10	total_net_position	4670267000	4788120000	4505793000	4872201000	5352813000	6374843000	82204

Output 1. Excel File That Is Generated by Options, Style Attributes, and the TAGATTR= Attribute

ENHANCING YOUR MICROSOFT EXCEL WORKSHEETS WITH FORMATS AND FORMULAS

This section examines methods for enhancing output that is created with the ODS Excel destination. The section also describes potential problems that you might encounter. One of the ways you can add more power to the ODS Excel destination is by using Excel formatting. This formatting includes the use of SAS formats that are converted to Excel format and the use of custom Excel formats that are implemented with the TAGATTR= style attribute.

You can use formatting to facilitate the correct display of data in Excel as well as to enhance your presentation. If you do not apply a format, the default format used for all cells is the General format. However, this format might not display the data as you want it to appear.

Be aware that various formatting issues can occur when you use the ODS Excel destination and Excel formats with (or without, in some cases) the ODS Excel destination. The following list describes certain problems that can occur when you use the General format.

- Leading and trailing zeros are not retained, with the exception of numbers between -1 and 1.
- Numbers with eleven or more digits are displayed in scientific notation.
- Numbers that have an embedded "E" might be interpreted as a value in scientific notation.
- Ranges might be translated into dates.
- Values that Excel does not recognize as numbers are stored as text. This problem can prevent calculations or affect sorting order.
- All numbers, regardless of how they are stored in SAS, default to the General format if you do not specifically apply a SAS format or a custom Excel format.
- To display text as you type it, you should use the TYPE parameter with a value of String in the TAGGATR= attribute (Example 3) or by using the $*w*. SAS format.

(list continued)

An Excel number format consists of four sections, separated by semicolons, that appear in this order:

The format includes positive numbers, negative numbers, zero values and text. If you specify only one section, the number format applies to all number types (for example, positive numbers, negative numbers, and zeros). If you specify only two sections, the first section is applied to positive numbers and zero values, and the second section is applied to negative numbers.

The following table lists features of SAS formats and custom Excel formats.

SAS Formatting	Custom Excel Formatting
SAS formatting is convenient and easy to use.	You have total formatting control (via control of each section of the format).
If you are already a SAS user, you might already be familiar with SAS formatting.	There is a small learning curve with using custom formatting.
SAS formatting supports National Language Support (NLS).	You cannot validate custom formatting with the TAGATTR= attribute (invalid values can corrupt the file).
Limited formatting is available.	Custom formatting is efficient.
SAS formats are documented extensively.	Custom Excel formatting changes the appearance and not the underlying value.

SAS® Formatting

When you apply SAS formats in your worksheets, the formats are mapped to the equivalent Excel format without you having to do anything else. This mapping ensures that Excel formats are applied, guaranteeing that the displayed Excel output looks the way that you expect. The SAS formats implement basic formatting, and the use of these formats enables you to avoid some of the issues that are mentioned earlier about using the General format without an Excel format.

The following example demonstrates the use of SAS formatting (with both character and numeric formats) to maintain the correct display in Excel.

Example 2

```
data one;
   char_leading="0001";       Num_leading=0001;
   char_long="123456789012"; Num_long=123456789012;
   char_string="22.900";      Num_string=22.900;
   char_sci="1e9";
run;
```

(code continued)

```
ods excel file="c:\temp\format.xlsx";

   proc print data=one;
      format Char_leading $char4. Num_leading Z4.;
      format Char_long $13.       Num_long Best.;
      format Char_string $8.      Num_string 6.3;
      format Char_sci $3.;
   run;

ods excel close;
```

In This Example

- The $CHAR4. SAS format is applied to the character variable Char_leading, and the numeric format Z4. is applied to the numeric variable Num_leading. These formats maintain the leading zeros for each variable value.
- The $13. format ($*w*.*d*) is applied to the character variable Char_long and the BEST. format is applied to the numeric variable Num_long. These formats prevent the display of the variable values in scientific notation.
- The $8. format ($*w*.*d*) to preserves the trailing zero for the variable Char_string. The 6.3 format (*w*.*d*) also preserves the trailing zero for the numeric variable Num_string.
- The $*w*.*d* format is applied to the variable Char_sci. This format forces the value for Char_sci to be a text value rather than numeric format.

Output

	A	B	C	D	E	F	G	H
1	Obs	Char_leading	Num_leading	Char_long	Num_long	Char_string	Num_string	Char_sci
2	1	1	1	1.23457E+11	1.23457E+11	22.9	22.9	1000000000

Output 2. Table before Formatting Is Applied

	A	B	C	D	E	F	G	H
1	Obs	Char_leading	Num_leading	Char_long	Num_long	Char_string	Num_string	Char_sci
2	1	0001	0001	123456789012	123456789012	22.900	22.900	1e9

Output 3. Table after SAS Formatting Is Applied

Custom Excel Formatting

Custom Excel formats control how numbers look in Excel, and these formats do so without changing any of your data. In addition to preventing the general format from displaying numbers incorrectly, you can also use custom Excel formatting to enhance the output. The following example uses custom Excel formatting to prevent the incorrect display of numbers. Custom formats can only work with value passed to ODS Excel, therefore, there are times when a SAS format is required in addition to the custom format.

Example 3

```
data one;
   leading=0001;
   number=22.900;
   long_value=123456789012;
   char_sci='1e9';
run;
```

(code continued)

```
ods excel file="c:\temp\custom.xlsx";

   proc print data=one;
      var leading /style(data)={tagattr="format:0000"};
      var number / style(data)={tagattr="format:##.000"};
      var long_value / style(data)={tagattr="format:####"};
      var char_sci / style(data)={tagattr="type:string"};
   run;

ods excel close;
```

In This Example

- The 0000 format displays leading zeroes in the value.
- The ##.000 format displays the trailing zero for the number variable.
- The # format prevents the variable LONG_VALUE from being displayed in scientific notation.
- The variable CHAR_SCI uses the TYPE parameter rather than the custom format to display the value exactly as it is. For more information, see the section "Using the Many Functions of the TAGATTR= Attribute" in "Insights from a SAS Technical Support Guy: A Deep Dive into the SAS® ODS Excel Destination." (**support.sas.com/content/dam/SAS/support/en/sas-global-forum-proceedings/2018/2174-2018.pdf**)

Output

	A	B	C	D	E
1	Obs	leading	number	long_value	char_sci
2	1	1	22.9	1.23457E+11	1000000000

Output 4. Table before Formatting Is Applied

	A	B	C	D	E
1	Obs	leading	number	long_value	char_sci
2	1	0001	22.900	123456789012	1e9

Output 5. Table after SAS Formatting Is Applied

The next example applies custom Excel formatting to the SASHELP.HEART data set (see download file) to highlight subjects that require follow-up based on their initial screening. To identify these subjects, the example uses custom formatting to color code higher-level diastolic and systolic numbers. The formatting also uses color coding and text substitution to highlight high cholesterol numbers.

Example 4

```
ods excel file="c:\temp\test.xlsx" options(embedded_titles="yes");

   proc report data=sashelp.heart ;
      title "Heart Health Report";
      column sex height weight diastolic systolic cholesterol bmi;
      define height / style(column)={tagattr='format:##.# "in"'};
      define weight / style(column)={tagattr='format:### "lbs"'};
```

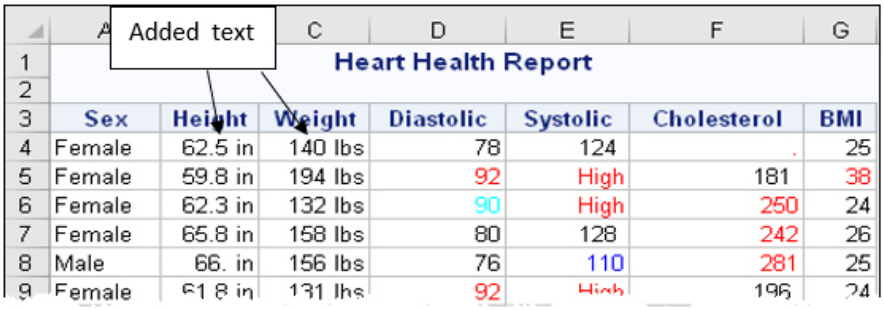
(code continued)

```
      define diastolic /
            style(column)={tagattr='format:[cyan][=90];[red][>90];###'};
      define systolic  /
             style(column)={tagattr='format:[blue][<120];[red][>140]"High";
             ### '};
      define cholesterol /
              style(column)={tagattr='format:[red][>200];### '};
      define BMI /
              style(column)={tagattr='formula:RC[-4]/RC[-5]/RC[-5]*703
              format:[red][>29] ##;##'};
  run;
ods excel close;
```

In This Example

- A custom format, used as part of the TAGATTR= attribute, is applied to the HEIGHT and WEIGHT variables. The formats append the text "in" and "lbs," respectively, for those variables.
- Color coding is added to the variable Diastolic. For borderline readings of 90, the color cyan is applied. For readings greater than 90, the color red applied.
- Color coding and text substitution is added to the variable Systolic variable. Readings greater than 200 are shown in red, and the values are replaced with the text "High." Readings less than 120 are shown in blue.
- The color red is applied for CHOLESTEROL values that are greater than 240.
- The TAGATTR= attribute and the Formula parameter are used to calculate body mass index (BMI). (See the next section for information about formulas.)

Output



	A	C	D	E	F	G
1			Heart Health Report			
2						
3	Sex	Height	Weight	Diastolic	Systolic	Cholesterol	BMI
4	Female	62.5 in	140 lbs	78	124	.	25
5	Female	59.8 in	194 lbs	92	High	181	38
6	Female	62.3 in	132 lbs	90	High	250	24
7	Female	65.8 in	158 lbs	80	128	242	26
8	Male	66. in	156 lbs	76	110	281	25
9	Female	61.8 in	131 lbs	92	High	196	24

Output 6. Output Custom Excel Formats and Formulas

ENHANCING WORKSHEETS BY USING THE SAS® REPORT WRITING INTERFACE WITH FORMATS AND FORMULAS

The Report Writing Interface is an amazing tool when it comes to generating customized reports. This reporting tool gives you control of tasks (for example, merging cells) that you cannot easily accomplish with the traditional ODS destinations. SAS 9.4M5 includes support for formats and formulas that use custom Excel formatting with the TAGATTR= attribute. For releases earlier than SAS 9.4M5, SAS formatting is not supported with the Report Writing Interface when you use it with Excel destination.

The following table shows the most common methods that are used in creating output with the Report Writing Interface.

Common Methods	Definition
TABLE_START()	Begins or opens a table.
ROW_START()	Starts a row within the table.
FORMAT_CELL()	Adds a cell within the row.
ROW_END()	Ends a row within the table.
TABLE_END()	Ends a table that is open.
TITLE()	Adds a title to a report.
IMAGE()	Ads an image to the worksheet.

When you use the Report Writing Interface, you first need to create an object that will be used with the various methods that are part of the Report Writing Interface. To create the object, use the DECLARE statement, as shown in the next example. The DECLARE statement is executed typically on the first iteration of the DATA step.

The following example uses the BUDGET data set that is created in Example 1.

Example 5

```
data _null_;
   set budget end=last;
   if _n_=1 then do;
   declare odsout obj();
end;
```

After you declare an object, you can then apply methods such as the popular TABLE_START() method and others that create tables, rows, and cells.

The next example uses the Report Writing Interface to generate a balance-sheet report . The report is built by creating a table of assets and liabilities. The Report Writing Interface enables you to use formulas that both calculate totals over the group and that format those totals. You implement formatting and formulas by using the TAGATTR= attribute. This example uses the data set from Example 1. This file contains the complete code for this example. (**ftp.sas.com/techsup/download/base/SGF2018-SAS3388.zip**)

Example 6

```
ods excel file="c:\temp\rwi_sample.xlsx" ;

   data _null_;
      set budget end=last;
      if _n_=1 then do;
      declare odsout obj();
      obj.image(href='c:\image\xyx.png');
      obj.table_start();
         obj.row_start();
            obj.format_cell(data:' ');
            obj.format_cell(data:'Report for December 31.',column_span:3);
         obj.row_end();
```

(code continued)

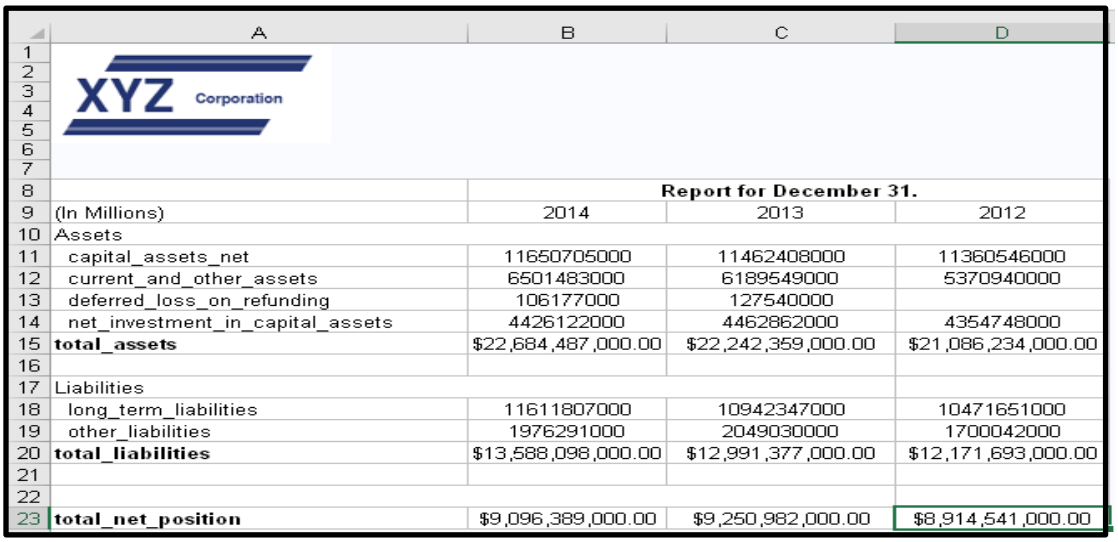
```
      obj.row_start();
         obj.format_cell(data:'(In Millions)',style_attr:"just=l");
         obj.format_cell(data:'2014');
         obj.format_cell(data:'2013');
         obj.format_cell(data:'2012');
      obj.row_end();

      obj.row_start();
         obj.format_cell(data:'Assets',column_span:4,style_attr:"just=left"
                                                                    );
      obj.row_end();
   end;
     obj.row_start();
        if category='total_assets' then do;
        obj.format_cell(data:category,style_attr:"just=l fontweight=bold");
        obj.format_cell(data:_2014,
                        style_attr:"tagattr='formula:=SUM(R[-4]C:R[-
                        1]C)format:currency'");
        obj.format_cell(data:_2013,
                        style_attr:"tagattr='formula:=SUM(R[-4]C:R[-1]C)
                        format:currency'");
        obj.format_cell(data:_2012,
                        style_attr:"tagattr='formula:=SUM(R[-4]C:R[-
                        1]C)format:currency'");
        obj.row_end();
        obj.row_start();
           obj.format_cell(data:" ");
        obj.row_end();
        obj.row_start();
           obj.format_cell(data:"Liabiliies",column_span:3,style_attr:
                                                    "just=left");
        obj.row_end();
      end;
     . . .more statements. . .
     if last then do;
        obj.table_end();
     end;
   run;

ods excel close;
```

In This Example

- The object OBJ() is added using the DECLARE statement.
- The IMAGE() method adds a logo at the beginning of the report.
- The TABLE_START() method is used to start a table.
- The ROW_START() method creates a new row, and the FORMAT_CELL() method adds column headings within the cell. The COLUMN_SPAN option merges the string that spans three columns.
- The IF conditions test whether the program is on the summary row. (The summary rows are named TOTAL_ASSETS, TOTAL_LIABILITIES, and TOTAL_NET_POSITION). Then, the STYLE_ATTR= option adds the TAGATTR= attribute, which adds the Excel format and formula.

- The ELSE condition uses the PRETEXT= option and ASIS= attribute to write cells so that the cells retain the leading space. (This condition is added in the code that is in the download file.) (**ftp.sas.com/techsup/download/base/SGF2018-SAS3388.zip**)

Output



	A	B	C	D
1				
2				
3	XYZ Corporation			
4				
5				
6				
7				
8		Report for December 31.		
9	(In Millions)	2014	2013	2012
10	Assets			
11	capital_assets_net	11650705000	11462408000	11360546000
12	current_and_other_assets	6501483000	6189549000	5370940000
13	deferred_loss_on_refunding	106177000	127540000	
14	net_investment_in_capital_assets	4426122000	4462862000	4354748000
15	**total_assets**	$22,684,487,000.00	$22,242,359,000.00	$21,086,234,000.00
16				
17	Liabilities			
18	long_term_liabilities	11611807000	10942347000	10471651000
19	other_liabilities	1976291000	2049030000	1700042000
20	**total_liabilities**	$13,588,098,000.00	$12,991,377,000.00	$12,171,693,000.00
21				
22				
23	**total_net_position**	$9,096,389,000.00	$9,250,982,000.00	$8,914,541,000.00

Output 7. Balance-Sheet Report That Is Added with the Report Writing Interface (Including Formats and Formulas)

POWERING UP YOUR WEB OUTPUT

The previous sections discuss creating output with the ODS Excel destination. This section explains how you can also enhance web output with the ODS HTML destination. HTML is the most accessible output format because every device or application (for example, email and desktop or mobile browsers) can access to HTML output. This section illustrates ways to power up (amplify) your HTML output within the body of email as well as in desktop and mobile browsers.

SENDING HTML OUTPUT TO THE BODY OF AN EMAIL

Email has become one of the most efficient and effective methods for disseminating information in our world. With the HTML destination, it is even easier to generate HTML content to enhance your email communications.

An easy and powerful way to send content to the body of email and to generate HTML output is to use the EMAIL access method (in the FILENAME statement) in combination with the ODS HTML destination. The *EMAIL access method* in the FILENAME statement enables you to programmatically send email from SAS by using the Simple Mail Transfer Protocol (SMTP).

Example 7 illustrates the combination of email access method and destination.

Example 7

```
options emailsys=smtp emailhost=your-mail-host;
filename temp email to="your-email-address"
                    content_type="text/html";

ods html file=temp rs=none;
ods text="Internal Only.";
```

```
    proc print data=sashelp.class(obs=3);
       title link="http://www.sas.com"  "Link to Detail";
    run;

ods html close;
```

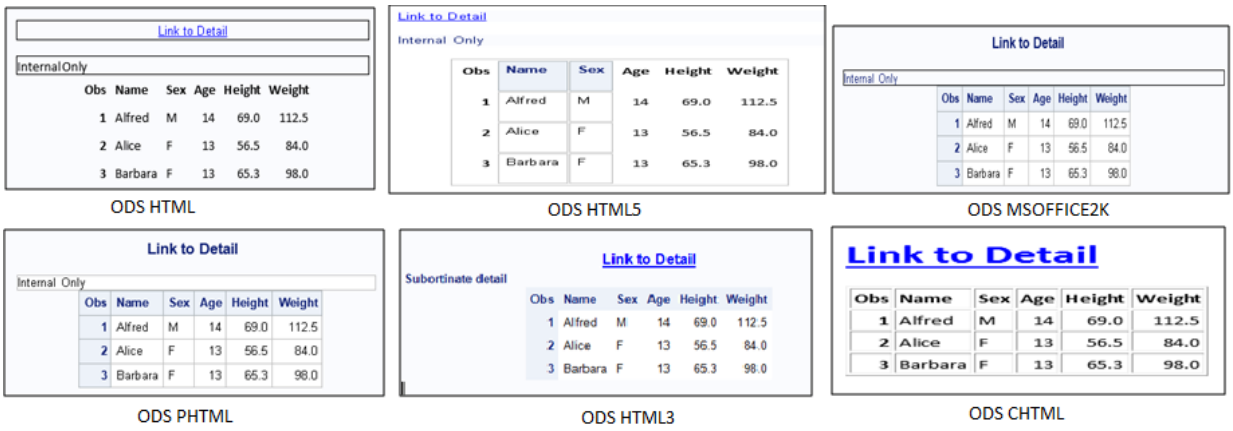
In This Example

- The OPTIONS statement contains two options (EMAILSYS=SMTP and EMAILHOST=*mail-host*) that must be in effect when you send HTML content.
- In the FILENAME statement, you must also set the CONTENT_TYPE= option in order to send HTML to the body of the email.
- The HTML destination passes output as HTML from PROC PRINT to the email server, where the content is added to the body of your email.

Emailing HTML Content: Style Issues

Most web email clients handle styles (and, specifically, cascading styles sheets) well. However, Microsoft Outlook for the desktop does not handle styles that contain CSS because it uses a Word viewer to display the HTML (since Outlook 2007). The use of the Word viewer can cause many issues with the style. The Word viewer can handle HTML 3.2, which is generated with the ODS HTML3 destination. The Word viewer handles this well because it uses HTML tagging (rather than a CSS) to display the style. Some other tagsets also display style information in the body of email (for example, the MSOffice2K tagset, which is HTML that is designed for Excel, and the PHTML tagset).

The HTML destination, on the other hand, does not display the style correctly because multiple class entries in the CLASS= attribute in Outlook for the desktop is not supported (as shown below in Output 8). The destination also adds borders incorrectly in locations such as the title area.



Output 8. Comparison of the Output from Various HTML Destinations and Tagsets within Outlook

Note: The downloadable Tagsets.Email tagset uses minimal CSS and parents from the PHTML tagset. (**ftp.sas.com/techsup/download/base/SGF2018-SAS3388.zip**)

Generating HTML Output: Efficiency of Size

The method that you choose to send HTML to the body of the email must also take into account the size of the output because there are defined size limitations with the email server. The size of the output also determines how long the file can take to load on mobile

devices. So, for example, using the ODS HTML3 destination is not equivalent to using the PHTML destination, not only with regard to what is displayed, but also in the amount of time it takes to load. The HTML3 destination takes almost six times as many characters to display the style and value in a cell. Consider the examples below that create a single cell within a table of the SASHELP.CLASS data set. You can see that the HTML destination takes 30 characters to display the same cell for which the HTML3 destination requires 118 characters. If you have a lot of data, the larger number of characters can cause the email server to reject the mail because of the size limitation. In addition, it might take an excessive amount of time for the content to load on a mobile device.

ODS HTML

```
<td class="l data">Alfred</td>
```
(30 characters)

ODS HTML3

```
<td align=left bgcolor="#D3D3D3"><font face="Arial, Helvetica, sans-serif"
size="3" color="#000000">Alfred</font></TD>
```
(118 characters)

Therefore, if you have large output, you need to consider which destination is best to use in order not to exceed the limit of the email server or to avoid the extended time that it can take to load.

Emailing Images

You can add images to output easily with HTML destinations or tagsets. To do that, you use an identifier on the attachment, which you then refer to in the HTML tag. But you need to decide whether it makes more sense to store the graphic on a web server to reference it or to attach the image directly to the email. The answer to this question depends on the size of the image and how long it will take to load the file.

You should also consider the following caveats when you want to send images:

- Use a compressed-graphic file format (such as JPG, PNG, and GIF) for emails.
- Add descriptive, alternative text in case the email server blocks the images from being viewed.

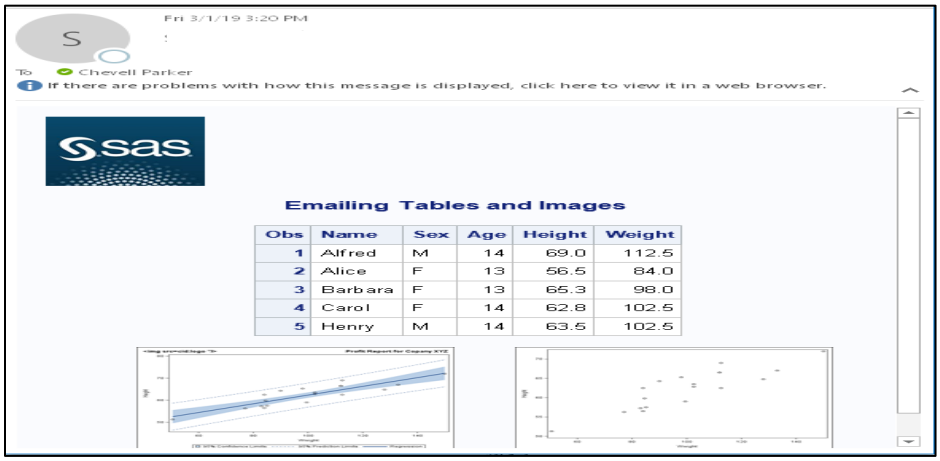
Example 8

```
options emailsys=smtp emailhost=your-email-host-name;
filename output email to="your-email-address"
          attach=('C:\images\SAS.jpg' inlined="logo"
                  'C:\images\sgplot.png'  inlined="logo1"
                  'C:\images\sgplot1.png' inlined="logo2")
          content_type="text/html" ;

ods phtml file=output rs=none style=htmlblue options(pagebreak="no");
   title j=l '<img src=cid:logo height="100" width="100"></img>' ;

   proc print data=sashelp.class(obs=5);
   run;

   proc odstext;
      p "<span style='white-space:pre'>
              <img src='cid:logo1' width=200 height=100/>
              <img src='cid:logo2' width=200 height=150/>
         </span>" / style={just=center};
   run;
ods phtml close;
```

In This Example

- The images are defined with the INLINED= suboption in the ATTACH= option. The images are added by placing a content identifier (`cid:`) in the title, by using the SRC= option in the ODS PHTML statement, by using the P statement in the ODSTEXT procedure.
- PROC PRINT displays the generated table.

Output



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

Output 9. Adding Images and a Table to the Body of Email While Maintaining the Style Information

GENERATING DATA TABLES USING THE TABLEEDITOR TAGSET

When you view a web page, you might want to perform tasks such as sorting content, filtering a web page, freezing headings so that they remain even when you scroll a web page, and so on. The ability to accomplish these tasks have been available with the TableEditor tagset for some time.

Consider the following example, which adds a drop-down box to query the output, limits the number of rows that are to be displayed, or enables you to select which page of the output to display. The following example points to a hosted jQuery library when you add the JQ_DataTable option. But you first have to download and compile the TableEditor tagset.

Example 9

```
ods tagsets.tableeditor path="c:\temp\test" body="test.html"
                          options(JQ_DataTabe="yes");
   proc print data=sashelp.cars noobs label;
   run;
ods tagsets.tableeditor close;
```

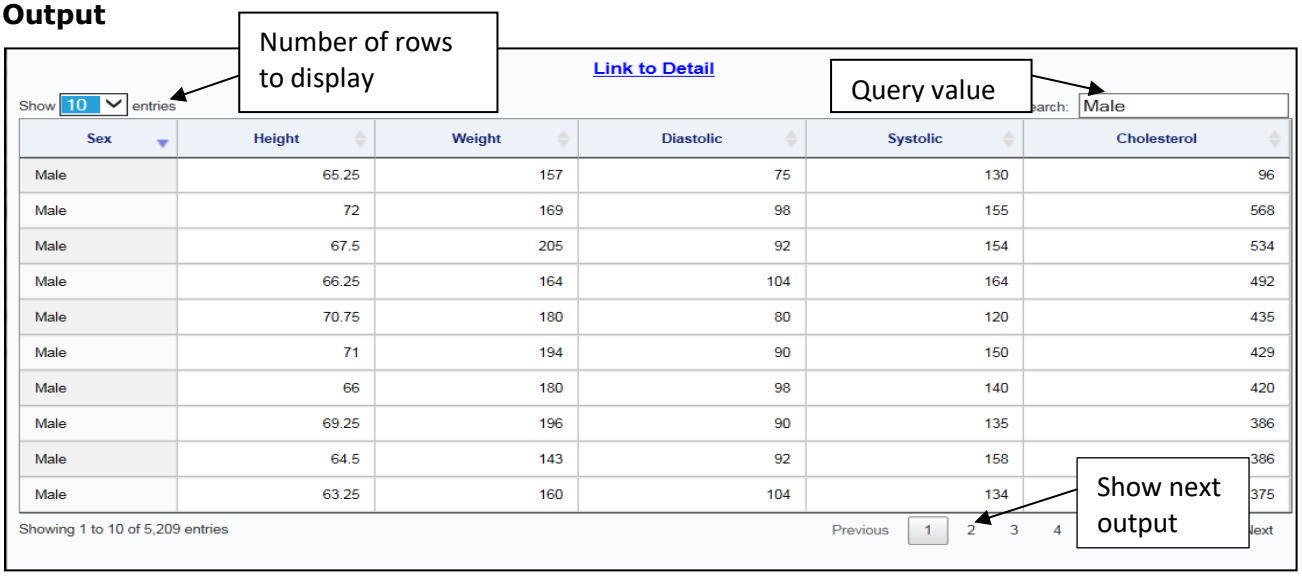
In This Example

The TableEditor tagset is so powerful because it accomplished many tasks:

- The tagset enables you to modify how many rows to displayto display on the web page.
- The tagset also selects records to search for within the table.
- It shows which records are displayed.
- It selects more data to view.

(list continued)

Output



Output 10. Output That Is Added by Using the TableEditor Tagset

TEST DRIVING THE ODS WORD DESTINATION

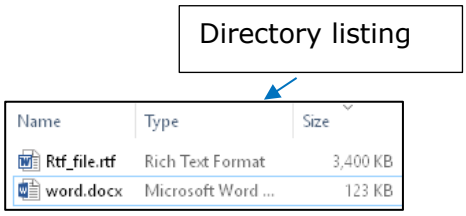
The new ODS WORD destination is pre-production beginning with SAS 9.4M6. This destination enables you to generate native Word files (DOCX).

The ODS WORD destination goes beyond all of the ODS RTF features and adds additional functionality. For example, ODS WORD uses smaller file sizes, uses native Word format, and has the ability to add color schemes. The WORD destination alleviates the file-size issue that is inherent in ODS RTF and the Tagsets.RTF destinations. It does so by using a much smaller file size. The WORD destination is the one that SAS plans to enhance going forward because Microsoft no longer supports the RTF format. Also, notice that ODS WORD uses the new Word style. You can also use the ODS Word destination with mobile devices.

The following example creates a native Word file (.DOCX) by using PROC PRINT. The file size is much smaller than that you can create with the ODS WORD destination. The image on the right displays the directory information on the two files.

Example 10.

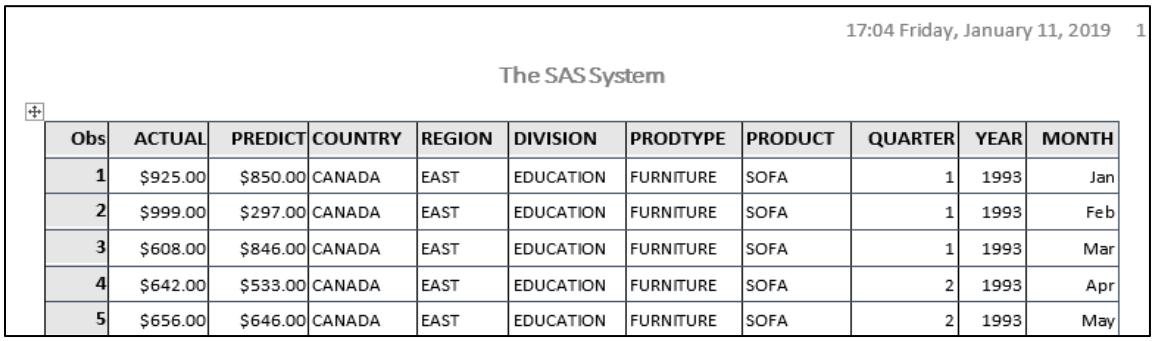
```
ods word file="c:\word.docx";
ods rtf file="c:\rtf_file.rtf";
   proc print data=sashelp.prdsale;
   run;
ods _all_ close;
```



In This Example

- The ODS WORD destination applies style (using the Styles.Word style, which is the destination's default style).
- PROC PRINT prints the output.

Output



17:04 Friday, January 11, 2019 1

The SAS System

Obs	ACTUAL	PREDICT	COUNTRY	REGION	DIVISION	PRODTYPE	PRODUCT	QUARTER	YEAR	MONTH
1	$925.00	$850.00	CANADA	EAST	EDUCATION	FURNITURE	SOFA	1	1993	Jan
2	$999.00	$297.00	CANADA	EAST	EDUCATION	FURNITURE	SOFA	1	1993	Feb
3	$608.00	$846.00	CANADA	EAST	EDUCATION	FURNITURE	SOFA	1	1993	Mar
4	$642.00	$533.00	CANADA	EAST	EDUCATION	FURNITURE	SOFA	2	1993	Apr
5	$656.00	$646.00	CANADA	EAST	EDUCATION	FURNITURE	SOFA	2	1993	May

Output 11. Output from the ODS WORD Destination

ENHANCING YOUR REPORTS USING PYTHON OPEN-SOURCE LANGUAGE

So far, you have learned about how you can use SAS ODS destinations to generate enhanced reports. But you can also use Python open-source language for reporting or to enhance existing reports. Python is integrated totally with the SAS® System in the following ways. You can use **`SASPy`** package, which enables you to execute SAS from Python. The Python API to SAS Viya enables execution of SAS in the cloud from Python. In addition, you can call Python functions with the FCMP procedure in SAS 9.4M6.

This section demonstrates how to use the **`SASPy`** package to enhance reporting from Python. It also demonstrates the **`openpyxl`** package to update existing Excel files. If you do not have Python installed, you can install it from the Anaconda Distribution page: **`www.anaconda.com/distribution`**.

USING SASPY TO GENERATE REPORTS

The **`SASPy`** module provides APIs to SAS. Then, you can connect and run your analytics directly from Python by using the object-oriented methods to which Python customers are accustomed. To use this module, you need a minimum of SAS 9.4 and Python 3.

After you complete the setup (instructions are available in the "SASPY Installation and configuration" document that is listed in the References section), you can connect to a SAS session and begin your analysis. After you connect to SAS, you can use methods or APIs that accepts Python commands and converts them to SAS code and returns the result to Python. You can use the methods in the **`SASPy`** API reference (**`sassoftware.github.io/saspy/api.html`**) to generate your analytics. Because Python is an open-source language, you can also create your own methods.

The next example takes advantage of the power of Python by using the Python **`WordCloud`** package. Word clouds are a good way to visualize or mine text that is based on feedback, tweets, posts, and so on, all in a single glance. The word cloud is created in Python. Then the FREQ procedure is used in SAS to further analyze the data while including the word cloud created in Python. A fully customized Excel spreadsheet is generated from this.

This example uses New York City open-source data to supply the most popular baby names between 2011-2016, by gender and race. The CSV file (available at **`ftp.sas.com/techsup/download/base/SGF2018-SAS3388.zip`**) is mined for that data.

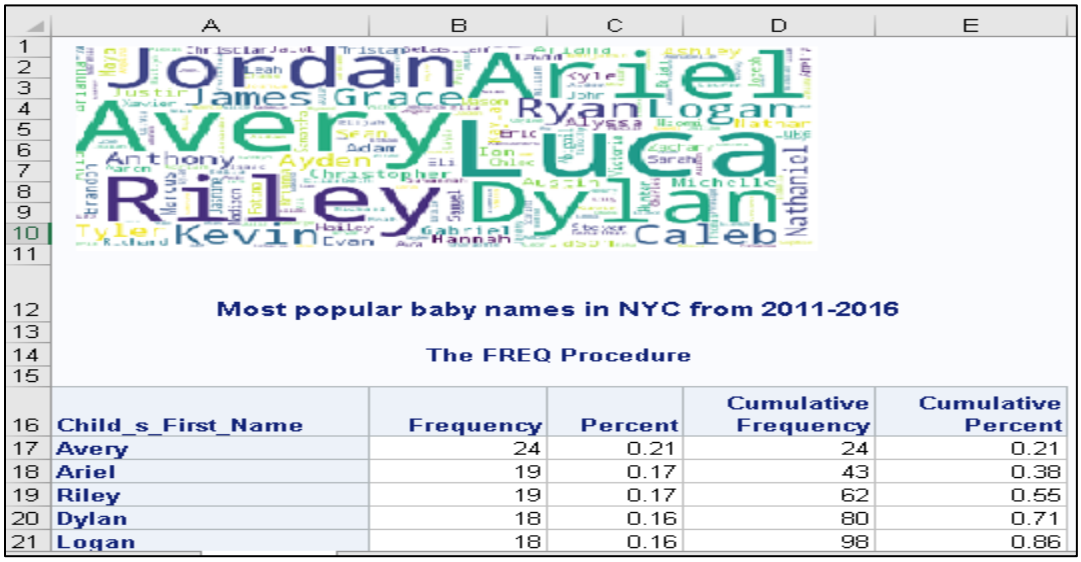
Example 11

```
import pandas as pd
import matplotlib.pyplot as plt
from wordcloud import WordCloud

df= pd.read_csv('https://data.cityofnewyork.us/api/views/25th-
nujf/rows.csv?accessType=DOWNLOAD')
text=' '.join(df["Child's First Name"].tolist())
wordcloud=WordCloud(relative_scaling=1.0,background_color='white').
generate(text)

plt.savefig('c:/temp/wordcloud.png')

import saspy
sas=saspy.SASsession(cfgname'winlocal')
sasds=sas.df2sd(df)

sas.submit(
"""
 ods _all_ close;
 ods excel file="c:\temp\test.xlsx"
           options(embedded_titles="yes" sheet_interval="none");
 goptions iback="c:\temp\wordcloud.png" imagestyle=fit hsize=3.5in
          vsize=3.5in;

    proc gslide;
    run;

    title "Most popular baby names in NYC from 2011-2016";
    proc freq data=_df order=freq;
       table Child_s_First_Name;
    run;
 ods excel close;
 """  )
```

In This Example

- First, the Python packages needed for this example including the WordCloud package is imported.
- The object df is created. This object represents a python pandas DataFrame that is created from reading the CSV file. The *DataFrame is* a one-dimensional array that can hold any data type. In addition, the DataFrame can include axis labels or an index.
- The Python variable text is created concatenating values of 'Child's First Name' into a single string by using the JOIN() function.
- A word cloud is generated by using the text string from the concatenated string (in step 3). Then the image is saved.
- The sas session object is created from a local SAS session in the Microsoft Windows operating environment.
- The sasds SAS data object is created from the DF2SD API, which converts a Python DataFrame to a SAS data set.
- The ODS Excel destination along with the PROC FREQ and GSLIDE pr *(list continued)* added within the Submit methods which generates both the table and the image.

Output



Child_s_First_Name	Frequency	Percent	Cumulative Frequency	Cumulative Percent
Avery	24	0.21	24	0.21
Ariel	19	0.17	43	0.38
Riley	19	0.17	62	0.55
Dylan	18	0.16	80	0.71
Logan	18	0.16	98	0.86

Output 12. Using the Python WordCloud Package with SAS to Create a Word Cloud and Add It to the Excel Spreadsheet

USING PYTHON TO ENHANCE YOUR REPORTING

There are times when you create Excel files (perhaps with PROC EXPORT or the ODS Excel destination) that you need to include additional features on the worksheet (for example, password protection, data validation, and so on). Python has a solution for that issue—the **openpyxl** library (package). The **openyxl** package reads from, writes to, and updates Excel files (XLSX). Python **openpyxl** has defined APIs and methods that you can use to modify every part of an Excel workbook. This section discusses how you can use **openpyxl** to accomplish these tasks.

After you install Python under Windows, you need to install the **openpyxl** package. To do that, submit the following command:

```
$ pip install openpyxl
```

After the package is installed, you can create, write, and update XLSX files. To work with the **openpyxl** package, you first need to know the methods that are available. Those methods are discussed in *openpyxl Documentation: Release 2.4.9* (**media.readthedocs.org/pdf/openpyxl/latest/ssopenpyxl.pdf).**

Methods for Executing the Python Script in the SAS® Session

SAS® software does not have a direct method for executing Python scripts from the SAS session. However, there are a few methods (command pipes, the DATA _NULL_ step, and the SAS Java object) that enable you to do so. The next example uses the PIPE command in the FILENAME statement to execute the Python script. The example also adds a macro variable for the location of the script. This macro variable, _LOC, is resolved in the FILENAME statement.

Example 12

```
%let _loc=C:\users\user\scripts\export.py;
filename temp pipe "C:\users\user\Anaconda3\python.exe &_loc";
```

```
data _null_;
   infile temp;
   input;
   put _infile_;
run;
```

Code for the Remaining Examples

This section presents two separate examples that perform the same task. These programs (one from PROC EXPORT; one from the Excel destination) modify existing XLSX files and add additional functionality.

PROC EXPORT Example

As mentioned earlier, the **openpyxl** library can be used to load existing XLSX files and to include additional functionality. For example, you can add page setup information, frozen headings, or filters.

This example adds worksheet properties to the XLSX file that is created with the following EXPORT procedure.

Example 13

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

This EXPORT procedure exports data to the worksheet for which is used with the below example.

After this PROC EXPORT step, to execute this Python script in SAS, include the code shown in Example 12.

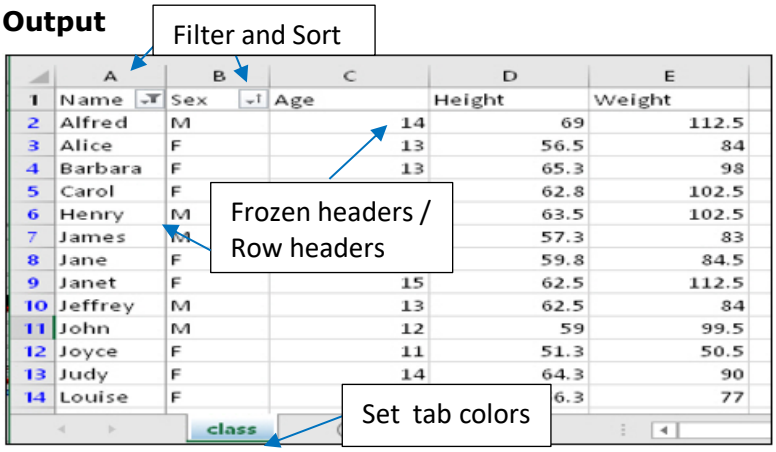
The export.py file (shown below) is executed from the FILENAME and DATA step statements in Example 12.

```
from openpyxl import load_workbook
wb = load_workbook('c:/temp/export.xlsx')

ws=wb.active

ws.freeze_panes='A2'
ws.freeze_panes='b2'

ws.page_setup.orientation = ws.ORIENTATION_LANDSCAPE

ws.auto_filter.ref='A1:B19'
ws.auto_filter.add_filter_column(0,["Alice","Carol","Henry"])
ws.auto_filter.add_sort_condition('B2:B19')

ws.sheet_properties.tabColor="1072BA"
wb.save('c:/temp/export_update.xlsx')
```

In This Example

- The workbook object wb is created from loading the worksheet sheet created with PROC EXPORT.
- The worksheet object ws is created from the first worksheet within the workbook.
- The FREEZE_PANES method is added to the worksheet object to freeze the first row and column.
- Preselected filters and sorting are added to specific columns after being applied to the worksheet object.
- The color of the worksheet tab is modified.

Output



	A	B	C	D	E
1	Name	Sex	Age	Height	Weight
2	Alfred	M	14	69	112.5
3	Alice	F	13	56.5	84
4	Barbara	F	13	65.3	98
5	Carol	F		62.8	102.5
6	Henry	M		63.5	102.5
7	James	M		57.3	83
8	Jane	F		59.8	84.5
9	Janet	F	15	62.5	112.5
10	Jeffrey	M	13	62.5	84
11	John	M	12	59	99.5
12	Joyce	F	11	51.3	50.5
13	Judy	F	14	64.3	90
14	Louise	F		6.3	77

Output 13. Modifying an Existing Excel File by Using Python

Excel Destination Example

The next example modifies .XLSX files that are generated by the ODS Excel destination. The approach for this example is slightly different because of an issue with the **openpyxl** package and how it interacts with files that are created with ODS Excel. To work around this issue, set the environmental variable OPENPYXL_LXML to FALSE. Once this value is set, you can modify the file that was created earlier with the Excel destination.

This example adds the following functionality to the worksheet (for example, adding images, adding an outline, adding password protection, and so on. The following code generates the Excel file that has a title and that starts at row 3, column 3:

Example 14

```
ods excel file="C:\scratch\ods.xlsx" options(embedded_titles="yes
          start_at="3,3");
   proc print data=saleshel.prdsale;
      title "Financial Report for Company XYZ.";
      var country region product prodtype year actual;
   run;
ods excel close;
```

The next piece of code uses the PIPE command in the FILENAME statement to execute the Python script from SAS. This script modifies the XLSX file that uses the **openpyxl** package.

```
%let _loc=C:\users\user\scripts\ods.py;
filename temp pipe "C:\users\user\Anaconda3\python.exe &_loc";
```

(code continued)

```
data _null_;
   infile temp;
   input;
   put _infile_;
run;
```

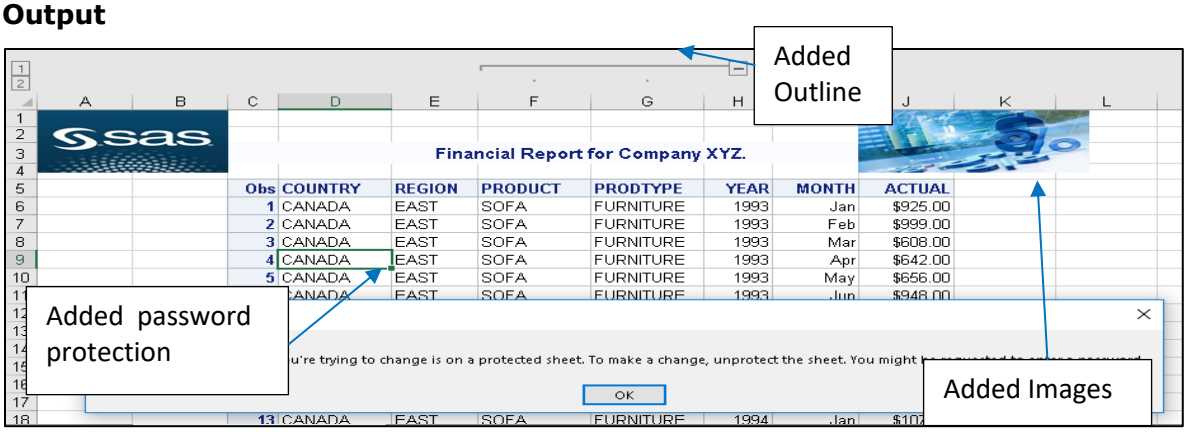
This image shows the ODS.py file that is executed by the code above.

```
from openpyxl import load_workbook
from openpyxl.drawing.image import Image
wb=load_workbook('c:/scratch/ods.xlsx')
ws=wb.active

img=Image('c:/scratch/sas.jpg')
img1=Image('c:/scratch/financial.png')
ws.add_image(img,'a1');
ws.add_image(img1,'j1');
ws.column_dimensions.group("F","G")
ws.protection.set_password('test')

wb.save('c:/scratch/ods_update1.xlsx')
```

In This Example

- The wb object creates a workbook object from the worksheet that is created in the PROC EXPORT step.
- The ws object creates a worksheet object with the first worksheet.
- The ADD_IMAGE method is included to the WS object, which enables images to be added to cells A1 and G1.
- The COLUMN_DIMENSION.GROUP method adds an outline to columns F-G.
- The PROTECTION.SET_PASSWORD method that is used on the ws object sets the password for the worksheet.
- The .SAVE method was added to the workbook object wb to save a new file.

Output



Output 14. The Final Updated Worksheet That Is Created with the ODS Excel Destination

CONCLUSION

SAS ODS destinations offer a wide variety of options and features that enable you to control output features so that you can create high-quality, visually appealing reports. In addition to highly presentational reports, you can also use the destinations to create custom CSV

files, send HTML content to email or the desktop, and generate fully interactive web pages. In addition, you can power up your reports even more by using the flexibility and features in the Python open-source language.

REFERENCES

Gazoni, Eric and Charlie Clark. 2017. *Openpyxl Documentation, Release 2.4.9.* Open-source documentation. Available at **media.readthedocs.org/pdf/ssopenpyxl/2.5.2/ssopenpyxl.pdf**.

Lund, Pete. 2014. "Have It Your Way: Creating Reports with the Data Step Report Writing Interface." *Proceedings of the SAS Global Forum 2018 Conference.* Cary, NC: SAS Institute Inc. Available at **support.sas.com/resources/papers/proceedings14/1362-2014.pdf**.

Openpyxl. 2019. "Openpyxl Tutorial." Available at **openpyxl.readthedocs.io/en/stable/tutorial.html**. Accessed on March 6, 2019.

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**.

SAS Institute Inc. 2018. *SAS® 9.4 Output Delivery System: User's Guide, Fifth Edition*. Cary, NC: SAS Institute Inc. Available at **https://go.documentation.sas.com/api/collections/pgmsascdc/9.4_3.4/docsets/odsug/content/odsug.pdf?locale=en#nameddest=titlepage**.

SAS Institute Inc. 2018. SASPy 2.4.3: Installation and Configuration. Cary, NC: SAS Institute Inc. Available at **sassoftware.github.io/saspy/install.html.**

ACKNOWLEDGMENTS

I would like to thank Susan Berry for her assistance 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/en/support-home.html**