Paper 2444-2018

Excelling to Another Level with SAS®

Arthur S. Tabachneck, Ph.D., AnalystFinder, Inc.; Tom Abernathy, Pfizer, Inc.; Matthew Kastin, NORC at the University of Chicago

ABSTRACT

Have you ever wished that with one click (or submitting one macro call) you could copy any SAS® dataset, include or exclude variable names or labels, automagically create or modify Excel workbooks, or paste tables into files? Or how about doing any of the above but, at the same time, create Pivot Tables and/or, if desired, base new worksheets on existing Excel templates or preformatted workbooks so that you don't have to spend time duplicating such things as font usage, effects, formulas, highlighting and/or graphs? You can and, with just base SAS, there are some little known but easy to use methods that are available for automating many of your (or your users) common tasks.

BACKGROUND

With the introduction of 64-bit computers, and the XLSX Excel Workbook, the task of exporting SAS® datasets to Excel became more difficult, typically requiring the purchase of additional products (like SAS/Access for PC File Formats) and/or installing the PC Files Server. Running macros that automatically write and run the necessary Visual Basic script, and take advantage of your system's clipboard, not only provides a free alternative, but one that runs faster and brings in capabilities that simply aren't available with other methods.

In 2014 the current paper's authors took on the challenge of writing a SAS macro that could do all of the things that PROC EXPORT can do with Excel workbooks, but do them without needing a SAS/Access product, and do them faster than using either PROC EXPORT or any of the available ODS or tagset alternatives (Tabachneck, Abernathy and Kastin, 2014). The macro and paper were so well received that we were encouraged to expand the macro's capabilities even further. However, the current version of the macro ended up having so many new capabilities that we felt a new publication was in order.

Since some like point-and-click environments, while others don't, the macro was written so that it can be run both ways (i.e., either submit code that closely resembles what one would submit to run PROC EXPORT, or right click on a dataset name in SAS Explorer, and select a task from the menu). In addition to the actions you currently see when right clicking on a dataset, SAS Explorer lets users add their own actions to the menu. This paper provides step-by-step instructions regarding how to add hundreds of different export actions designed to your own specifications.

SAS EXPLORER

Many SAS users may not know about the *Copy Contents to Clipboard* because it is only briefly mentioned in the documentation. We discovered it while writing the original paper which, initially, was only going to offer a PROC EXPORT alternative using the CLIPBOARD Access Method. In the documentation's description of the CLIPBOARD Access Method, the following statement can be found: *You can also copy data to the clipboard by using the Explorer pop-up menu item* ***Copy Contents to Clipboard***.

Any attempt to explain features of the SAS Windowing environment is complicated by the fact that users can alter the placement and presence of the various windows that will appear on their monitors when they open an interactive SAS session. A screenshot of a typical SAS Windowing environment is shown, on the following page, in Figure 1.

The SAS Explorer window, shown on the lower left side of the configuration shown in Figure 1, is the window from which one can view all active SAS libraries and their contents. The screenshot shown in Figure 1, specifically, is the way the screen would appear if one has traversed through the SAS Explorer

menu to show all of the tables (i.e., SAS datasets) and catalogs that exist in the Sashelp library. Right clicking on the *name* of any table shown in that window will cause the pop-up menu, like the one shown in Figure 2, to appear.

Moving one's mouse up and down through the pop-up menu makes the various options active. In our example, *Copy Contents to Clipboard* is *active* and, if one left-clicks while an option is active, they will cause the action to occur. This particular action creates an HTML version of the dataset and places that version within one's clipboard so that it can be pasted into an Excel workbook or any other program that includes a paste feature.

Figure 1

One Way a SAS Windowing Environment May Appear

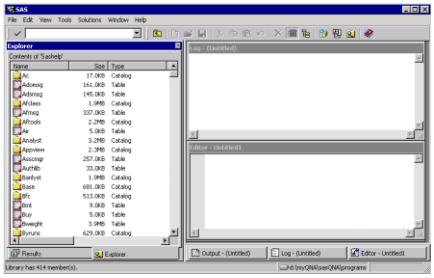
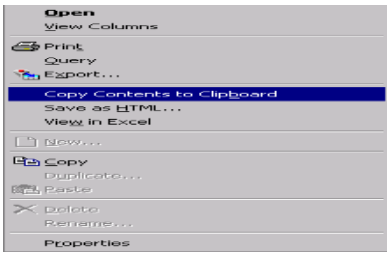


Figure 2

An Explorer Pop-up Menu



ADDING YOUR OWN MENU ACTIONS

As mentioned earlier, we discovered the *Copy Contents to Clipboard* action during our review of the documentation pertaining to the Clipboard Access Method. We could have stopped as soon as we discovered that the action was already available, but we didn't as the feature didn't quite meet our specifications. Specifically, the existing action runs quite slowly, produces HTML files that include possibly undesired header records and highlighted borders around the cells, and only copies the files to your system's clipboard. We were interested in creating actions that could create Excel workbooks and provide more capabilities than Proc Export currently provides, including:

- producing Excel workbooks with variable names in the first row
- producing Excel workbooks with variable names in the first row and letting you indicate the upper left cell where the tables should begin
- producing Excel workbooks without including variable names in the first row
- producing Excel workbooks, without variable names in the first row, and letting you indicate the upper left cell where the table should begin

letting you indicate whether formatted values should or shouldn't be used

producing Excel workbooks that are based on either Excel templates or other workbooks

including Pivot Tables in any workbooks that are created

copying SAS datasets, with or without variable names or labels, to your system's clipboard

modifying existing worksheets leaving all formatting intact and not altering non-affected cells

The current paper's authors were already familiar with adding SAS Explorer menu options (Tabachneck, et al, 2010). However, other than our own paper, and excellent papers by Richard DeVenezia and Art Carpenter, little documentation even mentions the capability. However, creating a pop-up action is fairly straight forward:

From the Explorer window, select Tools ➧ Options ➧ Explorer

Left click on Members

Double left click on Table

Left click on Add.

Enter a name for the action (i.e., the name you want to appear on the pop-up menu). Including an ampersand before one of the letters will cause the letter to be a shortcut one can use to select the action

Enter the command you want run whenever the action is selected

Select OK to exit the Add Action screen.

Select OK to exit the Table Options screen

CREATING AN ACTION COMMAND

An easy way to declare an action command is to use the *gsubmit* command. That is, if the action begins with the string *gsubmit*, followed by a space, followed by a single quote, everything after the quote (until the next single quote is found) will be submitted. You can submit any SAS code as long as the code uses 255 or fewer characters, uses two % signs whenever one is normally needed and, if quotation marks are needed, either uses masking functions or very carefully balanced double quotation marks.

However, since one can use the gsubmit command to submit a SAS macro, the 255 character limitation can be easily circumvented as only the characters used in calling the macro are counted. Of course, calling a macro in such a manner can only work if you had saved the macro in a directory that has been specified in your system's *autocall* path. A nice description of the various ways that one can ensure that their macros will be found is described in the following paper by Harry Droogendyk: http://analytics.ncsu.edu/sesug/2008/SBC-126.pdf).

An easy way to accomplish that task is described in the Knowledge Base Samples & SAS Notes' Usage Note 24451: http://support.sas.com/kb/24/451.html. Quoting from that note, "An AUTOCALL library on the PC is simply a directory that contains non-compiled MACRO code. The directory does not need to be in a specific location or to have a specific name. However, the MACRO code that is stored in the directory needs to be stored in a file that has the same name as the MACRO, and it needs to have a sas extension." Using that method, you only have to modify the

–SET SASAUTOS line in your SASV9.CFG file to point to the directory.

THE EXPORTXL MACRO

The macro uses Visual Basic Script (i.e., VBS) to achieve actions that currently aren't available with PROC EXPORT or any of the Excel-related DBMS engines, ODS methods, or tagsets. Specifically, the macro provides the means to accomplish such tasks as: (1) creating a new workbook without confronting

32/64 bit collisions; (2) adding a new worksheet to an existing workbook; (3) adding a data table to an existing worksheet; (4) exporting a table to a specific but not predefined range within a worksheet; (5) including or excluding a variable name header record; (6) enabling the choice of variable names or variable labels in any variable name header record; (7) using an Excel template or workbook to preformat the resulting worksheets from any of the above operations; (8) creating pivot tables during any export; (9) the ability to work on both stand alone and server environments; and (10) exporting to your system's clipboard so that you can paste any file into a Word document, Powerpoint, or any software that includes paste functionality.

Named Parameters. The macro uses named parameters. We attempted, as closely as possible, to use the same option names and statements as those used for PROC EXPORT. When calling the macro, the default values will be used unless you include the parameter and desired value. The macro's declaration is shown in the following statement:

```
%macro exportxl( data=, outfile=, sheet=,type=N, usenames=Y, range=A1,
    template=, templatesheet=, useformats=N, usenotepad=N, pivot=);
```

data is the parameter to which you would assign the one or two-level filename of the dataset that you want to export. Like with PROC EXPORT, dataset options can be included. If you assign a one-level filename, the libname *work* will be used. When the macro is called as an action from a SAS Explorer window, this parameter should be set to equal: `data=%8b.%32b` which the macro will interpret as *libname.filename* of the file that was selected.

outfile is the parameter to which you would assign the name of the file that you want the macro to either create or modify. The file's path must exist before the macro is run. Any one of the following values can be used:

Any valid filename (including path) for which you have write access

Null The output file will be created using the selected data file's pathname, a back slash, the selected data file's filename, and the xlsx extension

W Provide window for user input during run

sheet is the parameter to which you would assign the name of the worksheet that you want to either create or modify. Any one of the following values can be used:

Any valid worksheet name

Null The filename of the data parameter

W Provide window for user input during run

type is the parameter you would use to indicate the type of process that you want to run. The default value of this parameter is: N. Any one of the following values can be used:

N Create a new workbook

A Add a new worksheet to an existing workbook

M Modify an existing worksheet

C Copy the dataset to your system's clipboard

usenames is the parameter you would use to indicate whether you want the first row of the range to contain the first data record, the variable names, or the variable labels. The default value of this parameter is: Y. Any one of the following values can be used:

N Don't include a variable name row

Y You want the top most row to contain variable names

L You want the top most row to contain variable labels

W Provide window for user input during run

range is the parameter you would use to indicate the upper left cell where you want the table to begin. The default value is: A1 Any one of the following values can be used:

Any valid Excel cell name

W Provide window for user input during run

template If you have a preformatted Excel template(or workbook) that matches the dataset you are exporting, use this parameter to specify the template's path and filename (e.g., template=d:\art\template.xltx). Any one of the following values can be used:

Any existing workbook or template filename (including path)

Null No template is to be used

W Provide window for user input during run

templatesheet If you include the *template* parameter you *must* use this parameter to specify the Worksheet that contains the template you want to apply (e.g., templatesheet=template). Any one of the following values can be used:

Any existing worksheet name

W Provide window for user input during run

useformats is the parameter you would use to indicate whether you want a dataset's formats applied when you are exporting its data. The default value of this parameter is: N. Any one of the following values can be used:

Y You want a dataset's formats to be applied

N You don't want a dataset's formats to be applied

W Provide window for user input during run

usenotepad If you're running this macro on a system that doesn't provide direct access to your computer's clipboard (e.g., a server), or have an Excel configuration that clears the clipboard upon opening, set this parameter to equal 'Y'. You should only use this parameter if you meet one of the above conditions, as it is less efficient than the method used for this parameter's default value (i.e., N). Any one of the following values can be used:

N Don't use Notepad

Y Use your system's or server's version of Notepad

pivot If you want a pivot table created, this parameter should contain a space separated list of all of the class variables you want to use as class variables, followed by another space, and the name of the analysis variable. Any one of the following values can be used:

Space separated list of variable names

W Provide window for user input during run

Of course, some of the options might be meaningless for some of the types. For example, it would be meaningless to define a range if the task were to copy a dataset to your computer's clipboard. The following call would create a Workbook called c:\temp\class.xlsx, from sashelp.class, including a worksheet called *class*:

```
%exportxl(data=sashelp.class, outfile=c:\temp\class.xlsx)
```

USING EXCEL TEMPLATES

An Excel template worksheet is shown, below, in Figure 3. As you can see, an Excel template is simply an Excel workbook that has been saved as an Excel template, thus either has an xltx or xltm extension. With the *exportxl* macro you can specify any Excel template, or existing Excel workbook (i.e., xls or xlsx file), as a template The template shown in Figure 3, which we created as c:\art\template.xltx, has variable names formatted using a different font than the cells under them, as well as a different color for each variable name, and various highlighting to better differentiate each row.

Cells H19:J20 contain formulas that calculate the average age, height and weight for males and females, respectively. Finally, cells F14:LG15 contain formulas, while cells D1:G10 and cells J1:M10 contain line charts that will compare the average heights and weights of the males and females included in the data.

Figure 3
An Excel Template

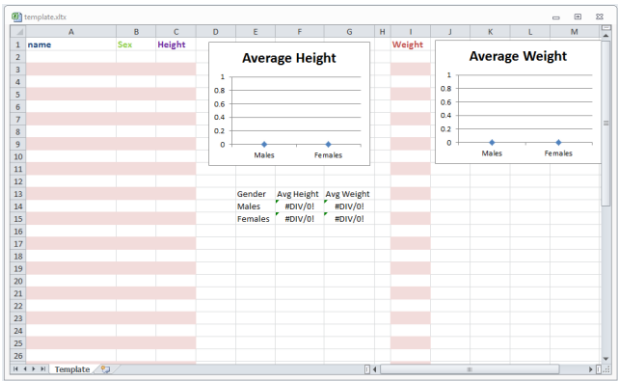
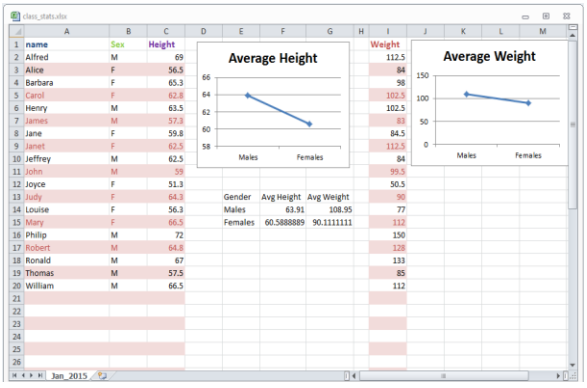


Figure 4
Desired Resulting Workbook



If your requirement was to produce a workbook (c:\temp\class_stats) that contained one worksheet labeled Jan_2015, using the template shown in Figure 3 and populated with the sashelp.class dataset, the two calls to the exportxl macro shown below would readily accomplish the result shown in Figure 4.

```
%exportxl( data=sashelp.class (keep=name sex height),
           template=c:\temp\template.xltx,
           templatesheet=Template,
           outfile=c:\temp\class_stats.xlsx,
           type=N, usenames=N,
           range=A2,
           sheet=Jan_2015)

%exportxl( data=sashelp.class (keep=weight),
           outfile=c:\temp\class_stats.xlsx,
           type=M, usenames=N, range=I2,
           sheet=Jan_2015)
```

In the above code the first call of the macro uses the *sashelp.class* dataset with the *Template* worksheet in the file c:\temp\template.xltx to create a new workbook (type=N) called *c:\temp\class_stats*, with one worksheet labeled *Jan_2015*, copying the students' names, sex and heights beginning at cell A2 (range=A2) and without including a variable name header row (usenames=N).

The second call of the macro modifies (type=M) the worksheet by adding the students' weights beginning at cell I2 (range=I2), again without including variable names.

As you can see, in Figure 4, the template's highlighting, formatting, and charts were retained, and the worksheet was populated with the desired data.

Exporting a Pivot Table. A tutorial on pivot tables is well beyond the scope of the present paper. Besides, numerous and extremely good explanations can be easily found doing a simple web search. Thus, instead, we'll show how the macro can be used to create one, and explain the resulting file. Of course, the macro's ability to create pivot tables is not the only way to accomplish the task in SAS. If one licenses SAS/Access to Microsoft Office, Excel's SAS menu will have an option to open a SAS dataset as a Pivot table. Alternatively, without SAS/Access to Microsoft Office you could download, modify and apply the ODS tagsets.tableeditor.

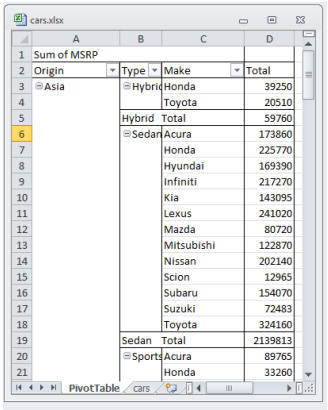
We included the capability to create pivot tables because we prefer extensible code driven solutions. Both of the above mentioned options require manual point-and-click steps to accomplish the task. To get the macro to create a pivot table one only has to include the pivot parameter as described in the parameter section earlier in this paper. As an example, if we wanted to create a Pivot Table based on the sashelp.cars file including the origin, type and make variables as classification columns and the MSRP variable as the column being analyzed, the entire task would be accomplished with the following macro call:

```
%exportxl(data=sashelp.cars,outfile=c:\cars.xlsx,pivot=Origin Type Make
MSRP)
```

The first few rows of the resulting workbook are shown in Figure 5, below.

Figure 5

Workbook Created using the Pivot Parameter



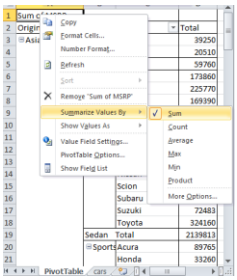
	A	B	C	D
1	Sum of MSRP			
2	Origin	Type	Make	Total
3	Asia	Hybrid	Honda	39250
4			Toyota	20510
5		Hybrid	Total	59760
6		Sedan	Acura	173860
7			Honda	225770
8			Hyundai	169390
9			Infiniti	217270
10			Kia	143095
11			Lexus	241020
12			Mazda	80720
13			Mitsubishi	122870
14			Nissan	202140
15			Scion	12965
16			Subaru	154070
17			Suzuki	72483
18			Toyota	324160
19		Sedan	Total	2139813
20		Sports	Acura	89765
21			Honda	33260

As you can see, the table automatically created two worksheets, one called *cars*, and the other called *Pivot Table*. The *cars* worksheet is simply a worksheet version of the sashelp.cars dataset. The *PivotTable* worksheet's rows are sums of the MSRPs for each Origin/Type/Make combination, as well rows tabulating the total MSRPs for each Origin/Type combination, total MSRPs for each Origin combination, and one for the grand total.

If you right click on cell A1 (Sum of MSRP), you can select whether the summarized measures of MSRP will reflects sums, counts, averages, maximum values, minimum values, products, counts, standard deviations or variances.

Figure 6

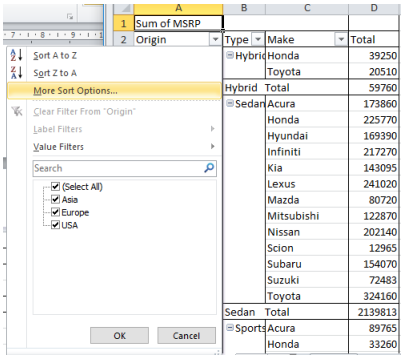
Example of Pivot Table Summary Options



If you left click on a down arrow immediately to the right of any of the categorical variables, you can select which values you want to either include or exclude, as well as how you want that variable sorted.

Figure 7

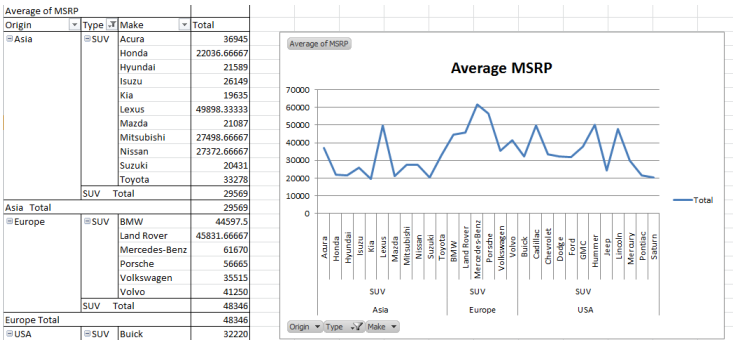
Example of Pivot Table Sort and Selection Options



Additionally, if you or your users insert a graph, the graph will automatically adjust to the included classification values and type of summary selected.

Figure 8

Example of Pivot Table Graph



We totally agree that Pivot Tables don't provide anything you can't already do with SAS, but it does give those same capabilities to others who either aren't proficient in or don't have access to SAS, or are more comfortable working with Excel.

HOW THE MACRO WORKS

The macro's declaration line was shown, earlier, in the *Named Parameters* section. The initial lines of the macro's code are shown on the following page. These lines accomplish five tasks. First, if a two-level filename is provided for the *data* parameter, the macro variable is parsed into two new macro variables, namely *libnm* and *filenm*. Conversely, if the *data* parameter contains a one-level filename, that name is assigned to the *filenm* macro variable and the *libnm* macro variable will be set to equal *work*.

The code also inspects the data parameter's value to determine if it contains a left parenthesis. If it does, any data step options that were included are parsed out of the data parameter, and used as the value of a macro variable called *&dsoptions*.

Then, if the *outfile* parameter isn't specified, the outfile macro variable is set to equal the path of *libname*, followed by a backslash, followed by the *filenm*, and ending with an *.xlsx* extension:

```
%let lp=%sysfunc(findc(%superq(data),%str(%())); 
%if &lp. %then %do;
  %let rp=%sysfunc(findc(%superq(data),%str(%)),b));
  %let dsoptions=%qsysfunc(substrn(%nrstr(%superq(data)),&lp+1,&rp-&lp-1));
  %let data=%sysfunc(substrn(%nrstr(%superq(data)),1,%eval(&lp-1)));
%end;
%else %let dsoptions=;
%if %sysfunc(countw(&data.)) eq 2 %then %do;
  %let libnm=%scan(&data.,1);
  %let filenm=%scan(&data.,2);
%end;
%else %do;
  %let libnm=work;
  %let filenm=&data.;
%end;

%if %length(&outfile.) lt 1 %then
  %let outfile=%sysfunc(pathname(&libnm.))\&filenm..xlsx;;
%if %length(&sheet.) lt 1 %then %let sheet=&filenm.;;
```

The next section of the code gives end users a way to enter the *outfile*, *sheet name*, *range*, *template*, *templatesheet*, *useformats*, *usenotepad*, and *pivot* parameters, interactively, after the macro has been called. For example, the following code is used to produce a window that lets end users interactively specify the range parameter:

```
%else %do;
  %if %upcase(&range.) eq W %then %do;
    data _null_;
     window range rows=8 columns=80
     irow=1 icolumn=2 color=black
     #2 @3 'Enter the upper left cell where range should begin (e.g. D5): '
      color=gray range $41. required=yes
      attr=underline color=yellow;
```

```
    DISPLAY range blank;
    call symputx('range',range);
    stop;
   run;
%end;
%else %if %length(&range.) lt 2 %then %do;
  %let range=A1;
%end;
```

The above code, if invoked with the specified parameter values, will cause the macro to display windows like the one shown in Figure 9, below. The window will appear on the user's monitor and wait until a cell name has been entered and the Enter key has been pressed.

Figure 9

User Input Window Created by Window Statement

The next section of the macro checks to ensure that the data dataset exists and, if it doesn't, stops the macro from proceeding any further:

```
data _null_;
  dsid=open(catx('.',"&libnm.","&filenm."));
  if dsid eq 0 then do;
    rc=close(dsid);
    link err;
  end;
  rc=close(dsid);
  err:
  do;
    m = sysmsg();
    put m;
    stop;
  end;
run;
```

If the data dataset exists the macro then creates a one-record version of the file, and uses PROC SQL to create a number of macro variables from *dictionary.columns*:

```
data t_e_m_p;
  set &libnm..&filenm. (%unquote(&dsoptions.) obs=1);
run;

proc sql noprint;
  select name,length,type,format,label
    into :vnames separated by "~",
         :vlengths separated by "~",
         :vtypes separated by "~",
```

```
          :vformats separated by "~",
          :vlabels separated by "~"
       from dictionary.columns
         where libname="WORK" and
               memname="T_E_M_P"
  ;
quit;
%let nvar=&sqlobs.;
```

The following section of code serves as a code generator to incorporate all of the information obtained from the macro variables within an include file that will be called in the macro's main data step:

```
filename code2inc temp;
data _null_;
  file code2inc;
  length script $80;
  length fmt $32;
  do i=1 to &nvar;
    if i gt 1 then put 'rc=fput(fid,"09"x);';
    %if %upcase(&useformats.) eq Y %then %do;
      fmt=scan("&vformats.",i,"~","M");
    %end;
    %else call missing(fmt);;
    if scan("&vtypes.",i,"~") eq 'char' then do;
      if missing(fmt) then
       fmt=catt('$',scan("&vlengths.",i,"~","M"),'.');
      script=catt('rc=fput(fid,putc(put(',
       scan("&vnames.",i,"~","M"),',',fmt,"),'$char",
       scan("&vlengths.",i,"~","M"),'.'));");
      put script;
    end;
    else do;
      if missing(fmt) then fmt='best32.';
      script=catt('rc=fput(fid,putc(put(',
       scan("&vnames.",i,"~","M"),',',fmt,"),'$char32.'));");
      put script;
    end;
  end;
  put 'rc=fwrite(fid);';
run;
```

The next section of code checks to ensure that the dataset has both variables and records after the specified dataset options have been applied:

```
data _null_;
  dsid=open("work.t_e_m_p");
  rc=attrn(dsid,'any');
  if rc ne 1 then do;
    rc=close(dsid);
    link err;
  end;
  rc=close(dsid);
  err:
  do;
```

```
      m = sysmsg();
      put m;
      stop;
    end;
 run;
```

The next section of code serves as the main driver for reading all of the data and writing it to the system clipboard. . The SAS documentation suggests a much simpler data step approach for accomplishing the same task but, when we tested the suggested method, it would only copy a dataset's first 256 characters. The only way we could discover for getting around that limitation was to accomplish the task using functions rather than statements. While the approach uses more code than the straight forward clipboard access method, all of the required code is created automatically by the datastep. Additionally, dependent upon the value of the *usenotepad* parameter, one of two different access methods are incorporated:

```
%if %upcase(&usenotepad) eq Y %then %do;
  %let server_path=%sysfunc(pathname(work));
  rc=filename('clippy',"&server_path.\clip.txt",'DISK');
%end;
%else %do;
  rc=filename('clippy',' ','clipbrd');
%end;
if rc ne 0 then link err;
fid=fopen('clippy','o', 0,'v');
if fid eq 0 then link err;
do i = 1 to &nvar.;
  %if %upcase(&usenames.) ne N %then %do;
    if i gt 1 then rc=fput(fid,'09'x);
    %if %upcase(&usenames.) eq Y %then %do;
      rc=fput(fid,scan("&vnames.",i,"~","M"));
    %end;
    %else %do;
      if missing(scan("&vlabels.",i,"~","M")) then
        rc=fput(fid,scan("&vnames.",i,"~"));
      else rc=fput(fid,scan("&vlabels.",i,"~","M"));
    %end;
  %end;
end;
%if %upcase(&usenames.) ne N %then %do;
  rc=fwrite(fid);;
%end;
do until (lastone);
  set &libnm..&filenm.
  %if %length(%unquote(&dsoptions.)) gt 2 %then (%unquote(&dsoptions.));
  end=lastone;
  %include code2inc;
end;
rc=fclose(fid);
rc=filename('clippy');
rc=filename('code2inc');
stop;

err:
do;
  m = sysmsg();
  put m;
```

```
    rc=filename('code2inc');
    stop;
  end;
run;
```

The next section of the macro was designed to accommodate those cases where a user sets the parameter *type* to a value of N, M or A, indicating that they want the macro to use VB script to either create an Excel workbook, modify an existing worksheet, or add a new worksheet to an existing workbook:

```
%if %upcase(&type.) eq N or %upcase(&type.) eq M
 or %upcase(&type.) eq A %then %do;
```

The next section of the macro creates and runs a file called PasteIt.vbs, in the user's work directory, which contains VB script. Of course, this section of the macro can only run on an operating system, like Windows, that can run both VB script and Excel. First, the file is declared and the code common to all types is specified:

```
data _null_;
  length script filevar $256;
  script = catx('\',pathname('WORK'),'PasteIt.vbs');
  filevar = script;
  script="'"||'cscript "'||trim(script)||'"'||"'";
  call symput('script',script);
  file dummy1 filevar=filevar recfm=v lrecl=512;

  put 'Dim objExcel';
  put 'Dim Newbook';
```

The next section of the macro creates the vbs script to use a template to add a worksheet to an existing workbook:

```
  %if %length(&template.) gt 1 %then %do;
    %if %upcase(&type.) eq A %then put 'Dim OldBook';;
    put 'Set objExcel = CreateObject("Excel.Application")';
    put 'objExcel.Visible = True';
    %if %upcase(&type.) eq N %then %do;
      script=catt('Set Newbook = objExcel.Workbooks.Add("',
       "&template.",'")');
      put script;
      script=catt('objExcel.Sheets("',"&TemplateSheet.",
       '").Select');
      put script;
      script=catt('objExcel.Sheets("',"&TemplateSheet.",
       '").Name = "',"&sheet.",'"');
      put script;
      put 'objExcel.Visible = True';
      script=catt('objExcel.Sheets("',"&sheet.",
       '").Range("',"&range.",'").Activate');
      put script;
```

Then, if the *usenotepad* parameter has a value of y or Y, the code is temporarily rerouted to a section (described later in the code) that creates the vb script needed to accomplish a copy operation using one's system's Notepad. A %goto statement is used to simulate the same action that would be accomplished by a link statement in non-macro code:

```
      %if %upcase(&usenotepad) eq Y %then %do;
        %let Return_to = Return_1;
        %goto use_npad;
        %Return_1:
      %end;
```

Then, the vb script needed to accomplish the paste operation is written:

```
      script=catt('objExcel.Sheets("',"&sheet.",'").Paste');
      put script;
      script=catt('objExcel.Sheets("',"&sheet.",
       '").Range("A1").Select');
      put script;
      put 'objExcel.DisplayAlerts = False';
      script=catt('NewBook.SaveAs("',"&outfile.",'")');
      put script;
    %end;
```

The next section of the macro creates the vb script needed for using a template to create a new workbook:

```
    %else %do;
      script=catt('strFile="',"&outfile.",'"');
      put script;
      script=catt('Set OldBook=objExcel.Workbooks.Open("',
       "&outfile.",'")');
      put script;
      script=catt('Set Newbook = objExcel.Workbooks.Add("',
       "&template.",'")');
      put script;
      script=catt('objExcel.Sheets("',"&TemplateSheet.",
       '").Select');
      put script;
      script=catt('objExcel.Sheets("',"&TemplateSheet",
       '").Name ="',"&sheet.",'"');
      put script;
      put 'objExcel.Visible = True';
      script=catt('objExcel.Sheets("',"&sheet.",
       '").Range("',"&range.",'").Activate');
      put script;
      %if %upcase(&usenotepad) eq Y %then %do;
        %let Return_to = Return_2;
        %goto use_npad;
        %Return_2:
      %end;
      script=catt('objExcel.Sheets("',"&sheet.",'").Paste');
      put script;
      script=catt('objExcel.Sheets("',"&sheet.",
       '").Range("A1").Select');
      put script;
      script=catt('objExcel.Sheets("',"&sheet.",
       '").Move ,OldBook.Sheets( OldBook.Sheets.Count )');
      put script;
      put 'objExcel.DisplayAlerts = False';
      script=catt('OldBook.SaveAs("',"&outfile.",'")');
      put script;
    %end;
  %end;
```

The next section of the macro creates the vb script declarations needed for all other operations:

```
    %else %do;
      %if %upcase(&type.) eq N or %upcase(&type.) eq A %then %do;
        %if %upcase(&type.) eq N %then put 'Dim NewSheet';;
        put 'Dim inSheetCount';
        %if %upcase(&type.) eq A %then put 'Dim strFile';;
      %end;

      put 'Set objExcel = CreateObject("Excel.Application")';
```

Then, the macro creates the VB script needed to create a new workbook. Basically, the code simply causes Excel to create a new workbook, deletes all of the default worksheets that are specified in the user's Excel options, and then adds a new worksheet with the name specified by the *sheet* parameter. Finally, Excel's SaveAs feature is used to save and name the resulting workbook. While other methods could have been used, for convenience we selected this method:

```
%if %upcase(&type.) eq N %then %do;
  put 'Set Newbook = objExcel.Workbooks.Add()';
  put 'objExcel.Visible = True';
  put 'inSheetCount = Newbook.Application.Worksheets.Count';
  script=catt('set NewSheet = Newbook.Sheets.Add',
   '( ,objExcel.WorkSheets(inSheetCount))');
  put script;
  put 'objExcel.DisplayAlerts = False';
  put 'i = inSheetCount';
  put 'Do Until i = 0';
  put ' Newbook.Worksheets(i).Delete';
  put ' i = i - 1';
  put ' Loop';
  script=catt('Newbook.Sheets(1).Name="',"&sheet.",'"');
  put script;
  script=catt('Newbook.Sheets("',"&sheet.",'").Select');
  put script;
  script=catt('Newbook.Sheets("',"&sheet.",
   '").Range("',"&range.",'").Activate');
  put script;
  %if %upcase(&usenotepad) eq Y %then %do;
    %let Return_to = Return_3;
    %goto use_npad;
    %Return_3:
  %end;
  script=catt('Newbook.Sheets("',"&sheet.",'").Paste');
  put script;
  script=catt('NewSheet.SaveAs("',"&outfile.",'")');
  put script;
%end;
```

The next section of the macro's code creates the VB script needed to add a new worksheet to an existing workbook. Again, Excel's SaveAs feature is used to save and name the resulting workbook.

```
%else %if %upcase(&type.) eq A %then %do;
  script=catt('strFile="',"&outfile.",'"');
  put script;
  put 'objExcel.Visible = True';
```

```
    put 'objExcel.Workbooks.Open strFile';
    put 'inSheetCount = objExcel.Application.Worksheets.Count';
    script=catt('set NewBook = objExcel.Sheets.Add( ,objExcel.',
     'WorkSheets(inSheetCount))');
    put script;
    script=catt('objExcel.Sheets(inSheetCount + 1).Name="',
     "&sheet.",'"');
    put script;
    script=catt('objExcel.Sheets("',"&sheet.",'").Select');
    put script;
    put 'objExcel.Visible = True';
    script=catt('objExcel.Sheets("',"&sheet.",'").Range("',
     "&range.",'").Activate');
    put script;
    %if %upcase(&usenotepad) eq Y %then %do;
      %let Return_to = Return_4;
      %goto use_npad;
      %Return_4:
    %end;
    script=catt('objExcel.Sheets("',"&sheet.",'").Paste');
    put script;
    put 'objExcel.DisplayAlerts = False';
    script=catt('Newbook.SaveAs("',"&outfile.",'")');
    put script;
  %end;
```

The next section of the macro's code creates the VB script needed to modify an existing worksheet.

```
  %else %do;
    script=catt('Set Newbook = objExcel.Workbooks.Open("',
     "&outfile.",'")');
    put script;
    script=catt('Newbook.Sheets("',"&sheet.",'").Select');
    put script;
    script=catt('Newbook.Sheets("',"&sheet.",
     '").Range("',"&range.",'").Activate');
    put script;

    %if %upcase(&usenotepad) eq Y %then %do;
      %let Return_to = Return_5;
      %goto use_npad;
      %Return_5:
    %end;
    script=catt('Newbook.Sheets("',"&sheet.",'").Paste');
    put script;
    put 'objExcel.DisplayAlerts = False';
    script=catt('Newbook.SaveAs("',"&outfile.",'")');
    put script;
  %end;
%end;
put 'objExcel.Workbooks.Close';
put 'objExcel.DisplayAlerts = True';
put 'objExcel.Quit';
```

The next section of the macro's code, as shown on the following page, contains the code that is needed to create pivot tables:

```
%if %length(&pivot.) gt 2 %then %do;
  put 'Set XL = CreateObject("Excel.Application")';
  put 'XL.Visible=True';
  script=catt('XL.Workbooks.Open "',"&outfile.",'"');
  put script;
  put 'Xllastcell= xl.cells.specialcells(11).address';
  put'XL.Sheets.Add.name = "PivotTable"';
  script=catt('xldata="',"&sheet.",'"');
  put script;
  put 'XL.Sheets(xldata).select';
  put 'XL.ActiveSheet.PivotTableWizard SourceType=xlDatabase,
    XL.Range("A1" & ":" & xllastcell),"Pivottable!R1C1",xldata';
  %do i=1 %to %sysfunc(countw(&pivot.));
    %if &i lt %sysfunc(countw(&pivot.)) %then %do;
      script=catt('XL.ActiveSheet.PivotTables(xldata).PivotFields("',
       "%scan(&pivot.,&i.)",'").Orientation = 1');
    %end;
    %else %do;
      script=catt('XL.ActiveSheet.PivotTables(xldata).PivotFields("',
       "%scan(&pivot.,&i.)",'").Orientation = 4');
    %end;
    put script;
  %end;
  put 'XL.ActiveWorkbook.ShowPivotTableFieldList = False';
  put 'XL.DisplayAlerts = False';
  script=catt('XL.ActiveWorkbook.SaveAs("',"&outfile.",'")');
  put script;
  put 'XL.Workbooks.Close';
  put 'XL.DisplayAlerts = True';
  put 'XL.Quit';
%end;
%goto lastline;
```

The next section of the macro's code contains the code that is linked to with %goto statements:

```
%use_npad:
  put 'Dim objShell';
  put 'Set objShell = CreateObject("WScript.Shell")';
  script=catt('objShell.Run "notepad.exe',
              " &server_path.\clip.txt",'"');
  put script;
  put 'Do Until Success = True';
  put 'Success = objShell.AppActivate("Notepad")';
  put 'Wscript.Sleep 1000';
  put 'Loop';
  put %str('objShell.SendKeys "%E"');
  put 'Do Until Success = True';
  put 'Success = objShell.AppActivate("Notepad")';
  put 'Wscript.Sleep 1000';
  put 'Loop';
  put 'objShell.SendKeys "A"';
  put 'Do Until Success = True';
  put 'Success = objShell.AppActivate("Notepad")';
  put 'Wscript.Sleep 1000';
```

```
        put 'Loop';
        put %str('objShell.SendKeys "%E"');
        put 'Do Until Success = True';
        put 'Success = objShell.AppActivate("Notepad")';
        put 'Wscript.Sleep 1000';
        put 'Loop';
        put 'objShell.SendKeys "C"';
        put 'Do Until Success = True';
        put 'Success = objShell.AppActivate("Notepad")';
        put 'Wscript.Sleep 1000';
        put 'Loop';
        put %str('objShell.SendKeys "%F"');
        put 'Do Until Success = True';
        put 'Success = objShell.AppActivate("Notepad")';
        put 'Wscript.Sleep 1000';
        put 'Loop';
        put 'objShell.SendKeys "X"';
        put 'Do Until Success = True';
        put 'Success = objShell.AppActivate("Notepad")';
        put 'Wscript.Sleep 1000';
        put 'Loop';
        put 'objShell.SendKeys "{TAB}"';
        put 'WScript.Sleep 500';
        put 'objShell.SendKeys "{ENTER}"';
        put 'Wscript.Sleep 1000';
        %goto &Return_to.;

      %lastline:
        %if %upcase(&usenotepad) eq Y %then put 'WScript.Quit';;
    run;
```

The final section of the macro's code runs the VB script, then deletes the temporary file T_E_M_P.:

```
      data _null_;
        call system(&script.);
      run;
    %end;
  %end;

  /*Delete all temporary files*/
  proc delete data=work.t_e_m_p;
  run;
%mend exportxl;
```

WHERE TO GET THE MACRO

We did our best to only include carefully written and tested code, but the code may have to be updated from time to time to correct for errors or enhancements that we or others might discover or request. Additionally, while a copy of the macro is included in this paper, copying and pasting from a pdf file often introduces stylish looking quotation marks which aren't correctly recognized by SAS. As such, we created a page for the paper on github. The page includes copies of the source code, a powerpoint presentation, and the most recently updated version of this paper. The page can be found at:
https://github.com/FriedEgg/Papers/tree/master/Excelling_to_Another_Level_with_SAS/doc

CONCLUSION

The purpose of the present paper was to create and describe alternative menu and non-menu driven methods one could use for exporting SAS datasets to Excel, as well as use SAS Explorer's built-in functionality for adding pop-up menu actions. The methods are generalizable to almost any task that has to be repeated. Thus, one could use the methods to build menu items for running PROC CONTENTS, PROC MEANS, PROC UNIVARIATE, or any proc or custom macro one might typically use to gain an understanding of any dataset, all with just one click.

REFERENCES

A Poor/Rich Users Proc Export, Tabachneck,A., Abernathy, T., and Kastin, M., SGF 2014 Proceedings, http://support.sas.com/resources/papers/proceedings14/1793-2014.pdf

Automagically Copying and Pasting Variable Names, Tabachneck, A., Herbison, R., Clapson, A., King, J., DeAngelis, R. and Abernathy, T., SGF 2010 Proceedings, http://support.sas.com/resources/papers/proceedings10/046-2010.pdf

Copy and Paste Almost Anything, Tabachneck, A., Herbison, R., King, J., DeVenezia, R., Derby, N., and Powell, B., SGF 2012 Proceedings, http://support.sas.com/resources/papers/proceedings12/238-2012.pdf

Doing More with the SAS® Display Manager: From Editor to ViewTable - Options and Tools You Should Know, Carpenter, A., SGF 2012 Proceedings, http://support.sas.com/resources/papers/proceedings12/151-2012.pdf

FILENAME, CLIPBOARD Access Method, SAS 9.2 Documentation, SAS Institute 2012, http://support.sas.com/documentation/cdl/en/lrdict/64316/HTML/default/viewer.htm#a002571877.htm

SAS® Explorer: Use and Customization, DeVenezia, R., NESUG 2005, http://www.nesug.org/proceedings/nesug05/ap/ap6.pdf

Step-by-Step Programming with Base SAS® Software, Customizing the SAS Widowing Environment http://support.sas.com/documentation/cdl/en/basess/58133/HTML/default/viewer.htm#a001115650.htm

Which SASAUTOS Macros Are Available to My SAS® Session?, Droogendyk, H, SESUG 2008, http://analytics.ncsu.edu/sesug/2008/SBC-126.pdf

CONTACT INFORMATION

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

Arthur Tabachneck, Ph.D., CEO
Analyst Finder, Inc. Thornhill, ON Canada
E-mail: art@analystfinder.com

Tom Abernathy
Pfizer, Inc.
New York, NY 1001
E-mail: tom.abernathy@pfizer.com

Matthew Kastin
NORC at the University of Chicago
Chicago, IL
E-mail: fried.egg@verizon.net

APPENDIX A
/*THE %EXPORTXL MACRO*/

```
/** The %exportxl macro
  *
  * This macro exports SAS datasets to Excel. It only requires base SAS and
  * provides the abilities to:
  *
  *   * create new workbooks, worksheets, and modify existing worksheets
  *   * include or not include a row containing variable names or labels
  *   * output to a range that isn't pre-defined
  *   * use Excel templates or workbooks as templates
  *   * output formatted or unformatted values
  *   * export a file to your system's clipboard so that it can be pasted
        into another program (e.g., Word or Powerpoint)
  *   * avoid 32/64 bit incompatibility issues
  *   * create pivot tables
  *
  * The macro was designed so that it can be called directly or included
  * as a SAS abbreviation
  *
  * AUTHORS: Arthur Tabachneck, Tom Abernathy and Matt Kastin
  * CREATED: August 6, 2013
  * MOST RECENT VERSION: November 10, 2017

  * Named Parameter Descriptions:

  * data: the parameter to which you would assign the name of the file that
    you want to export.  Like with PROC EXPORT, you can use either a one or
    two-level filename, and dataset options can be included.  If you assign
    a one-level filename, the libname work will be used.  When the macro is
    called as an action from a SAS Explorer window, this parameter should be
    set to equal: data=%8b.%32b which the macro will interpret as
    libname.filename of the file that was selected

  * outfile: the parameter to which you would assign the path and filename of
    the workbook that you want the macro to either create or modify.  The
    file's path must exist before the macro is run. Any one of the following
    values can be used:

    Any valid filename (including path) for which you have write access
      Null The output file will be created using the selected data file's
           pathname, a back slash, the selected data file's filename, and
           the xlsx extension
      W    Provide window for user input during run

  * sheet: the parameter to which you would assign the name of the worksheet
    that you want to either create or modify. Any one of the following
    values can be used:

      Any valid worksheet name
      Null The filename of the data parameter
      W    Provide window for user input during run

  * type: the parameter you would use to indicate the type of process that
    you want to run. The default value of this parameter is: N. Any one of
    the following values can be used:
```

N Create a new workbook
A Add a new worksheet to an existing workbook
M Modify an existing worksheet
C Copy the dataset to your system's clipboard

* usenames: the parameter you would use to indicate whether you want the first row of the range to contain the first data record, the variable names, or the variable labels. The default value of this parameter is: Y. Any one of the following values can be used:

N Don't include a variable name row
Y You want the top most row to contain variable names
L You want the top most row to contain variable labels if they exist, otherwise use variable names
W Provide window for user input during run

* range: the parameter you would use to indicate the upper left cell where you want the table to begin. The default value is: A1. Any one of The following values can be used:

Any valid Excel cell name
W Provide window for user input during run

* template: the parameter you would use if you have a preformatted Excel template (or Excel workbook) that you want to apply to the data you are exporting. In such a case, use this parameter to specify the template's path and filename (e.g., template=c:\temp\template.xltx). Any one of the following values can be used:

Any valid workbook or template filename (including path)
Null No template is to be used
W Provide window for user input during run

* templatesheet: If you include the template parameter you must use this parameter to specify the template's Worksheet that contains the template you want to apply (e.g., templatesheet=template). Either of the following values can be used:

Any existing worksheet name
W Provide window for user input during run

* useformats: the parameter you would use to indicate whether you want a dataset's formats to be applied when you exporting its data. The default value of this parameter is: N. Any one of the following values can be used:

Y If you want a dataset's formats to be applied
N If you don't want a dataset's formats to be applied
W Provide window for user input during run

* usenotepad: If you're running this macro on a server, or on an operating system that doesn't provide direct access to your computer's clipboard, or have an Excel configuration that clears the clipboard upon opening, set this parameter to equal 'Y'. You should only use this parameter if you meet one of the above conditions, as it is less efficient than the method used for this parameter's default value (i.e., N). Any one of the

```
    following values can be used:

      N Don't use Notepad
      Y Use your system's or server's version of Notepad

  * pivot: the parameter you would use to indicate whether you want a
    pivot table created at the same time as you're exporting data.  The
    default value of this parameter is a NULL value.

    If you want the macro to create a pivot table, this parameter should
    contain a space separated list of all of the character variables you
    want the Pivot Table to use as class variables, followed by another
    space, and the name of the variable you want the Pivot Table to use
    as its analytical variable. Any one of the following values can be used:

      Space separated list of variable names
      W Provide window for user input during run


*/

%macro exportxl(data=,
                outfile=,
                sheet=,
                type=N,
                usenames=Y,
                range=A1,
                template=,
                templatesheet=,
                useformats=N,
                usenotepad=N,
                pivot=);

/*Check whether the data parameter contains a one or two-level filename*/
/*and, if needed, separate libname and data from data set options */
  %let lp=%sysfunc(findc(%superq(data),%str(%()));
  %if &lp. %then %do;
   %let rp=%sysfunc(findc(%superq(data),%str(%)),b));
   %let dsoptions=%qsysfunc(substrn(%nrstr(%superq(data)),&lp+1,&rp-&lp-1));
   %let data=%sysfunc(substrn(%nrstr(%superq(data)),1,%eval(&lp-1)));
  %end;
  %else %let dsoptions=;
  %if %sysfunc(countw(&data.)) eq 2 %then %do;
    %let libnm=%scan(&data.,1);
    %let filenm=%scan(&data.,2);
  %end;
  %else %do;
    %let libnm=work;
    %let filenm=&data.;
  %end;

  %if %upcase(&outfile.) eq W %then %do;
    data _null_;
      window outfile rows=8 columns=80
      irow=1 icolumn=2 color=black
      #2 @3 'Enter path and filename of desired outfile: '
      color=gray outfile $70. required=yes
```

```
      attr=underline color=yellow;
      DISPLAY outfile blank;
      call symputx('outfile',outfile);
      stop;
    run;
  %end;
  %if %length(&outfile.) lt 1 %then
    %let outfile=%sysfunc(pathname(&libnm.))\&filenm..xlsx;;

  %if %upcase(&sheet.) eq W %then %do;
    data _null_;
      window sheet rows=8 columns=80
      irow=1 icolumn=2 color=black
      #2 @3 'Enter valid worksheet name: '
      color=gray sheet $40. required=yes
      attr=underline color=yellow;
      DISPLAY sheet blank;
      call symputx('sheet',sheet);
      stop;
    run;
  %end;
  %if %length(&sheet.) lt 1 %then
    %let sheet=&filenm.;;

  /*Left for compatibility with previous version*/
  %if %upcase(&type.) eq P %then %do;
    proc export
      data=&libnm..&filenm.
          %if %length(%unquote(&dsoptions.)) gt 2 %then
(%unquote(&dsoptions.));
      outfile= "&outfile."
      dbms=xlsx
      replace
      ;
      %if &sheet. ne "" %then sheet="&sheet.";;
    run;
  %end;
  /*end of compatibility code - Note: above is not documented in paper*/
  %else %do;
    %if %upcase(&range.) eq W %then %do;
      data _null_;
       window range rows=8 columns=80
       irow=1 icolumn=2 color=black
       #2 @3 'Enter the upper left cell where range should begin (e.g. D5): '
       color=gray range $41. required=yes
       attr=underline color=yellow;
       DISPLAY range blank;
       call symputx('range',range);
       stop;
      run;
    %end;
    %else %if %length(&range.) lt 2 %then %do;
      %let range=A1;
    %end;

    %if %upcase(template) eq W %then %do;
      data _null_;
```

```
    window template rows=8 columns=80
    irow=1 icolumn=2 color=black
    #2 @3 'Enter the template path and name: '
    color=gray template $41. required=yes
    attr=underline color=yellow;
    DISPLAY template blank;
    call symputx('template',template);
    stop;
  run;
%end;
%else %if %length(&template.) lt 2 %then %do;
  %let template=;
%end;

%if %upcase(&templatesheet.) eq W %then %do;
  data _null_;
    window templatesheet rows=8 columns=80
    irow=1 icolumn=2 color=black
    #2 @3 "Enter the template sheet's name: "
    color=gray templatesheet $41. required=yes
    attr=underline color=yellow;
    DISPLAY templatesheet blank;
    call symputx('templatesheet',templatesheet);
    stop;
  run;
%end;
%else %if %length(&templatesheet.) lt 2 %then %do;
  %let templatesheet=;
%end;

%if %upcase(&pivot) eq W %then %do;
  data _null_;
    window pivot rows=8 columns=80
    irow=1 icolumn=2 color=black
    #2 @3 'Enter the space separated class and analysis variable list: '
    color=gray pivot $41. required=yes
    attr=underline color=yellow;
    DISPLAY pivot blank;
    call symputx('pivot',pivot);
    stop;
  run;
%end;
%else %if %length(&pivot.) lt 2 %then %do;
  %let pivot=;
%end;

%if %upcase(&useformats) eq W %then %do;
  data _null_;
    window useformats rows=8 columns=80
    irow=1 icolumn=2 color=black
    #2 @3 'Enter whether to use formats (Y/N): '
    color=gray useformats $1. required=yes
    attr=underline color=yellow;
    DISPLAY useformats blank;
    call symputx('useformats',useformats);
    stop;
  run;
```

```
%end;

data _null_;
  dsid=open(catx('.',"&libnm.","&filenm."));
  if dsid eq 0 then do;
    rc=close(dsid);
    link err;
  end;
  rc=close(dsid);
  err:
  do;
    m = sysmsg();
    put m;
    stop;
  end;
run;

data t_e_m_p;
  set &libnm..&filenm. (%unquote(&dsoptions.) obs=1);
run;

proc sql noprint;
  select name,length,type,format,strip(coalescec(label,name))
    into :vnames separated by "~",
         :vlengths separated by "~",
         :vtypes separated by "~",
         :vformats separated by "~",
         :vlabels separated by "~"
      from dictionary.columns
        where libname="WORK" and
              memname="T_E_M_P"
  ;
quit;
%let nvar=&sqlobs.;

filename code2inc temp;
data _null_;
  file code2inc;
  length script $80;
  length fmt $32;
  do i=1 to &nvar;
    if i gt 1 then put 'rc=fput(fid,"09"x);';
    %if %upcase(&useformats.) eq Y %then %do;
      fmt=scan("&vformats.",i,"~","M");
    %end;
    %else call missing(fmt);;
    if scan("&vtypes.",i,"~") eq 'char' then do;
      if missing(fmt) then
       fmt=catt('$',scan("&vlengths.",i,"~","M"),'.');
      script=catt('rc=fput(fid,putc(put(',
       scan("&vnames.",i,"~","M"),',',fmt,"),'$char",
       scan("&vlengths.",i,"~","M"),'.'));");
      put script;
    end;
    else do;
      if missing(fmt) then fmt='best32.';
      script=catt('rc=fput(fid,putc(put(',
```

```
       scan("&vnames.",i,"~","M"),',',fmt,"),'$char32.'));");
      put script;
    end;
  end;
  put 'rc=fwrite(fid);';
run;

data _null_;
  dsid=open("work.t_e_m_p");
  rc=attrn(dsid,'any');
  if rc ne 1 then do;
    rc=close(dsid);
    link err;
  end;
  rc=close(dsid);
  err:
  do;
    m = sysmsg();
    put m;
    stop;
  end;
run;

data _null_;
  %if %upcase(&usenotepad) eq Y %then %do;
    %let server_path=%sysfunc(pathname(work));
    rc=filename('clippy',"&server_path.\clip.txt",'DISK');
  %end;
  %else %do;
    rc=filename('clippy',' ','clipbrd');
  %end;
  if fid eq 0 then link err;
  do i = 1 to &nvar.;
    %if %upcase(&usenames.) ne N %then %do;
      if i gt 1 then rc=fput(fid,'09'x);
      %if %upcase(&usenames.) eq Y %then %do;
        rc=fput(fid,scan("&vnames.",i,"~","M"));
      %end;
      %else %do;
        if missing(scan("&vlabels.",i,"~","M")) then
         rc=fput(fid,scan("&vnames.",i,"~"));
        else rc=fput(fid,scan("&vlabels.",i,"~","M"));
      %end;
    %end;
  end;
  %if %upcase(&usenames.) ne N %then %do;
    rc=fwrite(fid);;
  %end;
  do until (lastone);
    set &libnm..&filenm.
      %if %length(%unquote(&dsoptions.))>2 %then (%unquote(&dsoptions.));
      end=lastone;
    %include code2inc;
  end;
  rc=fclose(fid);
  rc=filename('clippy');
  rc=filename('code2inc');
```

```
  stop;

  err:
  do;
    m = sysmsg();
    put m;
    rc=filename('code2inc');
    stop;
  end;
run;

%if %upcase(&type.) eq N or %upcase(&type.) eq M
 or %upcase(&type.) eq A %then %do;

  data _null_;
    length script filevar $256;
    script = catx('\',pathname('WORK'),'PasteIt.vbs');
    filevar = script;
    script="'"||'cscript "'||trim(script)||'"'||"'";
    call symput('script',script);
    file dummy1 filevar=filevar recfm=v lrecl=512;

    put 'Dim objExcel';
    put 'Dim Newbook';

    %if %length(&template.) gt 1 %then %do;
      %if %upcase(&type.) eq A %then put 'Dim OldBook';;
      put 'Set objExcel = CreateObject("Excel.Application")';
      put 'objExcel.Visible = True';
      %if %upcase(&type.) eq N %then %do;
        script=catt('Set Newbook = objExcel.Workbooks.Add("',
         "&template.",'")');
        put script;
        script=catt('objExcel.Sheets("',"&TemplateSheet.",
         '").Select');
        put script;
        script=catt('objExcel.Sheets("',"&TemplateSheet.",
         '").Name = "',"&sheet.",'"');
        put script;
        put 'objExcel.Visible = True';
        script=catt('objExcel.Sheets("',"&sheet.",
         '").Range("',"&range.",'").Activate');
        put script;
        %if %upcase(&usenotepad) eq Y %then %do;
          %let Return_to = Return_1;
          %goto use_npad;
          %Return_1:
        %end;
        script=catt('objExcel.Sheets("',"&sheet.",'").Paste');
        put script;
        script=catt('objExcel.Sheets("',"&sheet.",
         '").Range("A1").Select');
        put script;
        put 'objExcel.DisplayAlerts = False';
        script=catt('NewBook.SaveAs("',"&outfile.",'")');
        put script;
      %end;
```

```
  %else %do;
    script=catt('strFile="',"&outfile.",'"');
    put script;
    script=catt('Set OldBook=objExcel.Workbooks.Open("',
     "&outfile.",'")');
    put script;
    script=catt('Set Newbook = objExcel.Workbooks.Add("',
     "&template.",'")');
    put script;
    script=catt('objExcel.Sheets("',"&TemplateSheet.",
     '").Select');
    put script;
    script=catt('objExcel.Sheets("',"&TemplateSheet",
     '").Name ="',"&sheet.",'"');
    put script;
    put 'objExcel.Visible = True';
    script=catt('objExcel.Sheets("',"&sheet.",
     '").Range("',"&range.",'").Activate');
    put script;
    %if %upcase(&usenotepad) eq Y %then %do;
      %let Return_to = Return_2;
      %goto use_npad;
      %Return_2:
    %end;
    script=catt('objExcel.Sheets("',"&sheet.",'").Paste');
    put script;
    script=catt('objExcel.Sheets("',"&sheet.",
     '").Range("A1").Select');
    put script;
    script=catt('objExcel.Sheets("',"&sheet.",
     '").Move ,OldBook.Sheets( OldBook.Sheets.Count )');
    put script;
    put 'objExcel.DisplayAlerts = False';
    script=catt('OldBook.SaveAs("',"&outfile.",'")');
    put script;
  %end;
%end;
%else %do;
  %if %upcase(&type.) eq N or %upcase(&type.) eq A %then %do;
    %if %upcase(&type.) eq N %then put 'Dim NewSheet';;
    put 'Dim inSheetCount';
    %if %upcase(&type.) eq A %then put 'Dim strFile';;
  %end;

  put 'Set objExcel = CreateObject("Excel.Application")';

  %if %upcase(&type.) eq N %then %do;
    put 'Set Newbook = objExcel.Workbooks.Add()';
    put 'objExcel.Visible = True';
    put 'inSheetCount = Newbook.Application.Worksheets.Count';
    script=catt('set NewSheet = Newbook.Sheets.Add',
     '( ,objExcel.WorkSheets(inSheetCount))');
    put script;
    put 'objExcel.DisplayAlerts = False';
    put 'i = inSheetCount';
    put 'Do Until i = 0';
    put ' Newbook.Worksheets(i).Delete';
```

```
  put ' i = i - 1';
  put ' Loop';
  script=catt('Newbook.Sheets(1).Name="',
   "&sheet.",'"');
  put script;
  script=catt('Newbook.Sheets("',"&sheet.",'").Select');
  put script;
  script=catt('Newbook.Sheets("',"&sheet.",
   '").Range("',"&range.",'").Activate');
  put script;
  %if %upcase(&usenotepad) eq Y %then %do;
    %let Return_to = Return_3;
    %goto use_npad;
    %Return_3:
  %end;
  script=catt('Newbook.Sheets("',"&sheet.",'").Paste');
  put script;
  script=catt('NewSheet.SaveAs("',"&outfile.",'")');
  put script;
%end;
%else %if %upcase(&type.) eq A %then %do;
  script=catt('strFile="',"&outfile.",'"');
  put script;
  put 'objExcel.Visible = True';
  put 'objExcel.Workbooks.Open strFile';
  put 'inSheetCount = objExcel.Application.Worksheets.Count';
  script=catt('set NewBook = objExcel.Sheets.Add( ,objExcel.',
   'WorkSheets(inSheetCount))');
  put script;
  script=catt('objExcel.Sheets(inSheetCount + 1).Name="',
   "&sheet.",'"');
  put script;
  script=catt('objExcel.Sheets("',"&sheet.",
   '").Select');
  put script;
  put 'objExcel.Visible = True';
  script=catt('objExcel.Sheets("',"&sheet.",'").Range("',
   "&range.",'").Activate');
  put script;
  %if %upcase(&usenotepad) eq Y %then %do;
    %let Return_to = Return_4;
    %goto use_npad;
    %Return_4:
  %end;
  script=catt('objExcel.Sheets("',"&sheet.",'").Paste');
  put script;
  put 'objExcel.DisplayAlerts = False';
  script=catt('Newbook.SaveAs("',"&outfile.",'")');
  put script;
%end;
%else %do;
  script=catt('Set Newbook = objExcel.Workbooks.Open("',
   "&outfile.",'")');
  put script;
  script=catt('Newbook.Sheets("',"&sheet.",'").Select');
  put script;
  script=catt('Newbook.Sheets("',"&sheet.",
```

```
     '").Range("',"&range.",'").Activate');
    put script;

    %if %upcase(&usenotepad) eq Y %then %do;
      %let Return_to = Return_5;
      %goto use_npad;
      %Return_5:
    %end;
    script=catt('Newbook.Sheets("',"&sheet.",'").Paste');
    put script;
    put 'objExcel.DisplayAlerts = False';
    script=catt('Newbook.SaveAs("',"&outfile.",'")');
    put script;
  %end;
%end;
put 'objExcel.Workbooks.Close';
put 'objExcel.DisplayAlerts = True';
put 'objExcel.Quit';

%if %length(&pivot.) gt 2 %then %do;
  put 'Set XL = CreateObject("Excel.Application")';
  put 'XL.Visible=True';
  script=catt('XL.Workbooks.Open "',"&outfile.",'"');
  put script;
  put 'Xllastcell= xl.cells.specialcells(11).address';
  put'XL.Sheets.Add.name = "PivotTable"';
  script=catt('xldata="',"&sheet.",'"');
  put script;
  put 'XL.Sheets(xldata).select';
  put 'XL.ActiveSheet.PivotTableWizard SourceType=xlDatabase,
    XL.Range("A1" & ":" & xllastcell),"Pivottable!R1C1",xldata';
  %do i=1 %to %sysfunc(countw(&pivot.));
    %if &i lt %sysfunc(countw(&pivot.)) %then %do;
      script=catt('XL.ActiveSheet.PivotTables(xldata).PivotFields("',
       "%scan(&pivot.,&i.)",'").Orientation = 1');
    %end;
    %else %do;
      script=catt('XL.ActiveSheet.PivotTables(xldata).PivotFields("',
       "%scan(&pivot.,&i.)",'").Orientation = 4');
    %end;
    put script;
  %end;
  put 'XL.ActiveWorkbook.ShowPivotTableFieldList = False';
  put 'XL.DisplayAlerts = False';
  script=catt('XL.ActiveWorkbook.SaveAs("',"&outfile.",'")');
  put script;
  put 'XL.Workbooks.Close';
  put 'XL.DisplayAlerts = True';
  put 'XL.Quit';
%end;
%goto lastline;

%use_npad:
  put 'Dim objShell';
  put 'Set objShell = CreateObject("WScript.Shell")';
  script=catt('objShell.Run "notepad.exe',
              " &server_path.\clip.txt",'"');
```

```
          put script;
          put 'Do Until Success = True';
          put 'Success = objShell.AppActivate("Notepad")';
          put 'Wscript.Sleep 1000';
          put 'Loop';
          put %str('objShell.SendKeys "%E"');
          put 'Do Until Success = True';
          put 'Success = objShell.AppActivate("Notepad")';
          put 'Wscript.Sleep 1000';
          put 'Loop';
          put 'objShell.SendKeys "A"';
          put 'Do Until Success = True';
          put 'Success = objShell.AppActivate("Notepad")';
          put 'Wscript.Sleep 1000';
          put 'Loop';
          put %str('objShell.SendKeys "%E"');
          put 'Do Until Success = True';
          put 'Success = objShell.AppActivate("Notepad")';
          put 'Wscript.Sleep 1000';
          put 'Loop';
          put 'objShell.SendKeys "C"';
          put 'Do Until Success = True';
          put 'Success = objShell.AppActivate("Notepad")';
          put 'Wscript.Sleep 1000';
          put 'Loop';
          put %str('objShell.SendKeys "%F"');
          put 'Do Until Success = True';
          put 'Success = objShell.AppActivate("Notepad")';
          put 'Wscript.Sleep 1000';
          put 'Loop';
          put 'objShell.SendKeys "X"';
          put 'Do Until Success = True';
          put 'Success = objShell.AppActivate("Notepad")';
          put 'Wscript.Sleep 1000';
          put 'Loop';
          put 'objShell.SendKeys "{TAB}"';
          put 'WScript.Sleep 500';
          put 'objShell.SendKeys "{ENTER}"';
          put 'Wscript.Sleep 1000';
          %goto &Return_to.;

        %lastline:
          %if %upcase(&usenotepad) eq Y %then put 'WScript.Quit';;
      run;

      data _null_;
        call system(&script.);
      run;
    %end;
  %end;

  /*Delete all temporary files*/
  proc delete data=work.t_e_m_p;
  run;
%mend exportxl;

/*Useage Examples. The following examples assume that you have
```

```
write access to a directory named: c:\temp. Having that specific
directory isn't a requirement for the macro, but you do need to have
write access to the file that you specify in the outfile parameter

* Example 1: Create a new workbook (c:\temp\class.xlsx), copying all
  records from sashelp.class, letting the macro automatically name the
  worksheet (i.e., use the data parameter's filename: class), with the
  worksheet's first row containing the dataset's variable names:

  %exportxl(data=sashelp.class, outfile=c:\temp\class.xlsx)

* Example 2: Create the same workbook as in Example 1, but name the
  worksheet 'Students', and don't include a variable name header record:

    %exportxl(data=sashelp.class, outfile=c:\temp\class.xlsx,usenames=N,
      sheet=Students)

* Example 3: Same as Example 2, but running on a system that doesn't
  provide direct access to your computer's clipboard (e.g., a server),
  or have an Excel configuration that clears the clipboard upon opening:

    %exportxl(data=sashelp.class, outfile=c:\temp\class.xlsx,usenames=N,
      sheet=Students, usenotepad=Y)

* Example 4: Create a new workbook from sashelp.cars, name the worksheet
  'cars', and have the worksheet's first row contain the dataset's
  variable labels:

    %exportxl( data=sashelp.cars, outfile=c:\temp\cars.xlsx,usenames=L)

* Example 5: Create a new workbook (c:\temp\class.xlsx), copying all
  records for males from sashelp.class, name the worksheet 'Males', and
  have the worksheet's first row contain the dataset's variable names:

    %exportxl( data=sashelp.class(where=(sex eq 'M')),sheet=Males,
      outfile=c:\temp\class.xlsx)


* Example 6: Modify the workbook created in Example 5, adding a new
  worksheet named 'Females', copying all records for females from
  sashelp.class, and have the worksheet's first row contain the
  dataset's variable names:

    %exportxl( data=sashelp.class(where=(sex eq 'F')),sheet=Females,
      outfile=c:\temp\class.xlsx,type=A)

* Example 7: Create a workbook using an Excel template

    %exportxl( data=sashelp.class (keep=name sex age height),
      template=c:\temp\template.xltx, templatesheet=template,
      outfile=c:\temp\class_stats.xlsx, usenames=N,
      range=A2, sheet=Jan_2018)

* Example 8: Modify workbook created by running Example 7, adding
  the weight variable to column E

    %exportxl( data=sashelp.class (keep=weight), type=M, range=E2,
```

```
        outfile=c:\temp\class_stats.xlsx, usenames=N, sheet=Jan_2018)

  * Example 9: Create a new workbook including a Pivot Table

      %exportxl( data=sashelp.cars, outfile=c:\temp\cars.xlsx,
        pivot=Origin Type Make MSRP)

*/
```

APPENDIX B

/*THE %EXPORTXL MACRO TIP SHEET*/

Exportxl Macro Tip Sheet

Purpose: The macro exports SAS datasets to Excel workbooks and/or worksheets. It can create new workbooks or worksheets, modify existing worksheets, export to a range that hasn't been predefined, use variable names or labels, use or not use variable formats, use Excel templates or existing worksheets as templates, and create pivot tables.

Named Parameters: The macro uses Named parameters so that: (1) default values can be assigned and (2) the various parameters only have to be specified when values other than the default values are needed.

Parameter	Required	Possible Values	Default	Description
data	Yes	Any valid one or two-level SAS filename	Null	The 1 or2-level filename you want to export
outfile	Yes	Any valid filename (including path) W = provide window for user input Null = path and filename of the data parameter + xlsx extension	Null	The path and filename of the workbook that you want the macro to create or modify
sheet	No	Valid worksheet name W = provide window for user input Null = filename of data parameter	Null	The name of the worksheet you want to create or modify
type	No	P= run PROC EXPORT N = create a new workbook using VBS A = add new worksheet using VBS M = modify worksheet using VBS C = copy dataset to system clipboard	N	The type of process that you want to run
usenames	No	N = don't include a variable name row Y = include a variable name row L = include a variable label row W = provide window for user input	Y	Whether the first row of the range will contain the first data record, the variable names, or the variable labels
range	No	Any valid Excel cell name W = provide window for user input	A1	The upper left cell where you want the table to begin
template	No	Null = No template is to be used filename (including path) of an existing Excel template or workbook W = provide window for user input	Null	The filename (including path) of an Excel template or workbook that you want applied as a template to the file you are exporting
templatesheet	Yes if template specified	Null = No template is to be used Worksheet name W = provide window for user input	Null	The name of the worksheet to be used as a template
useformats	No	Y = Yes N = No W = provide window for user input	N	Whether dataset's formats should be applied when exporting its data
usenotepad	No	N = Don't use Notepad Y = Use Notepad	N	Notepad is needed If you're running this macro on a system that doesn't provide direct access to your computer's clipboard (e.g., if you're running this macro on a server)
pivot	No	Null Space separated list of variable names W = provide window for user input	Null	Space separated list of the character variable names to use as Pivot Table's class variables, followed a space, and the name of the analytical variable

Usage Examples: The following examples assume that you have write access to a directory named: c:\temp. Having that specific directory isn't a requirement for the macro, but you do need to have write access to the file that you specify in the outfile parameter

Example 1: Create a new workbook (c:\temp\class.xlsx), copying all records from sashelp.class, letting the macro automatically name the worksheet (i.e., use the data parameter's filename: class), with the worksheet's first row containing the dataset's variable names

```
%exportxl(data=sashelp.class, outfile=c:\temp\class.xlsx)
```

Example 2: Create the same workbook as in Example 1, but name the worksheet 'Students', and don't include a variable name header record

```
%exportxl(data=sashelp.class,  outfile=c:\temp\class.xlsx,usenames=N,  sheet=Students)
```

Example 3: Same as Example 2, but running on a system that doesn't provide direct access to your computer's clipboard (e.g., a server), or have an Excel configuration that clears the clipboard upon opening

```
%exportxl(data=sashelp.class, outfile=c:\temp\class.xlsx, usenames=N, sheet=Students, usenotepad=Y)
```

Example 4: Create a new workbook from sashelp.cars, name the worksheet 'cars', and have the worksheet's first row contain the dataset's variable labels

```
%exportxl( data=sashelp.cars, outfile=c:\temp\cars.xlsx, usenames=L)
```

Example 5: Create a new workbook (c:\temp\class.xlsx), copying all records for males from sashelp.class, name the worksheet 'Males', and have the worksheet's first row contain the dataset's variable names

```
%exportxl( data=sashelp.class(where=(sex eq 'M')), sheet=Males, outfile=c:\temp\class.xlsx)
```

Example 6: Modify the workbook created in Example 5, adding a new worksheet named 'Females', copying all records for females from sashelp.class, and have the worksheet's first row contain the dataset's variable names

```
%exportxl( data=sashelp.class(where=(sex eq 'F')), sheet=Females, outfile=c:\temp\class.xlsx, type=A)
```

Example 7: Create a workbook using an Excel template

```
%exportxl( data=sashelp.class (keep=name sex age height), template=c:\temp\template.xltx,
    templatesheet=template, outfile=c:\temp\class_stats.xlsx, usenames=N, range=A2, sheet=Jan_2018)
```

Example 8: Modify workbook created by running Example 7, adding the weight variable to column E

```
%exportxl( data=sashelp.class (keep=weight), type=M, range=E2, outfile=c:\temp\class_stats.xlsx,
   usenames=N, sheet=Jan_2018)
```

Example 9: Create a new workbook including a Pivot Table

```
%exportxl( data=sashelp.cars, outfile=c:\temp\cars.xlsx, pivot=Origin Type Make MSRP)
```