Paper 1793-2014

A Poor/Rich SAS® User's Proc Export

Arthur S. Tabachneck, Ph.D., myQNA, Inc., Thornhill, Ontario Canada
Tom Abernathy, Pfizer, Inc., New York, NY
Matthew Kastin, I-Behavior, Inc., Penn Valley, PA

ABSTRACT

Have you ever wished that with one click you could copy any SAS® dataset, including variable names, so that you could paste the text into a Word file, powerpoint or spreadsheet? 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 introductions of both 64-bit computers, and XLXS versions of Excel Workbooks, the task of exporting SAS® datasets to Excel has become increasingly difficult and often requires the purchase of additional products (like SAS/Access for PC File Formats) and installing the PC Files Server system. Interestingly, a little known SAS Explorer feature provides a free alternative. When one right clicks on a dataset name in SAS Explorer, one of the menu options is *Copy Contents to Clipboard*. Clicking on that option writes an HTML version of the dataset to one's clipboard, thus allowing the data to be pasted into Excel or any other program that allows pasting from one's clipboard.

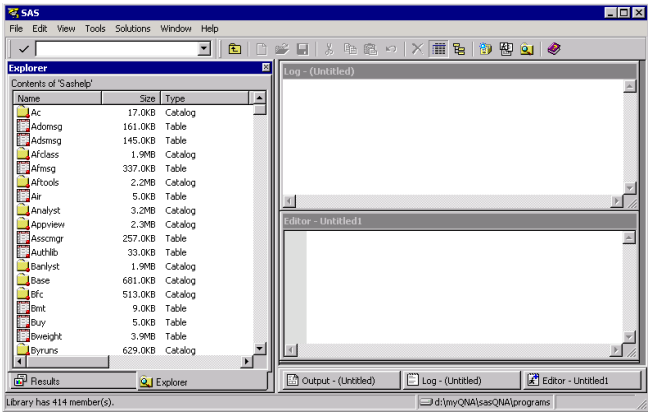
The present paper goes a couple of steps further. In addition to the actions you see when right clicking on a dataset, SAS Explorer provides a way for users to add their own actions to the menu. This paper provides step-by-step instructions on how to add seven different export actions designed to your 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 this paper which, originally, was 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****.*

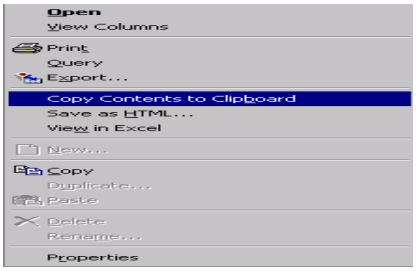
Any attempt to explain features of the SAS Windowing environment is complicated by the fact that users can alter the placement and presence of the various windows that will appear on their monitors when they open an interactive SAS session. A screenshot of one of the author's typical SAS Windowing environment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

Figure 1
One Way a SAS Windowing Environment May Appear



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:

Figure 2
An Explorer Pop-up Menu



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.

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 either automagically run Proc Export and create Excel workbooks or, for those who don't license SAS Access for PC File Formats or who simply want more capabilities than Proc Export currently provides, run a combination of data step code and vbs script that will:

(1) produce Excel workbooks with variable names in the first row;
(2) produce Excel workbooks with variable names in the first row and let you indicate the upper left cell where the tables should begin;
(3) produce Excel workbooks without including variable names in the first row;
(4) produce Excel workbooks, without variable names in the first row, and let you indicate the upper left cell where the table should begin;
(5) copy SAS datasets, with variable names, to your system's clipboard
(6) copy SAS datasets, without variable names, to your system's clipboard

The current paper's authors were already familiar with adding SAS Explorer menu options as two of us had collaborated on writing a paper, in 2010, which used the method to give users a way to copy and paste the variable names from any SAS dataset. However, other than our own paper, and some excellent papers by Richard DeVenezia and Art Carpenter, only one paragraph in the SAS documentation even mentions the capability. However, creating a pop-up action is fairly straight forward:

1. From the Explorer window, select Tools ➧ Options ➧ Explorer
2. Left click on Members
3. Double left click on Table
4. Left click on Add.
5. 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
6. Enter the command you want run whenever the action is selected
7. Select OK to exit the Add Action screen.
8. 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 located can be found 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 seven actions described in the present paper differ only on how the exportxl macro is called. In the macro declaration, five relational parameters are specified, namely libnm (libname), filenm (filename), type (which can either be P for Proc Export, S for running a VBS script) or C if one only want to copy a file to their system's clipboard), usenames (which can either be YES or NO to indicate whether variable names should be written to the output's first row), and range (which can either be YES or NO, where YES would indicate that you want to be able to specify the upper left cell where you want the table to begin).

```
%macro exportxl(libnm,filenm,type,usenames,range);
```

The following line of code isn't part of the macro but, rather, the action command one would use to cause the macro to run Proc Export to create an Excel workbook in the same directory where the SAS dataset is located:. When one left clicks on a SAS dataset in SAS Explorer, and selects an action, the dataset's libname is captured as %8b and the filename is captured as %32b.

```
gsubmit '%%exportxl(%8b,%32b,P,YES,NO)';
```

As such, the macro's first line of code uses the pathname function to create a macro variable, called *filepath*, which will contain the selected dataset's file path.

```
  %let filepath=%sysfunc(pathname(&libnm.));
```

The next section of the macro's code is all that is needed to run Proc Export. The macro will run Proc Export if the *type* parameter is set to equal P. The macro was designed to create an Excel workbook in the filepath of the selected dataset's libname, assigning the same file name as the SAS dataset. You may have to modify the macro if your system requires, or you prefer, a dbms other than xlsx (e.g., Excel, ExcelCS, or any of the other Excel engines).

The macro's other two parameters, namely *usenames* and *ranges*, are irrelevant to Proc Export as they refer to capabilities that the procedure currently doesn't offer. In fact, the following eight lines are the macro's only lines of code that are needed to accomplish this action:

```
%if &type. eq P %then %do;
  proc export
    data=&libnm..&filenm.
    outfile= "&filepath.\&filenm..xlsx"
    dbms=xlsx
    replace;
  run;
%end;
```

The next section of the macro uses Proc FCMP to create a function which we called c2cb. The purpose of the function is to write the selected dataset to your system's 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 function.

The first lines of code simply run Proc FCMP, create the function, pass in the necessary parameters (i.e., libname, member name and whether the user wants to use the variable names), and then opens the system clipboard for write access:

```
%else %do;
  proc fcmp outlib=work.func.util;
    function c2cb(lib $,mem $, usenm $);
      rc=filename('clippy',' ','clipbrd');
      if rc ne 0 then return(1);
      fid=fopen('clippy','o',0,'v');
      if fid eq 0 then do;
        rc = filename( 'clippy' );
        return(2);
      end;
```

The next lines of code open the dataset and, if the dataset can't be found, cleans up after itself and then exits the function:

```
      dsid=open(catx('.',lib,mem));
      if dsid eq 0 then do;
        rc=fclose(fid);
        rc=filename('clippy');
        return(3);
      end;
```

The next lines of code check whether the dataset has both observations and variables. If the dataset doesn't contain both, all of the files are closed and the function is exited. Conversely, if both observations and variables do exist in the dataset, the number of variables are identified and an array is initialized:

```
      rc=attrn(dsid,'any');
      if rc ne 1 then do;
        rc=fclose(fid);
        rc=close(dsid);
        rc=filename('clippy');
        return(4);
      end;
      nvar=attrn(dsid,'nvar');
      array v[1] /nosymbols;
      call dynamic_array(v,nvar);
```

The next lines of code go through all of the variables and fill the array with one of two values: 1 to indicate that the variable is a character variable or a 2 to indicate that it is a numeric variable. If the user had indicated that a variable name header row should be included, the tab-delimited variable names are then written to the clipboard:

```
      do i = 1 to nvar;
        v[i]=ifn( vartype(dsid,i)='C',1,2);
        if usenm eq 'YES' then do;
          if i gt 1 then rc=fput(fid,'09'x);
          rc=fput(fid,varname(dsid,i));
        end;
      end;

      if usenm eq 'YES' then rc=fwrite(fid);
```

The next lines of code go through all of the observations and writes all of the tab-delimited data to the clipboard. If a format hadn't been declared for any specific variable, the code creates one :

```
    do i=1 to attrn(dsid,'nlobs');
      rc=fetchobs(dsid,i);
      do j=1 to nvar;
        if j gt 1 then rc=fput(fid,'09'x);
        if (v[j] eq 1) then rc=fput(fid, getvarc(dsid,j));
        else do;
          fmt=varfmt(dsid,j) ;
          if missing(fmt) then fmt='best12.';
          rc=fput(fid,putc(putn(getvarn(dsid,j),fmt ),'$char12.'));
        end;
      end;
      rc=fwrite(fid);
    end;
```

The last section of the function closes all of the files that had been opened, ends the function, compiles it and, finally runs the function:

```
    rc=fclose(fid);
    rc=close(dsid);
    rc=filename('clippy');
    return(0);
  endsub;
quit;
%local cmplib;
%let cmplib=%sysfunc(getoption(cmplib));
options cmplib=(work.func);
%put %sysfunc(c2cb(&libnm,&filenm,&usenames)) ;
options cmplib=(&cmplib);
```

The next section of the macro was designed to accommodate those cases where a user wants the dataset to be written to a particular range within a workbook rather than simply pasting the clipboard starting at the upper left corner of the workbook. If the value of the *range* parameter has a value of YES, the window statement is called within a data step to provide the user with a way to tell the upper left corner cell of the workbook where they want the program to begin writing the data. If the range parameter doesn't have a value of YES, cell A1 is assigned:

```
%if &range. eq YES %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 dsn $41. required=yes
   attr=underline color=yellow;
   DISPLAY range blank;
   call symput('range',range);
   stop;
  run;
%end;
%else %do;
  %let range=A1;
%end;
```

The code, above, will cause the macro to display a window as shown in Figure 3 on the following page. The window will appear on the user's monitor and wait until a cell has been entered and the Enter key has been pressed. Regardless of the value of the range parameter, the parameter will be reset to equal either A1 or the user specified starting cell.

Figure 3
User Input Window Created by Window Statement

All of the above code, starting with the %else %do statement, accomplishes the task of writing a selected dataset to one's clipboard. The next section of the macro was designed to accommodate those cases where a user sets the parameter *type* to a value of S, indicating that they want the macro to use VB script to create an Excel workbook, paste the clipboard into the workbook, as well as save and close the workbook. The macro accomplishes the task by creating and running a file called PasteIt.vbs, in the user's work directory. Of course, this section of the macro can only run on an operating system, like Windows, that can run both VB script and Excel:

```
      %if &type. eq S %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';

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

          put 'objExcel.Visible = True';
          script='Newbook.Sheets("Sheet1").Range("'||
           "&Range."||'").Activate';
          put script;
          put 'Newbook.Sheets("Sheet1").Paste';

          put 'Newbook.Sheets("Sheet1").Select';
          put 'Newbook.Sheets("Sheet1").Name = "'@;
          put "&filenm."@;
          put '"';

          put 'Newbook.SaveAs("'@;
          put "&filepath.\&filenm..xlsx"@;
          put '")';

          put "Newbook.Close";

          put 'objExcel = Nothing';
          put 'Newbook = Nothing';
        run;

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

THE SEVEN ACTION COMMANDS

The ExportXL macro can be called seven different ways. The seven ways are shown, below, including our action labels, the actual action commands for each, and a description of what each command does:

Action 1: &Export to Excel
Action Command: gsubmit '%%exportxl(%8b,%32b,P,YES,NO)';
Description: Runs Proc Export to create an Excel Workbook, with the same name as the dataset, in the same directory where the selected data resides

Action 2: Export to Excel with Variable &Names
Action Command: gsubmit '%%exportxl(%8b,%32b,S,YES,NO)';
Description: Creates and runs the c2cb function and PasteIt.vbs script to create an Excel Workbook, with the same name as the dataset, in the same directory where the selected data resides. The table's first row will contain the variable names

Action 3: Export to &Range with Variable &Names
Action Command: gsubmit '%%exportxl(%8b,%32b,S,YES,YES)';
Description: Creates and runs the c2cb function and PasteIt.vbs script to create an Excel Workbook, with the same name as the dataset, in the same directory where the selected data resides. However, rather than beginning the table in cell A1, the user is asked to specify the upper left cell where the table should begin. The table's first row will contain the variable names

Action 4: Export to Excel w/&o Variable &Names
Action Command: gsubmit '%%exportxl(%8b,%32b,S,NO,NO)';
Description: Creates and runs the c2cb function and PasteIt.vbs script to create an Excel Workbook, with the same name as the dataset, in the same directory where the selected data resides. The table's first row will contain the first observation

Action 5: Export to &Range w/o Variable &Names
Action Command: gsubmit '%%exportxl(%8b,%32b,S,NO,YES)';
Description: Creates and runs the c2cb function and PasteIt.vbs script to create an Excel Workbook, with the same name as the dataset, in the same directory where the selected data resides. However, rather than beginning the table in cell A1, the user is asked to specify the upper left cell where the table should begin. The table's first row will contain the first observation.

Action 6: E&xport to Clipboard with Variable &Names
Action Command: gsubmit '%%exportxl(%8b,%32b,C,YES,NO)';
Description: Creates and runs the c2cb function and PasteIt.vbs script to copy the selected dataset to the system's clipboard. The table's first row will contain the variable names

Action 7: E&xport to Clipboard w/o Variable &Names
Action Command: gsubmit '%%exportxl(%8b,%32b,C,NO,NO)';
Description: Creates and runs the c2cb function and PasteIt.vbs script to copy the selected dataset to the system's clipboard. The table's first row will contain the first observation

FUTURE DEVELOPMENT

We stopped adding functionality to the macro in order to reserve sufficient time for testing and documentation, but there is definitely additional functionality that we would have liked to include. For example, one may not want to create the workbook in the same directory as the dataset existed. In fact, a user may not want to create a new workbook at all but, rather, append the dataset to an existing workbook. Additionally, one may want to create something other than an Excel workbook, i.e., be able to specify the DBMS that will be used.

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. 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 sasCommunity.org. The page includes copies of the source code and updated versions of this paper. The page can be found at: http://www.sascommunity.org/wiki/A_Poor/Rich_SAS_Users_Proc_Export

SUMMARY AND CONCLUSION

The purpose of the present paper was to create and describe alternative menu driven methods one could use for exporting SAS datasets to Excel. Seven methods were provided, each using 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 the INSIGHT software, 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.

DISCLAIMER

The content of this paper is the work of the authors and does not necessarily represent the opinions, practices or recommendations of the authors' organizations.

CONTACT INFORMATION

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

Arthur Tabachneck, Ph.D., President
myQNA, Inc. Thornhill, ON Canada
E-mail: atabachneck@gmail.com

Tom Abernathy
Pfizer, Inc.
235 E. 42nd Street
New York, NY 1001
E-mail: tom.abernathy@pfizer.com

Matthew Kastin
I-Behavior, Inc.
Penn Valley, PA
E-mail: matthew.kastin@gmail.com

REFERENCES

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

APPENDIX I

/*THE %EXPORTXL MACRO*/

```
%macro exportxl(libnm,filenm,type,usenames,range);
  %let filepath=%sysfunc(pathname(&libnm.));
  %if &type. eq P %then %do;
    proc export
      data=&libnm..&filenm.
      outfile= "&filepath.\&filenm..xlsx"
      dbms=xlsx
      replace;
    run;
  %end;
  %else %do;
    proc fcmp outlib=work.func.util;
      function c2cb(lib $,mem $, usenm $);
        rc=filename('clippy',' ','clipbrd');
        if rc ne 0 then return(1);
        fid=fopen('clippy','o',0,'v');
        if fid eq 0 then do;
          rc = filename( 'clippy' );
          return(2);
        end;
        dsid=open(catx('.',lib,mem));
        if dsid eq 0 then do;
          rc=fclose(fid);
          rc=filename('clippy');
          return(3);
        end;
        rc=attrn(dsid,'any');
        if rc ne 1 then do;
          rc=fclose(fid);
          rc=close(dsid);
          rc=filename('clippy');
          return(4);
        end;
        nvar=attrn(dsid,'nvar');
        array v[1] /nosymbols;
        call dynamic_array(v,nvar);

        do i = 1 to nvar;
          v[i]=ifn( vartype(dsid,i)='C',1,2);
          if usenm eq 'YES' then do;
            if i gt 1 then rc=fput(fid,'09'x);
            rc=fput(fid,varname(dsid,i));
          end;
        end;
        if usenm eq 'YES' then rc=fwrite(fid);

        do i=1 to attrn(dsid,'nlobs');
          rc=fetchobs(dsid,i);
          do j=1 to nvar;
            if j gt 1 then rc=fput(fid,'09'x);
            if (v[j] eq 1) then rc=fput(fid, getvarc(dsid,j));
            else do;
              fmt=varfmt(dsid,j) ;
              if missing(fmt) then fmt='best12.';
              rc=fput(fid,putc(putn(getvarn(dsid,j ),fmt),'$char12.'));
            end;
          end;
          rc=fwrite(fid);
        end;
        rc=fclose(fid);
        rc=close(dsid);
        rc=filename('clippy');
```

```
      return(0);
    endsub;
  quit;
  %local cmplib;
  %let cmplib=%sysfunc(getoption(cmplib));
  options cmplib=(work.func);
  %put %sysfunc(c2cb(&libnm,&filenm,&usenames)) ;
  options cmplib=(&cmplib);

  %if &range. eq YES %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 dsn $41. required=yes
      attr=underline color=yellow;
      DISPLAY range blank;
      call symput('range',range);
      stop;
    run;
   %end;
   %else %do;
     %let range=A1;
   %end;

  %if &type. eq S %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';

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

      put 'objExcel.Visible = True';
      script='Newbook.Sheets("Sheet1").Range("'||"&Range."||'").Activate';
      put script;
      put 'Newbook.Sheets("Sheet1").Paste';

      put 'Newbook.Sheets("Sheet1").Select';
      put 'Newbook.Sheets("Sheet1").Name = "'@;
      put "&filenm."@;
      put '"';

      put 'Newbook.SaveAs("'@;
      put "&filepath.\&filenm..xlsx"@;
      put '")';

      put "Newbook.Close";

      put 'objExcel = Nothing';
      put 'Newbook = Nothing';
    run;

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