**PhUSE 2010**

**Paper CC01**

# AN ANIMATED GUIDE: SENDING SAS FILE TO EXCEL

Russ Lavery, Contractor for K&L Consulting Services, King of Prussia, U.S.A.

## ABSTRACT

The primary purpose of this paper is to provide a generic DDE macro to easily write a SAS data set to Excel. Secondarily, the paper compares methods of writing SAS data to an Excel file. SAS programmers want to easily write SAS files to tabs in Excel Workbooks (with and without named ranges). This paper examines the nine situations one can encounter when writing SAS data to an Excel-created named range.

The DDE program in the appendix will allow the reader, with minimal typing, to experiment with the methods described in this paper. The program is heavily commented and will not be explained

## INTRODUCTION:

The purpose is to provide a generic DDE macro to write a SAS data set to Excel and to compare methods of writing SAS data to an Excel file.

Using DDE to put a SAS data set into Excel can be annoying. The DDE syntax is fussy and the error messages are not the most helpful. Programmers often keep a snippet of code stored on a drive and modify that snippet when they want to write to Excel. The macro in this paper can be put in a user's abbreviation file (and I suggest that everyone have an abbrev file -please see paper citations at end) and then called when the programmer wants to use DDE to send files to Excel. The macro provided is a very general write-to-Excel macro and, especially when put in an abbrev file, can make it very easy to send a SAS data to Excel using DDE. There is a section in the macro, in yellow, that is not general, but is included as an example of how to put information (like a run date) into an xls sheet to make the sheet self-documenting.

The SAS libname engine writes and reads Excel files. Using the libname engine to write to a named range is very useful when a programmer wants to write the "back page" of an XLS workbook and update the data in an Excel-based dashboard.

As a review, a named range is a way of referring to a group of cells and creating named ranges is easy in Excel.

The process is simply to highlight a block of cells and then type the name of the named range in the "name box".

In the figure to the right, we are creating a named range called ThreeByThree.

Named ranges like "Sales_2007", "Sales_2008" and "Sales_2009", when used in formulas, are self documenting and make understanding the workbook logic much easier.

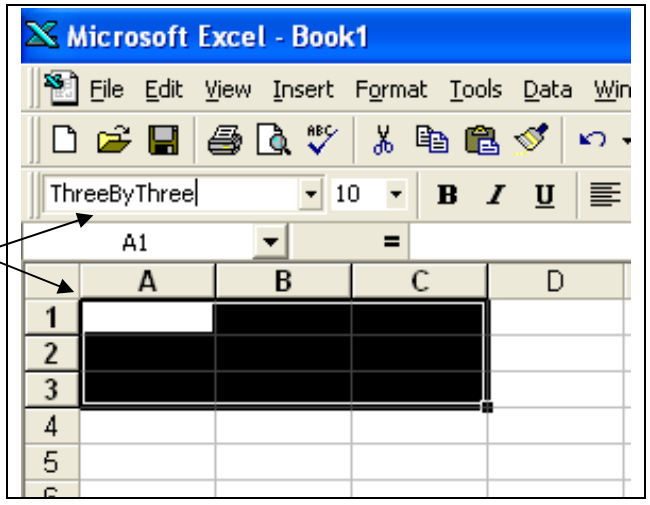



FIGURE 1

Named ranges are a useful Excel feature..

You can replace the cell references, in a formula, with a named range.

You can use the named range ThreeByThree in a formula as is shown in the graphic to the right.

Instead of specifying the cells to sum, the formula says sum all the cells in the named range ThreeByThree.

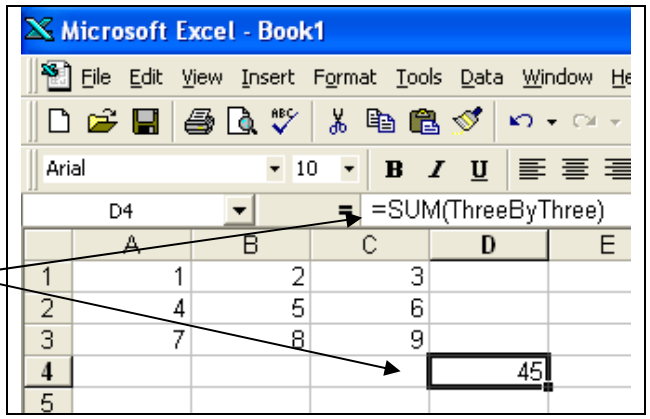



FIGURE 2

The SAS libname engine is an alternative to PROC IMPORT and PROC EXPORT and can be used to both create an Excel workbook and to write to it. The code below illustrates some of the features and limitations of this method.

This libname tells SAS to create, and write to, the Workbook NewWkbkV3.xls.

We easily write data to Excel. Immediately after the run executes a programmer would not be able to open NewWkbkV3.xls in Excel.

At this point in code execution, the workbook is locked by SAS. If a programmer wants to see the results of her/his writing s/he must clear the libname.

After clearing the Libname, we can see what is in the Excel workbook.

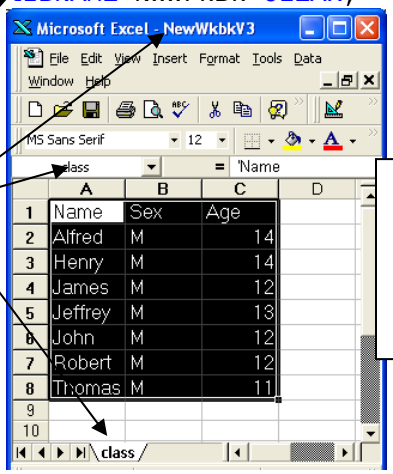
There is a workbook, named NewWkbkV3.xls with a named range called class on a page called class.

```
LIBNAME NwWrkBk EXCEL 'E:\Phuse Papers
2010\SendingFilesToExcel\NewWkbkV3.xls' ;

DATA NwWrkBk.class; /*Create a workbook and named
range*/
SET sashelp.class(drop=weight height);
Where sex="M" and age LT 15;
RUN;
```

```
NOTE: There were 7 observations read from the data set SASHELP.CLASS.
      WHERE (sex='M') and (age<15);
NOTE: The data set NWWRKBK.class has 7 observations and 3 variables.
```

```
LIBNAME NwWrkBk CLEAR;
```




We wrote out 7 observations and 3 variables to the named range.

The first line contains names of variables.

FIGURE 3

One often (well, at least one sometimes) needs to fix mistakes and re-write the new answers to Excel.

A good question is; "what happens if you want to fix a mistake and then re-send the data to the same workbook/sheet?

Re-establish the libname and run the code to the right.

You can not.
The libname engine does not support simple overwriting of a named range.

```
/*What happens if you made a mistake & need to re-
write the data*/

LIBNAME NwWrkBk EXCEL 'E:\Phuse Papers
2010\SendingFilesToExcel\NewWkbkV3.xls' ;
DATA NwWrkBk.class;/*Fails- can't overwrite a named
range*/
SET sashelp.class(drop=weight height);
Where sex="M" and age LT 13;
RUN;
```

```
ERROR: The MS Excel table class has been opened for OUTPUT. This table
already exists, or there is a name conflict with an existing object. This
table will not be replaced. This engine does not support the REPLACE
option.
```

You can re-send the information using a multi-step process.

First, you clear the named range with a PROC DATASETS.

```
PROC DATASETS LIB= NwWrkBk; /*This clears information
from the named range*/
DELETE class;
Quit;
```

```
NOTE: Deleting NWWRKBK.class (memtype=DATA).
```

At this point, we will take a small "detour from developing the main issue" to show more information about named ranges and SAS. The detour will be contained inside the macro called skip.

Please look at the picture to the right. Above, we executed the PROC DATASETS to delete the named range and the libname has been cleared. We then opened the Excel workbook and selected the named range "class".

You can see, after the execution of the PROC DATASETS, that the workbook exists and the named range "class" still exists. No values are in the named range. Note that the named range "class" seems to have 8 rows and three columns.

The point of this detour was to show that executing a PROC DATASETS does not delete either the workbook or the SAS created named range. We created this named range by using the SAS libname engine and not through Excel commands,

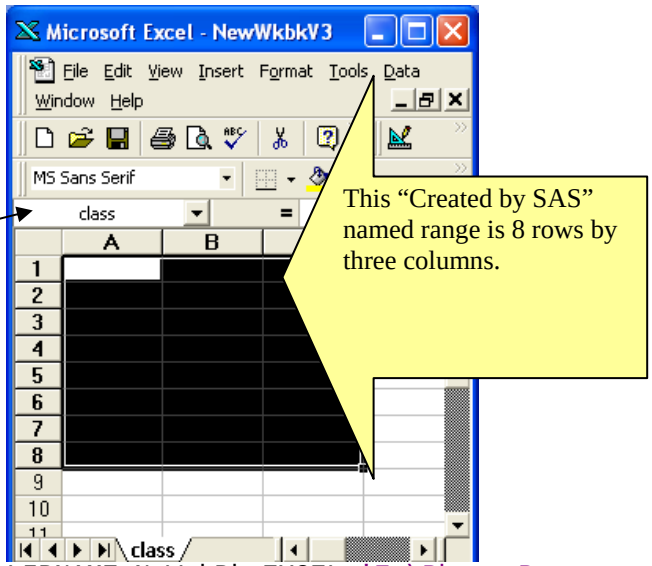
We must close the XLS workbook before proceeding.

Because of the detour, we need to establish the libname again.

This ends the detour.

```
%macro skip;
LIBNAME NwWrkBk Clear ;
/*Look at the xls file*/

Below we see the destination workbook opened
in Excel and the named range is displayed.
```




```
LIBNAME NwWrkBk EXCEL 'E:\Phuse Papers
2010\SendingFilesToExcel\NewWkbkV3.xls' ;
/*Now that the range is clear, write to it*/
%mend skip;
```

FIGURE 4

Use this data step to send new data to the named range.

```
DATA NwWrkBk.class;
  SET sashelp.class;
RUN;
```

```
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set NWWRKBK.class has 19 observations and 5 variables.
```

Clear the Libname so we can open the XLS sheet.

```
LIBNAME NWWRKBK CLEAR ; /*LOOK AT THE XLS FILE*/
```

Using the libname engine, we were able to write a 19 row by 5 column data set to a SAS-created named range that, at one time, seemed to have the dimensions of 8 rows by 3 columns (FIGURE 4).

This raises an issue. When writing to a named range, do the dimensions of the named range and the data set need to "agree".

Before we answer this question we should emphasize that there are two types of named ranges.

One type is created by SAS as we just did. This type of named range seems to be very adaptable.

By adampable I mean, if a re-run of a program sends more *rows and columns*, to the XLS sheet, than the first run of the program did, SAS and the SAS created named range adapt well to the new situation.

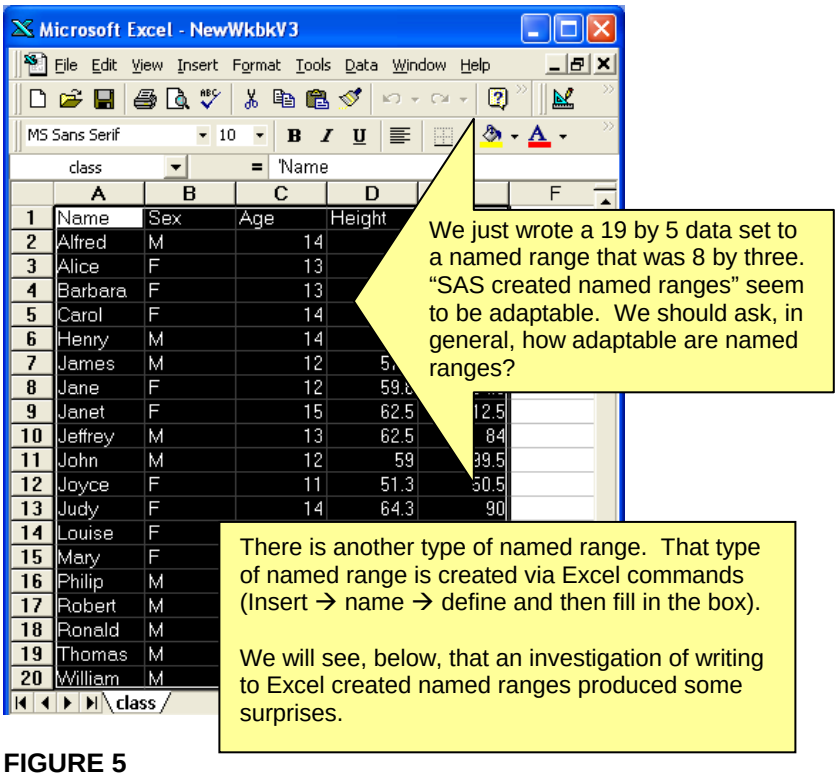



**FIGURE 5**

Re-establish the libname engine with scan_text=NO.

A PROC Append is commonly used to add data to the named range. Here we drop a variable so that we append 4 variables to a named range that can hold five.

Note, in the panel below, the warnings that SAS sends.

```
LIBNAME NwWrkBk
 EXCEL 'E:\Phuse Papers
2010\SendingFilesToExcel\NewWkbkV3.xls'
  SCAN_TEXT=NO;

Data girls(drop=age);
set sashelp.class;
where sex="F";
run;

Proc Append
 base=NwWrkBk.class
 data=girls;
quit;
```

The Excel workbook can not be viewed until the libname has been cleared.

We see the appending was successful but we get a warning.

In summary, using the Excel libname engine to write to a SAS created named range seems quite flexible.

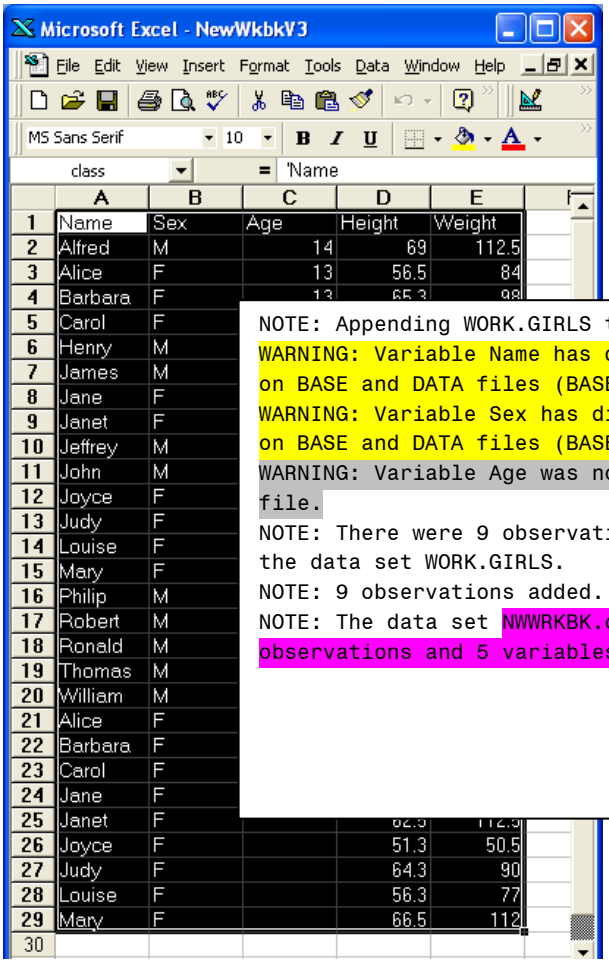
Data can not be overwritten but a PROC DATASETS can be used to remove old data values and, after that, SAS can again write data to a named range.

There does not seem to be an issue with the dimensions of the SAS data set not matching the dimension of existing named ranges - if SAS has created the named range.

In general, PROC APPEND is used to add data to an existing named range.

To eliminate offensive notes in the log, one needs to manage the lengths of created variables.

```
LIBNAME NWWRKBK CLEAR ; /*LOOK AT THE XLS FILE*/
```




```
NOTE: Appending WORK.GIRLS to NWWRKBK.class.
WARNING: Variable Name has different lengths
on BASE and DATA files (BASE 255 DATA 8).
WARNING: Variable Sex has different lengths
on BASE and DATA files (BASE 255 DATA 1).
WARNING: Variable Age was not found on DATA
file.
NOTE: There were 9 observations read from
the data set WORK.GIRLS.
NOTE: 9 observations added.
NOTE: The data set NWWRKBK.class has .
observations and 5 variables
```

**FIGURE 6**

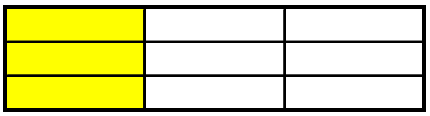
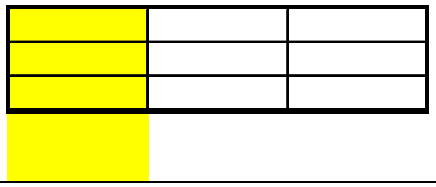
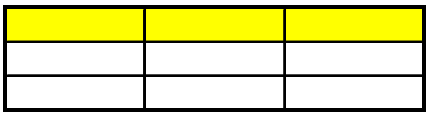
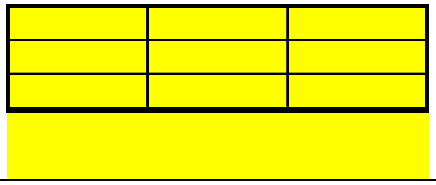
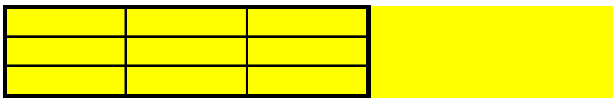
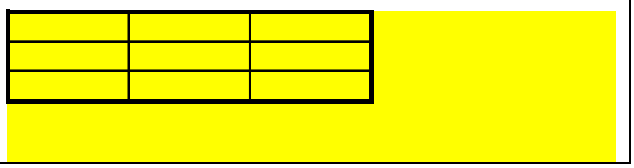
There is a second type of named range: one that is created using Excel commands. This type of named range is worth studying because SAS is often used to write data to a hidden page in a template for an XLS based dashboard. Many dashboards are created by having SAS crunch large amounts of data and writing summary informaiotn ot a hidden Excel page in a dashboard. In these situations, Excel is used to surface the data to visible pages of the dashboard and the destination to which SAS sends data can be an Excel-created-named range on a hidden sheet.

Often this writing to a hidden sheet occurs on a monthly basis and the size of the data being written changes every month. Changes in monthly business activity, activity on which the Dashboard reports, can cause more or fewer rows to be written each month. The next section investigates issues caused by more or fewer columns and rows being written to an excel-defined named range

The graphic on the next page summarizes what happens when you try to write a SAS file to a named range that has row and column dimensions that can be smaller than, equal to or larger than the size of the data set. This paper will send a 3 by 3 data set to a series of Excel created named ranges. The named ranges, in both rows and columns, will be smaller, the same size as, and then larger than the 3x3 SAS file.

The graphics below explore the results of writing a SAS data set to an Excel-created named range with similar and different dimensions. We have seen, for SAS-created named ranges, that a named range will expand – both in the number of rows and in the number of columns – to accept more data.

This section explores the ability of Excel-created named ranges to expand as SAS feeds in data. From the testing done for this paper, it seems that Excel-created named ranges behave differently from SAS-created named ranges. Please review the nine situations pictured, and described, below.

| | |
|---|---|
| 1) Attempt to write a 3 by three SAS data set (bordered squares) into a named range that was defined as a 1 by 1 (yellow box). The SAS file has more rows and more columns than the named range. **Result: Failure.** |  |
| 2) Attempt to write a 3 by three SAS data set (bordered squares) into a named range that was defined as a 3 by 1 (yellow boxes). The SAS file has the same number of rows as the named range but more columns. **Result: Failure.** |  |
| 3) Attempt to write a 3 by three SAS data set (bordered boxes) into a named range that was defined as a 5 by 1 (yellow boxes). The SAS file has fewer rows than the named range but has more columns. **Result: Failure.** |  |
| 4) Attempt to write a 3 by three SAS data set (bordered boxes) into a named range that was defined as a 1 by 3 (yellow boxes). The SAS file has more rows than the named range but the same number of columns. **Result: Success – The Excel named range expands to include more rows.** |  |
| 5) Attempt to write a 3 by three SAS data set (bordered boxes) into a named range that was defined as a 3 by 3 (yellow boxes). The SAS file has the same number of rows and columns as the named range. **Result: Success** |  |
| 6) Attempt to write a 3 by three SAS data set (bordered boxes) into a named range that was defined as a 5 by 3 (yellow boxes). The SAS file has fewer rows than the named range but has the same number of columns. **Result: Success.** |  |
| 7) Attempt to write a 3 by three SAS data (bordered boxes) set into a named range that was defined as a 1 by 5 (yellow boxes). The SAS file has more rows than the named range but fewer columns. **Result: Success – The Excel named range expands to include more rows.** |  |
| 8) Attempt to write a 3 by three SAS data (bordered boxes) set into a named range that was defined as a 3 by 5 (yellow boxes). The SAS file has the same number of rows as the named range but fewer columns. **Result: Success.** |  |
| 9) Attempt to write a 3 by three SAS data (bordered boxes) set into a named range that was defined as a 5 by 5 (yellow boxes). The SAS file has fewer rows and fewer columns than the named range. **Result: Success.** |  |

## THE MACRO

The macro included in the appendix does not delete any existing data. To avoid problems caused by multiple writes to the same destination it relies on the mechanism of reading a template and saving it under a new name. This will overwrite the results of the previous call of the macro. This macro does not support appending of data.

Some features of the code are:
It is easy to understand.
Writing does not have to start in Row1 Col1.
With repeated calling of the macro, several data sets can be sent to the same worksheet.
With the last write to excel, the macro will send data to specific cells on a specific page (useful for adding run dates to output files) and then save the XLS workbook to a specific location.

Most of the code is general, but attention should be paid to the section, in yellow, that is not general. This is the section that writes specific data when the programmer has instructed SAS to close the XLS sheet. The specifications for this action are not standard enough to be coded for all uses. It is hoped that the examples are enough to allow a novice user to deduce the underlying logic and then make the changes s/he might desire.

You can copy the macro from this paper or make it always available via a SAS abbrev file. I suggest installing the abbrev file for access to this macro and for many other reasons. The abbrev file is at www.datameans.com.
Please read http://www.lexjansen.com/phuse/2008/cc/cc07.pdf to learn about abbrev files.

## CONCLUSION

A general SAS macro to write a SAS data set to Excel is provided in the appendix. This is the major deliverable of the paper.

It seems, while the Excel Libname engine can not overwrite data in a SAS defined named range, it is powerful and flexible. Appending to a named range is done via PROC APPEND.

It seems that Excel-created named ranges expand only in the row dimension.
It seems that a programmer wishing to write to an Excel-created named range (possibly on a hidden sheet in a "template" should define a one row by many-more-columns-than-is-expected named range.

## REFERENCES:

Richardson , Kari, " 2004, Using Macros to Automate SAS® Processing", *Proceedings of the Twenty-ninth annual SAS ® Users Group International Conference, paper 126-29,* http://www2.sas.com/proceedings/sugi29/126-29.pdf

Hamilton, Paul, 2005, "*%SYSFUNC: Extending the SAS Macro Language*" Proceedings of the Pacific North West SAS Users Group, http://www.lexjansen.com/pnwsug/2005/how/SysFunc.pdf

Ceranowski, Elizabeth, 2009 , "SAS® Abbreviations Are Your Friends; Use a Template Method to Code!", http://support.sas.com/resources/papers/proceedings09/073-2009.pdf

Grant, Paul, 2009, "Creating Code Templates in the SAS® Enhanced Editor using Abbreviations and User Defined Keywords", http://support.sas.com/resources/papers/proceedings09/077-2009.pdf

Choat and Martell, "De-Mystifying the SAS® LIBNAME Engine in Microsoft Excel: A Practical Guide", *Proceedings of the Thirth-first annual SAS ® Users Group International Conference,* http://www2.sas.com/proceedings/sugi31/024-31.pdf

Owl, Benjamin, 2010, "The Little Engine That Could: Using EXCEL LIBNAME Engine Options to Enhance Data Transfers between SAS® and Microsoft® Excel Files" , *Proceedings of the 2010 PharmaSUG Conference*

## CONTACT INFORMATION

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

Russ Lavery Independent Contractor Philadelphia Area - USA Email: Russ@Russ-Lavery.com

APPENDIX

Create a blank worksheet if you want a to test sending data to a template. This learning template does not have to be realistic.

Create some macro variables that can be used in **manually stepping through** the macro.

```
/*************************************************************
Program: XXXXXXXXXXXXXXXX
by:      Several
Programming Owner:
Business (report) Owner:

Purpose: use DDR to write to excel

infiles: Soecified via parameters

outfiles:Specified via paramater

Use Instructions: Thsi can be called many times to write data to several XLS sheets

Modifications:

*************************************************************/
%macro skip;
%let FirstWrite=Y;       /*A Y here will cause SAS to start Excel.
                           AND MUST be valued as Y for the Macro to open an Excel template
                           - used for first call fo XLS sheet*/

%let SASFile=SASHelp.class;           /*File to send to the XLSDataTab Xls sheet*/
%let whereClause=%str(where Sex="F";);  /*a complete where statement with semicolon*/
                          /*eg:     %str(where 1=1;)    */
%let DropVars=%str((drop=sex));         /*a complete "drop data set option" with
                          /* ()and withOUT semicolon  eg; (drop=SortOrder4Prod) */

%let XLSDataTab=Sheet2;   /*Tab to which to write the table in SASFile*/
%let XLSSmryTab=Sheet1;   /*Often,at the end of loading a template,
                        we write one or two pieces of info to a summary tab*/
%let StartRowNo=5;  /*You do not have to start writing in R1C1 -Look at the template-*/
                     /*See where the data section starts*/
%let StartColNo=1; /*You do not have to start writing in R1C1 -Look at the template-*/
                    /*See where the data section starts- sometimes we start at column 2*/
%let templateLoc=%str(C:\test\template.xls);
                     /
                     /*
%let SaveNowYN=N;    /
                     /
%let NewName=C:\Test\

%mend skip;

options mprint mcompilenote=all source source2;
```

**The command x "C:\Program Files\Microsoft Office97\Office\EXCEL.EXE"; will not work**
**The real DOS path can not contain directory names longer than 8 characters.**
**The real DOS path often has ~ in it. Your path may differ from the one in this paper.**

You can call the macro several times to write to several sheets in a workbook – without closing the workbook. We will do this in the examples.

Note the path. Your path may differ.

```
%MACRO Write2XlsV3( /*Write ONE SAS file to ONE Tab of an Excel Workbook*/
  FirstWrite=       /*A Y here will cause SAS to start Excel.
                          AND MUST be valued as Y for the Macro to open an Excel template
                          - used for first call fo XLS sheet*/

 ,SASFile=       /*File to send to the XLSDataTab Xls sheet*/
 ,whereClause=  /*a complete where statement eg:     %str(where 1=1;)    */
 ,DropVars=       /*a complete "drop data set option" with
                                    and withOUT semicolon  eg; (drop=SortOrder4Prod) */
 ,XLSDataTab=   /*Tab to which to write the table in SASFile*/
 ,XLSSmryTab=   /*Often,at the end of loading a template,
                  we write one or two pieces of info to a summary tab*/
 ,StartRowNo=5  /*You do not have to start writing in R1C1 -Look at the template-*/
                /*See where the data section starts*/
 ,StartColNo=1 /*You do not have to start writing in R1C1 -Look at the template-*/
               /*See where the data section starts- sometimes we start at column 2*/
 ,templateLoc=  /*path to the template.  If you value this AND set firstWrite to Y*/
                /*, the macro will open the template*/
 ,SaveNowYN=N   /*Saves & Closes Excel.  After writing last tab in a multi-tab SS,*/
                /*change to Y & enter path in NewName*/
 ,NewName=C:\Temp\DashBoard.xls /*save the XLS sheet under this name.
                                  Saving only happens when SaveNowYN=Y  */
  );

  %macro OpenExcel;
         %if (&FirstWrite =Y) or (&FirstWrite =y) or (&FirstWrite =YES) %then
         %do; /*This is the first time we write to the XLS sheet - we need to open it*/
                options noxwait noxsync;
                /* The Command below will not work!
                x "C:\Program Files\Microsoft Office97\Office\EXCEL.EXE";
                  Use the command dir /X to see the real path */
                 x "C:\Progra~1\MIFF2D~1\Office\EXCEL.EXE";
                DATA _NULL_;
                 Rc = SLEEP(5);
                RUN;
         %end;
 %mend OPenExcel;
 %OpenExcel;




%macro OpenTemplate;
  /*This macro can be used to write to several pages to a new SS, or template,
```

The template opens if two conditions are met.

If you have run the macro before, clean up the working files to prevent any possible confusion.

You might want to filter on a variable you do not want to send to Excel.

```
     and we ONLY open the Template: 1) on the first call and 2) if it exists
    - We Must tell the macro if this is the first call and if the template exists*/
       %if ( (%length(&templateLoc) GT 0 ) /*if GT 0, User has specified a template*/
         and ( %upcase(&FirstWrite =Y) or (%upcase(&FirstWrite) =YES ))
            ) %then
          %do;            /*if above is true, user has specified a template to which he wants to write*/
          /*if no template specified, write to the sheet that opens when we open excel*/
              Filename Excel dde 'EXCEL|SYSTEM';
              data _null_;
               file excel;
               put "[open(""&TemplateLoc"")]";
               run;
            %end;
                %else
                %do;
                      %put No template specified;
                %end;
 %mend OpenTemplate;
 %OpenTemplate;

 %macro CleanUp;
        %if %sysfunc(exist(GoodObs))NE 0 %then
              %do;
          Proc SQL;
           Drop table GoodObs;
           Drop table VarsToExport;
          quit;
       %end;
         %else
         %do;
       %put NO table cleanup required - no table to drop;
         %end;
 %mend CleanUp;
 %cleanUp;

/*We might want to filter observations on a variable we do not want to send to excel*/
/*Filter observations ASAP and drop variables in the next step*/

 Data GoodObs&DropVars;
    set &SASFile;
    &whereClause;      /*<-----  Drop rows you do NOT want to print*/
     /*This, and multiple calls of the macro,
       can be used to split files that are too large to fit on an XLS sheet*/
 run;




/*for the DDE, We need to know how many obs and variables are in the file we are exporting.
```

We used to, sometimes, overload Excel. If there are more rows than Excel can handle, the write does not happen but there is no message in the log.

This is the trick. Use macro

```
 Put the number of rows and columns into macro variables so we can dimension the DDE writing area*/
/*Note &DropVars.  You do not have to export all the variables */
Proc Contents data=GoodObs varnum noprint
                           out=VarsToExport ;
run;/* create data set with variables info--*/

Proc Sort data=VarsToExport ;
    by varnum;
run;

Data _null_;   /*How many variables - This number is used to dimension the DDE "writing" area*/
     if 0 then set VarsToExport Nobs=NOfObs;
     call symput("NoOfVars",Strip((put(NOfObs,best12.))));
run;
%put We have &NoOfVars Variables to export from &&SASFile ;

Data _null_;    /*in the DDE, We need to be able to loop and "write" the names of the variables*/
     set VarsToExport(keep=name);
     /*create an array of macro variables*/
    call symput("vn"||left(Put(_n_,7.)),name); /*DANGER hardcode*/
run;

%macro LeaveRecord;
   /*As a debugging tool, write the variables we want to export in the log*/
  %do QC=1 %to &NoOfVars;
       %put for loop number &qc the variable we write is  &&vn&QC;
  %end;
%mend LeaveRecord;
%LeaveRecord;

Data _null_; /*How many rows - This number is used to dimension the DDE "writing" area */
   if 0 then set GoodObs Nobs=NOfObs;
   call symput("NoOfRows",Strip(NOfObs));
   If NOfObs GT 3 then
      do;  /*DANGER HARDCODE*/
        put "************************";
        put "WA" "RNING **  some version of Excel can hold only 66,000 rows";
        put "There are " NOfObs " rows in this file.";
        put "    Check to see if the might fill up an XLS sheet";
        put "************************";
      end;
run;

%put  After applying the where clause, we want to print &NoOfRows rows from the file &SASFile;




filename blah dde
```

variables to calculate the size of the “writing area”.

Use a loop to write the put statement.

Often there is a Management Sheet, or Summary Sheet, page in the front of the workbook. You write the run date and other info to it before you close the workbook.

This is the departure from generality. There are several hardcodes in this example. Please modify the code to meet your needs

If this is the first

```
"excel|&XLSDataTab!R&StartRowNo.C&StartColNo.:R%eval(&StartRowNo+&NoOfRows)C%eval(&StartColNo+&NoOfVars)"   NOTAB;
/* in statement above, we dimensioned the DDE "writing" area */

Data _null_; /*this data null reads the GoodObs table and does the writing to excel*/
    set GoodObs;
    file blah LRECL=2050;
       /*The value of Lrecl can range from 1 to 1,048,576 (1 megabyte). According to the
          experience of SAS experts,  an Lrecl of 8192 covers 99% of the cases that most
         SAS programmers encounter.       */
       %macro LoopOverVars;    /*use a loop to list all the variable names in the DDE statement*/
            put  %do k=1 %to &NoOfVars;
                    &&vn&k "09"X
                 %end ;
       %mend LoopOverVars;
    %LoopOverVars;
run;

%if (%upcase(&SaveNowYN) =Y) or (%upcase(&SaveNowYN) =YES) %then
%do;  /*If you tell the macro you have written the last data tab, it will
      1)** Write any information to the "summary sheet"
      2)** Put the date in Cell A2 of Sheet 1
      3)** Save the file under a new name  */
      Filename Sumry  dde "excel|&XLSSmryTab!R1C1:R3C2" NOTAB; /*HARDCODED range*/
               /*This writing should not ”Overlap" the range where we write data*/
              Data _null_; /*HARDCODE -  /*We write the 6 cells below into a 3 by 2*/
                 file Sumry LRECL=2050;
                   Put "Put in Cell A1" "09"X  "Put in Cell A2" "09"X ;
                   Put "Run On:" "09"X         "%sysfunc(Date(),worddate18.)" "09"X ;
                   Put "Put in Cell A3" "09"X  "Put in Cell B3" "09"X ;
              run;

              Filename Final  dde "excel|&XLSSmryTab!R1C1:R1C1"  NOTAB;   /*HARDCODE*/
               /*(above) Make &XLSSmryTab active before saving then (below)Save and rename*/
                filename outexcel dde "EXCEL|SYSTEM";
                  data _null_;
                        file outexcel;
                        put "[save.as(""&NewName"")]";
                        put "[close]";
                        put "[quit]";
                       run;

%end;
%Mend Write2XlsV3;




%Write2XlsV3( /*Write ONE SAS file to ONE Tab of an Excel Workbook*/
```

call of the macro (&FirstWrite is properly valued), and the macro opens Excel..

Write to sheet 2.

Enter a path, If you are writing to a template.

*You can also call the macro when no template exists.*

This call of the macro writes to sheet 3 and then to the summary ta

The template was specified when we opened Excel, so we do not need to do it here.

```
  FirstWrite=Y       /*A Y here will cause SAS to start Excel.
                       AND MUST be valued as Y for the Macro to open an Excel template
                       - used for first call fo XLS sheet*/

 ,SASFile=SASHelp.class          /*File to send to the XLSDataTab Xls sheet*/
 ,whereClause=%str(where sex="F";)  /*a complete where statement eg:     %str(where 1=1;)    */
 ,DropVars=%str((drop=sex))        /*a complete "drop data set option" with ()*/
                                  /*and withOUT semicolon  eg; (drop=SortOrder4Prod) */

 ,XLSDataTab=Sheet2   /*Tab to which to write the table in SASFile*/
 ,XLSSmryTab=   /*Often,at the end of loading a template,  */
                 /*we write one or two pieces of info to a summary tab*/
 ,StartRowNo=5  /*You do not have to start writing in R1C1 */
                /* -Look at the template- See where the data section starts*/
 ,StartColNo=1 /*You do not have to start writing in R1C1 -Look at the template-    */
                /*See where the data section starts- sometimes we start at column 2*/
 ,templateLoc=%str(c:\test\template.xls)
        /*path to the template.  If you value this AND set firstWrite to Y*/
        /* the macro will open the template*/
 ,SaveNowYN=N   /*Saves & Closes Excel.*/
                /*After writing last tab in a multi-tab SS,*/
                /*change to Y & enter path in NewName*/
 ,NewName= /*save the XLS sheet under this name.*/
          /*Saving only happens when SaveNowYN=Y  */
  );


%Write2XlsV3( /*Write ONE SAS file to ONE Tab of an Excel Workbook*/
  FirstWrite=N       /*A Y here will cause SAS to start Excel.
                       AND MUST be valued as Y for the Macro to open an Excel template
                       - used for first call fo XLS sheet*/

 ,SASFile=SASHelp.class          /*File to send to the XLSDataTab Xls sheet*/
 ,whereClause=%str(where sex="M";)  /*a complete where statement eg:     %str(where 1=1;)    */
 ,DropVars=%str((drop=sex))        /*a complete "drop data set option"*/
                                 /* with () and withOUT semicolon  eg; (drop=SortOrder4Prod) */
 ,XLSDataTab=Sheet3   /*Tab to which to write the table in SASFile*/
 ,XLSSmryTab=Sheet1   /*Often,at the end of loading a template,
                  we write one or two pieces of info to a summary tab*/
 ,StartRowNo=5  /*You do not have to start writing in R1C1 -Look at the template-*/
                /*See where the data section starts*/
 ,StartColNo=1 /*You do not have to start writing in R1C1 -Look at the template-*/
               /*See where the data section starts- sometimes we start at column 2*/
 ,templateLoc=  /*path to the template.  If you value this AND set firstWrite to Y  */
                /*, the macro will open the template*/
 ,SaveNowYN=Y   /*Saves & Closes Excel.  After writing last tab in a multi-tab SS,*/
                 /*change to Y & enter path in NewName*/
 ,NewName=C:\Test\GirlsAndBoys.xls /*save the XLS sheet under this name.*/
                                 /* Saving only happens when SaveNowYN=Y  */
  );
```