

YOU USE SAS, YOUR BOSS USES EXCEL, GUESS WHERE YOUR RESULTS ARE GOING TO APPEAR ! THREE PROCESSES TO HELP PUT YOUR SAS DATA AND RESULTS INTO EXCEL

William E Benjamin Jr, Phoenix, Arizona

ABSTRACT

This paper describes three general processes helping programmers move their data from SAS® readable file structures into Microsoft Excel® spreadsheets. The processes include trivial data output solutions, simple DDE tasks, and progress to complex SAS and Excel routines that Manipulate Excel from SAS and then access SAS data from Excel. This paper will assume the reader can access the SUGI 27, and SUGI 28 Proceedings, either online or from the CD's available from the conferences. SAS code and macros will be demonstrated to push data into Excel, execute Excel macros, and present data to Excel users. This paper further assumes the reader has some knowledge of SAS macros, Excel macros, and Visual Basic Programming. Finally, Excel and Visual Basic code will pull data into the spreadsheet and give the programmer the ability to build an Excel macro to exploit the data and display information for the end user. As a result the SAS programmer will have tools that will allow the creation of spreadsheets for distribution, the automation of standardized reports, and the building of powerful spreadsheet applications with access to information built by offline SAS processes that the end user can refresh.

INTRODUCTION

How often have you heard the words, "Just e-mail me a spreadsheet with the data."? This week? Or even today? Well, if you have been programming for any length of time, and your boss is more than a few months younger than you, then you hear it more often than you want to hear that request. In the days when Programmers worked on Mainframes, and Managers worked on PC's, the Programmers never got those requests, and Managers keyed in the data themselves. But, then came down-sizing, and right-sizing, and left-sizing, and up-sizing, and do-whatever-you-must-to-keep-a-job-sizing. So, now you get "THE REQUEST" for "THE SPREADSHEET", and the file is too big to re-enter. Now what?

PROCESS ONE (THE TRIVIAL SOLUTIONS)

The trivial solutions send a comma delimited files to your supervisor, and let him or her "Get to Know the Data" by reading it into Excel and arranging it however he or she wants the data, but that only works zero to one times. The trivial solutions are really just part of a smoke screen to stall until you can figure out how to do this a better way. This paper will use data readily available for all users. The data is part of the SAS runtime environment, under the LIBNAME of SASHELP. The paper will also leave as a student exercise the printing of the full dataset, while including some information from the Proc Contents listing at the end of the paper.

Solution 1.a (Export the data using Proc Export menus)

This trivial method uses the File tab and the EXPORT option of SAS (Version 8.2 for PC's.) The following code is created and executed (SAS will let users place the output code into a file to reuse.) This code writes the output file onto Drive C: in directory \wuss_2003 with a file name Shoes_1a.csv. The code may be easily captured, by continuing until asked for a file name to store the SAS code for the future use. Then it can be modified for use with any file.

The following SAS code is produced:

```
PROC EXPORT DATA=SASHELP.SHOES
  OUTFILE="C:\wuss_2003\Shoes_1a.csv"
  DBMS=CSV REPLACE;
```

But, as with most trivial solutions, you have few options and little control over the output file produced. In this case all variables are output, in the order that they were defined, with the variable name as the title of the variable for Excel.

Solution 1.b (Write the data to a file using PUT)

This solution is little more than the first solution, The object is to create a file similar to the one created by the Proc Export commands. The file must be created with titles on the first line and data rows following the titles. Of course there is the question of the delimiters and string processing.

Code similar to the following could be used:

```
Filename out_csv
  "C:\wuss_2003\Shoes_1b.csv";

Data _null_;
  Tab_char = '09'x;
  set DATA=SASHELP.SHOES;
  .
  .
  .
  more SAS code
  .
  .
  .
  file out_csv noprint notitles;
  If _n_ = 1
  then do;
  Put "Total Inventory", Tab_char,
      "Product", Tab_char,
      "Total Sales";
  end;
  else do;
  Put Inventory, Tab_char,
      Product, Tab_char,
      Sales;
  end;
run;
```

At least here you have some control over the output file, and the order of the variables.

Solution 1.c (ODS – Output Delivery System)

OK! OK! ODS is truly not trivial, but it is easy to use ! Turn it on, do what you want, turn it off. What could be more simple? Well, whole books have been written about ODS. The net result is that you still have a "*.CSV" file to give to a boss that never wants to see another one. This is not a "SPREADSHEET", and SAS warns you that it is

experimental in version 8.2. Admittedly “*.CSV” files will be opened by Excel, and look like a spreadsheet, but you may not always have full control over the data when it is input to Excel. The code follows here:

```
ods csv file='c:\wuss_2003\Shoes_1c.csv';
proc print data=SASHELP.SHOES ;

run;
ods close;
run;
```

Well, these three methods all have several things in common, they are easy to use, involve simple code (unless you read the ODS books), and they produce an output file with a similar structure. These files then all need to be loaded into Excel and then used as a “SPREADSHEET”.

PROCESS TWO (DDE - DYNAMIC DATA EXCHANGE, ONE STEP, FROM SAS TO EXCEL)

Here is where the fun starts, DDE has been around for a long time. This is a method of trading data between clients and servers. In this case SAS being a client and Excel being a server. Koen Vyverman has done a very good job of explaining DDE, how it works, and why it works. The magic of DDE is that it boils down to using SAS as a way to enter data directly into an open Excel spreadsheet. Actually, you can write to Microsoft Word © or many other products not addressed by this paper. Note that the condition was that the Excel spreadsheet was open. Because the data transfer is a “real time” “do it right now” type of a data transfer both programs must be running. But, never fear, the SAS application can open the Excel program and close it by issuing simple DDE commands. This frees the user from having to stop doing something to open spreadsheet one, or close spreadsheet two, before doing a second output task.

Getting started, the “Hello World” project

In thirty years of programming this author has begun using many new techniques, and software packages. The usual first task is simple, and involves the “can I really make this work” concept of programming called a “Hello World” project. Other people call it different things, but the concept is the same, if a known text string can be moved from one place and then found or retrieved in a second place then the project is a success. So, we will start with a simple project, put the words “Hello” and “World” into an Excel spreadsheet, and find them.

This code writes to an open Excel session from an open SAS session:

```
filename ddewrite dde
'excel|[ddewrite.xls]Sheet1!r1c1:r1c2';

data out;
  file ddewrite;
  x='Hello World';
  put x;

run;
```

The beauty of a “Hello World” project is that it is trivial too. Most of the code listed above is something that nearly every SAS programmer will recognize as commonly used features of the SAS system. In fact this paper has already used all of the SAS commands in this segment of code already. (Well OK most of it) Let us look at the code a little closer. In SAS if you count the number of semicolons that are not

hidden by quoted strings you can find out how many instructions have been executed. This code has six semicolons, therefore six instructions. So, if we examine them in order we find the following:

(1) Filename ddewrite dde

```
'excel|[ddewrite.xls]Sheet1!r1c1:r1c2';
```

This FILENAME instruction links an external file “DDEWrite.xls” using a DDE “engine” that points to a small space in an open excel worksheet, using what is called a “DDE Triplet”. The coding convention of a triplet, ‘excel|[ddewrite.xls]Sheet1!r1c1:r1c2’, should be easy to see here too. Since SAS is acting as a client in this client/server pair we can identify Excel as the server, we have already noticed that DDEWrite.xls is the open Excel spreadsheet, the actual page of the spreadsheet is selected by the rest of triplet “Sheet1!r1c1:r1c2”. So by examining the spreadsheet by switching from the SAS session to the Excel session the “Hello” “World” text is found below in spreadsheet “DDEWrite.xls”, on the sheet named “Sheet1”, in Row 1, columns 1 and 2. The rest of the code opens a DATA step, declares that filename ddewrite will be used for output, and writes the words “Hello” “World”.



Excel can also be closed at the beginning.

So far both SAS and Excel have been open for the examples used, or the data was moved into Excel after it was created. SAS, or more specifically a system command executed by SAS, can be used to open the Excel spreadsheet. SAS has at least two options that relate to how SAS deals with system commands. The options (NO)XWAIT and (NO)XSYNC (in Microsoft Windows environments) control the opening and closing of the MS DOS command windows and the execution of system commands.

```
OPTIONS NOXWAIT NOXSYNC;
```

```
X "C:\Program Files\Microsoft Office\Office\EXCEL.EXE";
```

The following simple routine will wait for the spreadsheet to open, other routines are available but this is simple and effective. The time can be adjusted depending upon the speed of the users system, and the location of Excel may need to be adjusted to fit the execution system.

```
DATA _NULL_;
rc = SLEEP(60);
RUN;
```

The object is to open Excel from SAS. Another item worthy of note is the DDE Triplet :

```
FILENAME ddecmds DDE "excel|system";
```

This triplet allows Excel commands to be sent directly to Excel for execution.

Now the meat of the paper (A non-trivial example)

The SAS Institute, "Technical Support Document #325 –The SAS System and DDE " describes a non-supported SAS macro that came from the SUGI 17 proceedings, page 117, entitled "Data Exchange Between the SAS System and Microsoft Excel". The macro moves data from a SAS dataset into an Excel spreadsheet. This paper will build upon that macro, and add a useful feature or two. Additions to the original are in **BOLD TYPE** and the code is reformatted to fit in two columns on the page, any spaces introduced by reform formatting should be removed by the user.

```

/*****
/* Copied from SAS Institute, "Technical
/* Support Document #325
/* SAS2EXCL.sas, example from SUGI 17
/* proceedings page 117
/* Will send data via dde from sas to excel
/* and retain column headings
/* this version has been modified slightly
/* to correct errors in original
/* This paper for WUSS 10 (2003) adds a
/* new feature or two.
/* sasin= changed sasuser to sashelp
/*****
/* Parameters
/* START=, indicates whether SAS should
/* start Excel session. Y or anything
/* else) Default is Y.
/* SASIN=, Source SAS dataset. A one or two
/* level name, already defined in SAS
/* session.
/* EXDIRIN=, Directory of target Excel file,
/* including the final slash. Not
/* needed if the target is SHEET1.
/* EXCELIN=, Target Excel file. Default is
/* SHEET1, the blank spreadsheet that
/* is opened when Excel is started.
/* EXCELOUT=, Saved Excel file, including
/* the complete directory path.
/* CLOSE=, Indicates whether SAS should
/* close the Excel session (Y or
/* anything else). Default is Y.

```

```

/* COLHEAD=, Indicates whether the
/* variable name or label should
/* appear in the Excel column
/* heading. (NAME or LABEL).
/* Default is NAME.
/* Where=, Code portal for processing
/* user code changes.
/* NOTE: If you choose to start
/* Excel you must not have any
/* previous DDE links in your
/* current SAS session.
/*****
options noxwait noxsync mprint;
%macro sas2excl (start=Y,
close=N,
where=,
colhead=LABEL,
sasin=sasuser.class,
exdirin=,
excelin=sheet1,
excelout=d:\excel4\sheet1 );
%*****;
%* check macro parameters
%;
%*****;
%if %upcase(&colhead) ne NAME and
%upcase(&colhead) ne LABEL %then %do;
data _null_;
put 'Parameter error: COLHEAD='
"&colhead";
abort;
run;
%end;
%*****;
%* start up Excel
%;
%*****;
%if &start eq Y %then %do;
x "&excelout";
data _null_;
x=sleep(2);
run;
%end;
%*****;
%* open file to contain Excel commands;
%*****;
filename cmdexcel dde 'excel|system';
%*****;
%* open existing Excel spreadsheet
%;
%*****;
%if %upcase(&excelin) ne SHEET1 %then
%do;
data _null_;
*file cmdexcel;
*put
"[open(%bquote (&exdirin&excelin)) ]";
*run;
%end;
%*****;
%* gather info about SAS dataset
%;
%*****;
proc contents data=&sasin noprint
out=conts(keep=nobs NAME label);
run;
data temp;
set conts end=eof;
&where
if label eq '' then
label=NAME;

```

```

call symput ('col' || left (_n_), trim (NAME));
call symput ('lab' || left (_n_), trim (LABEL));
if eof then do;
call symput ('columns', trim (left (_n_)));
call symput ('rows', trim (left (nobs)));
end;
run;
%*****;
%* Link to Excel spreadsheet to put column ;
%* heading in first row ;
%*****;
filename excel dde "excel|
&excelin.!r1c1:r1c&columns" notab;
%*****;
%* send column heading to spreadsheet ;
%*****;
data _null_;
file excel;
hextab='09'x;
%if %upcase (&colhead) eq NAME %then %do;
put %do i=1 %to &columns;
"%trim (&&col&i)" hextab
%end;
;
%end;
%else %if %upcase (&colhead) eq LABEL %then
%do;
put %do i=1 %to &columns;
"%trim (&&lab&i)" hextab
%end;
;
%end;
run;
%*****;
%* Link to Excel spreadsheet to send SAS ;
%* data beginning 2nd row ;
%*****;
filename excel dde
"excel|&excelin.!r2c1:R%eval (&rows+1)C&columns
." notab;
%*****;
%* now send data to Excel spreadsheet ;
%*****;
data _null_;
set &sasin;
hextab='09'x;
file excel;
put %do i=1 %to &columns;
&&col&i hextab
%end;
;
run;
%*****;
%* Save the Excel spreadsheet ;
%*****;
%if &excelout ne ' ' %then %do;
data _null_;
file cmdexcel;
put '[error(false)]';
put "[save.as(%bquote("&excelout"))]";
run;
%end;
%*****;
%* Close Excel ;
%*****;
%if &close = Y %then %do;
data _null_;

```

```

file cmdexcel;
put '[error(false)]';
put '[quit()]';
run;
%end;
%mend sas2excl;
%*****;
%* do it ;
%*****;
%sas2excl (start=Y,
close=N,
where=%nrstr (
where (indexw ('Inventory Product
Sales Stores',name)ne 0);
select (name);
when ('Inventory') order = 4;
when ('Product') order = 3;
when ('Sales') order = 1;
when ('Stores') order = 2;
otherwise;
end;
proc sort data=temp; by order;
data _null_;
set temp end=eof;
),
colhead=LABEL,
sasin=sashelp.shoes,
exdirin=c:\wuss_2003\my_wkbook.xls,
excelin=Sheet2,
excelout=c:\wuss_2003\my_wkbook.xls
);
run;

```

The parameters “start”, “close” and “colhead” were unchanged, while “sasin”, “exdirin”, “excelin”, and “excelout” were only updated to process the new file names. The new parameter “where” (set apart by blank lines) was added in such a way that it provides a window (portal) into the existing macro. The SAS macro instruction %nrstr (NO RE-SCAN STRING) allows the insertion of almost any character into a macro parameter. The code listed here selects four variables, assigns a sort order, and resorts the variable list before moving the data from SAS into an Excel spreadsheet. These minor changes made to this old macro allow the user to breathe new life back into use a basic piece of code. This will give the user flexible code module that can be included and re-used for many jobs by changing the way the macro is called.

PROCESS THREE (VISUAL BASIC, ONE STEP, FROM EXCEL TO SAS AND BACK)

Here is a simple introduction to building a routine in Excel that will send commands to SAS and build a routine that will retrieve the data back to Excel. This example begins with a saved Excel macro that is linked to an Excel Hot-Key. By using the “TOOLS”/”MACRO”/”RECORD NEW MACRO” in Excel the user can begin to record a macro

that can be used as a starting point for a tool that will do the job. Once the recording is turned on then import a file like Shoes.csv, and after that is complete then stop recording the macro. The following is an example of a macro to import data.

```
Sub Macro1()
' Macro1 Macro
' Macro recorded 7/31/2003
' by William E Benjamin Jr
'
' Keyboard Shortcut: Ctrl+z
Dim SAS As Object
Dim mean As Integer

Set SAS = CreateObject("SAS.Application")
SAS.Visible = True
SAS.Wait = True

SAS.Submit ("Proc Export Data=sashelp.shoes")
SAS.Submit
("OUTFILE='C:\wuss_2003\Shoes_1a.csv'")
SAS.Submit ("DBMS=CSV REPLACE;")
SAS.Submit ("Data _null_;")
SAS.Submit ("x=sleep(10);")
SAS.Submit ("run;")

Worksheets("Sheet1").Activate

If SAS.Visible = True Then
SAS.Quit
Set SAS = Nothing
End If

With ActiveSheet.QueryTables.Add _
(Connection:= _
"TEXT;C:\wuss_2003\shoes_1a.csv", _
Destination:=Range("A1"))
.Name = "shoes"
```

```
.FieldNames = True
.RowNumbers = False
.FillAdjacentFormulas = False
.PreserveFormatting = True
.RefreshOnFileOpen = False
.RefreshStyle = xlInsertDeleteCells
.SavePassword = False
.SaveData = True
.AdjustColumnWidth = True
.RefreshPeriod = 0
.TextFilePromptOnRefresh = False
.TextFilePlatform = xlWindows
.TextFileStartRow = 1
.TextFileParseType = xlDelimited
.TextFileTextQualifier = _
xlTextQualifierDoubleQuote
.TextFileConsecutiveDelimiter = _
False
.TextFileTabDelimiter = True
.TextFileSemicolonDelimiter = False
.TextFileCommaDelimiter = True
.TextFileSpaceDelimiter = False
.TextFileColumnDataTypes = _
Array(1, 1, 1, 1, 1, 1, 1)
.Refresh BackgroundQuery:=False
End With
End Sub
```

Simple Macro, Simple task, Complex concept.

This final example is like, most of the other examples, is a simple task, using a simple macro (from excel this time) that opens the door to a very large complex set of options. Space in this paper is limited and only allowed the author to display simple concepts related to the movement of data between SAS and Excel. Hopefully, these concepts will be useful enough to give the reader a sound foundation for the future.

The CONTENTS Procedure

```
Data Set Name: SASHELP.SHOES
Observations : 395
Member Type : DATA
Variables : 7
Label : Fictitious Shoe Company Data
File Name : C:\Program Files\SAS Institute\SAS\V8\core\sashelp\shoes.sas7bdat
```

-----Alphabetic List of Variables and Attributes-----

#	Variable	Type	Len	Pos	Format	Informat	Label
6	Inventory	Num	8	16	DOLLAR12.	DOLLAR12.	Total Inventory
2	Product	Char	14	57			
1	Region	Char	25	32			
7	Returns	Num	8	24	DOLLAR12.	DOLLAR12.	Total Returns

5	Sales	Num	8	8	DOLLAR12.	DOLLAR12.	Total Sales
4	Stores	Num	8	0			Number of Stores
3	Subsidiary	Char	12	71			

Output from the ODS system using CSV.

```
"Obs","Region","Product","Subsidiary","Stores","Sales","Inventory","Returns"
" 1","Africa","Boot","Addis Ababa","12"," $29,761"," $191,821"," $769"
" 2","Africa","Men's Casual","Addis Ababa"," 4"," $67,242"," $118,036"," $2,284"
" 3","Africa","Men's Dress","Addis Ababa"," 7"," $76,793"," $136,273"," $2,433"
```

- More data -

```
"393","Western Europe","Sport Shoe","Rome","14"," $9,969"," $74,848"," $549"
"394","Western Europe","Women's Casual","Rome"," 2"," $19,964"," $62,256"," $954"
"395","Western Europe","Women's Dress","Rome","16"," $106,676"," $389,861"," $3,160"
```

CONCLUSION

The movement of data between software packages has always been the bane of the programmer. Software companies need to protect their ability safeguard the data files and unique formats have been the vehicle of choice. The techniques displayed in this paper respect the unique formats of the vendors, and allow the users to move data between software packages. Consider these tips to be starting points of a great adventure into cross platform programming. Excel need not be restricted to reading *.CSV files, the called SAS program can actually place the data back into the Excel file while it is running using the other processes described in the paper.

REFERENCES

SAS Institute, "Technical Support Document #325 –The SAS System and DDE". <http://ftp.sas.com/techsup/download/tech-note/ts325.pdf>, updated 1999.

Vyverman, K. "Using Dynamic Data Exchange to Export Your SAS Data to MS Excel — Against All ODS, Part I". *Proceedings of the Twenty-Seventh Annual SAS Users Group*

International Conference, paper 5, 2002.

Vyverman, K. "Fancy MS Word Reports Made Easy: Harnessing the Power of Dynamic Data Exchange — Against All ODS, Part II —". *Proceedings of the Twenty-Eighth Annual SAS Users Group International Conference*, paper 16, 2003.

CONTACT INFORMATION

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

William E Benjamin Jr
 Work Phone: 623-492-4028
 E-mail Address: WmEBenjaminJr4@Juno.com

Sending Commands to Excel via DDE