Look Ma, No Hands! Or How We Move SAS® Into Microsoft Excel® With No Manual Intervention
John J. Cohen

ABSTRACT
No matter how prolific our SAS® processes or robust, detailed, and intricate our results, the seemingly-secondary task of providing output to our end users in some format other than standard SAS data sets and reports may undercut the perceived value of our otherwise brilliant work. So, porting results successfully to, say, the favorite end-user tool of choice in my environment – Microsoft Excel® – becomes critical to successful completion of a project.

In the circumstance of a one-off analysis, any manual cutting-and-pasting, formatting, and the like, is likely a critical part of one’s analysis anyway. However, if a project reaches a point of needing to be “productionized,” generalized across multiple sales geographies or therapeutic areas, updated on a monthly basis, or some other requirement, the likelihood of moving SAS results into Excel using a substantially manual process with any degree of timeliness and accuracy is probably unacceptably small. Further, part of the appeal of Excel as the deliverable is the ability to build additional functionality and intelligence into one’s spreadsheet.

We will discuss here techniques for populating Excel reports with the ultimate objective of never having to open up a workbook. Whether using SAS DDE (Dynamic Data Exchange), SAS ODS-to-XML, Proc Export into worksheet templates which are pre-formatted/containing Excel macros/VBA, or the latest, SAS stored processes with the Microsoft Office Add Ins, the goal of creating beautiful output “hands free” can be realized. These will be contrasted briefly with SAS DDE explored in considerable detail.

INTRODUCTION
Excel is the output format of choice for many of our customers. A widely-accepted tool for creating customized reports and graphics, embedding calculations and programming intelligence, through use of filter buttons and freeze panes, pivot tables and charts, this offers a powerful data repository for our end users, containing the end result of our data processing, metrics selection, and analytics achieved in SAS and the functionality for additional selection and manipulation of the data by the end user.

As a destination for one-off projects, then, Excel is a terrific choice. The effort invested in manually designing, formatting, adding functionality, and the like to our initial SAS output is usually well-received. The challenge of reproducing the same report updated the following month or for a different geography or brand is where further automation of some part of our process – of moving SAS output into Excel – becomes a critical component.

In Figure 1 below we show an example of a project where the requirement was to create a properly-formatted report for each member of the Sales Force who fit certain criteria. (Each sales representative had a minimum number of customers of interest to a particular promotional program.) The individual reports were to include an individualized summary page, to be aligned within our sales force structure, and each file was to be named with the appropriate territory code so that they could be loaded into our data portal. Further, individual territory reports were to be combined and summarized at each successive level of management (District, Region, and Business Center) along with appropriate summaries and file naming conventions. Only by automating the set of tasks involved could one reasonably contemplate taking on this project, which resulted in the creation of 2,611 individual reports. The data selection, number crunching, and analytics portion are easy for us SAS users – these are tasks we engage in every day. Taking the results and porting to Excel in an automated, hands-free fashion, in contrast, is likely a new experience.

In a similar fashion, for a smaller project we may have instead elected to populate a single spreadsheet with one report – in a separate tab – for each sales representative. Or brand. Or state. Or whatever the dimensions of interest to our particular question. The excitement comes from being able to format Excel directly from SAS –AND- the opportunity to therefore make ones output scalable, producing as many or as complex final form reports as required (within some finite set of constraints).
FIGURE 1 - Multiple Files in Identical Format (2,611 created in this particular run)

(With apologies to Dr. Seuss), would you, could you ever try to do this manually?

<table>
<thead>
<tr>
<th>Name</th>
<th>Size</th>
<th>Type</th>
<th>Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Store 753.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 754.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 755.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 756.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 757.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 758.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 759.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 760.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 761.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 762.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 763.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 764.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 765.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 766.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 767.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 768.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 769.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 770.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 771.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 772.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 773.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 774.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 775.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 776.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 777.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 778.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 779.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 780.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 781.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
<tr>
<td>Store 782.xls</td>
<td>14 KB</td>
<td>Microsoft Excel WorkBook</td>
<td>8/1/2009</td>
</tr>
</tbody>
</table>
SOLUTIONS
Clever folks at SAS and in the greater SAS community have come up with a variety of approaches to achieving the goal of automation of the process of moving SAS output into Excel and controlling the final output. We will briefly examine several of these.

Cutting-and-Pasting
The quick-and-dirty approach to moving data from SAS into Excel, it likely involves some combination of highlighting, then copying SAS output, pasting into Excel (Edit/paste special/html or unformatted text) followed by a formatting step (say, Data/text to columns) and then additional manual effort. In early development stages when figuring out what you want to do, what data make sense, and what steps are most effective to end up with a desired final result, this is likely a productive part of the development process. And for those who then turn on Tools/Macro/Record Macro, the first time manual steps can be turned into Excel macros (program statements) which can be applied in a somewhat systematic fashion to taking “raw” SAS output and turning it into formatted Excel.

Where this begins to have additional power is when the Excel workbook is, in fact, pre-formatted. The data are actually the “data table” – often then hidden – which are used to populate Pivot tables and Pivot charts. The steps which follow to update the spreadsheet likely include centering your mouse someplace inside the Pivot Table, then right click/Wizard/back to update cell references. (The new run might contain a slightly modified number of rows or columns if, for instance, you added another month of data or changed the number of customers displayed.) This is also likely accompanied by then hitting Next/layout/o.k. to update the column or row dimensions, change metrics displayed, and the like.

Where the power of the Pivot Table and/or Chart (or VBA programs) provide much of the additional formatting and end-user functionality, this is a reasonable approach. One can set up the data table and Pivot table updates to also be automated. However, if there are multiple spreadsheets or multiple tabs within a spreadsheet this gets both tedious and increasingly subject to error.

Proc Export
Proc Export allows one to move SAS data sets directly into Excel workbooks. A Wizard will handle the details and the resulting SAS program can be saved and used as a template for other export efforts. The result is a “raw” Excel file with formatting required afterwards. But with program control, the results are likely more systematic than with a cutting-and-pasting approach. And control of the location of the results, all the benefits of batch processing (can be run overnight, naming conventions, and the like) obtain.

Drawbacks include a whole new set of error messages to decipher and that final output deliverables to customers likely will require additional processing steps. Manual formatting, summarization tables, row and column headers, and the like are all after thoughts in this process. (Differences between SAS Versions 8 and 9 may also require a degree of coordination. Version 8 will not write to multiple tabs within the same worksheet, for instance.)

But as a solution to the creation of many spreadsheets in a controlled fashion it has much value.

SAS ODS-to-XML
One of the exciting new approaches to moving SAS output in Excel has been championed especially by Vince Del Gobbo (see reference to recent instance below). The benefit is that one can take SAS Proc Report, Proc Print, etc. results, output these as ODS XML formatted files, import these into Excel with virtually all (or maybe all!) of the required visual and logical formatting changes being indicated in our SAS programs, coded into the ODS XML statements, and then executed by Excel. We may find it prudent to import the XML into Excel and then Save As .xls files before distributing them to our end users. But the extent of manual effort otherwise may be minimized to an extraordinary extent.

Certain drawbacks need to be indicated, however. The success in importing of XML files relies NOT just on which version or how functional our Base SAS software, but also now on Microsoft Excel. An immediate limitation in this regard is that XML import requires Excel 2003 or more recent (which, surprisingly, remains a hurdle at many loca-
tions). Fully taking advantage of the functionality will also require becoming proficient in using ODS and likely the Microsoft Excel tagset. Finally, (and the reason many continue to use SAS/DDE as a solution) is that the ODS output largely assumes that output will be in organized rows and columns. Any formatting – easy to create manually in Excel, and part of the power of Excel as a data reporting tool – that is specific to particular cells out-of-synch, as it were, with the rest of the table, use of merged cells, and the like, are NOT supported and would require manual intervention to create. (Vince has suggested certain work-arounds, but these nevertheless require additional work and validation.)

SAS Microsoft Office Add Ins®

Another exciting approach takes advantage of certain additional software offerings from SAS, their Microsoft Office Add Ins. Typically referenced as part of the SAS BI® toolkit, these allow a one-time creation of a given Microsoft Office product document, spreadsheet, or presentation file (for MS/Word®, MS/Excel, or MS/PowerPoint®, respectively) from imported SAS output. These may then be updated on demand by the end user – to the end user appearing to be directly created within their Microsoft application – without any explicit invocation of SAS. (In fact, SAS is invoked in the background.)

For the SAS programmer the immediate requirement involves taking on a new set of SAS tools (most likely also including Enterprise Guide®), with considerable attention required regarding tool set synchronization, operating system architecture, internet connectivity, and the like in order to insure that end users will get the desired results. The end users will need certain additional software installed, their ability to work remotely/off line is constrained, and the elegance of handing off a discrete, stand-alone product is compromised.

*SAS/DDE – DYNAMIC DATA EXCHANGE*

Why We Like It

This approach is enormously popular in the SAS community – as any Google search will uncover – and has been written about at least since 1996. It allows the controlled movement of SAS data into Excel workbooks on a production basis, including control of file naming conventions, number of tabs, and content dynamically-driven by the contents of one’s incoming data. Considerable control may be exerted in the formatting of output and data is written one cell at-a-time, a set of rows by columns at a time, or across an entire worksheet. It can take advantage of an early version of Microsoft VBA and uses SAS PUT statements to write Excel macro statements to Excel and heavily-customize our results. Alternatively we can open a pre-created template and write data in a *vanilla* fashion into the existing format. Or a combination of these approaches may be employed. In any case, the value comes from minimizing the manual effort required to complete a project. And we only employ Base SAS.

In exchange we must take on a new tool set and syntax. The error messages are entirely delightful (i.e., as confusing as any other set we have learned to enjoy). And there are occasional instances where the desired outcome – even though seemingly correctly coded in our SAS program – is somehow NOT delivered. So manual touchups may still be required for more complex output. A follow up solution uses an approach suggested for the manual cut-and-paste approach above. If a standard manual/editing set of tasks is required across multiple tabs or a series of spreadsheets, a recorded Excel macro – in combination with the SAS/DDE program – could result in a (nearly?) fully-automated solution. And we must have SAS and Excel resident on the same machine.

Components of the Approach

Seeing is believing, as we hope to demonstrate at the accompanying presentation at the Conference. (We will ignore the conundrum regarding which comes first – does believing lead to seeing or . . .) But an annotated checklist of programming steps, components, and considerations are as follows (with selected accompanying sample program code in Appendix 1):

- **Open Excel** – we will be talking to another application on our computer – Excel – from SAS. (That is what Dynamic Data Exchange is all about.) Certain controls must be in place in order to insure that the com-
communication is received at the appropriate time (essentially so that it occurs as a well-synchronized dialogue).

- **Open a specific worksheet** – whether a blank worksheet, a pre-formatted template, a series of templates, or some combination of these.

- **Identify the specific location within the spreadsheet** – using a libname statement we will identify the cells individually or as collections (by row and column references) in which we will write data.

- **Writing SAS data** – taking our SAS data sets and porting the values via PUT statements into the referenced libnames.

- **Saving the Excel spreadsheet** – to the desired drive/directory/file name. If writing multiple spreadsheets, we then re-open the template (or blank worksheet) and repeat the process.

- **Whether to customize the format of cells, rows, and columns or to use a template** – certain considerations.

- **Controller Macro Variable** – how we determine how many times we repeat a process of opening, writing, then closing and saving new files, thereby creating multiple spreadsheets.

- **Differential formatting of parts of a spreadsheet** – this includes font size, special formats (e.g., currency), column width, font & background color, setting of Excel filter buttons and freeze panes, addition of column and row calculations, and the like.

- **Multiple tabs** – adding, deleting, and renaming.

- **Validation and de-bugging issues** – such as around using SAS Macro variables, tab name lengths and repeating of data-driven values, planning for enough rows and columns.

**CONCLUSIONS**

Whether we are SAS output snobs or multi-cultural Excel aficionados ourselves, for most of our user communities Excel is a preferred output format. Finding reasonable ways to move SAS results into Excel for regular or production efforts is critical to achieving results in a timely basis, whether during data preparation, process control, or validation efforts. The techniques outlined here all require an investment in learning a new tool set (and some combination of these may prove to be necessary). The hope is that once a set of techniques becomes sufficiently familiar to the programmer, one finds that the new tools are happily-employed to create desired output at an improved level of productivity.

**REFERENCES**


John Cohen and James Shields, “SAS/DDE: How to Make Our Customers Happy and Still Use SAS®”, NESUG 17, 2004, paper HW01, Baltimore, MD


Koen Vyverman, “Using Dynamic Data Exchange to Export Your SAS Data to MS Excel – Against All ODS, Part I,” SUGI 26, April, 2001, paper 190-27, Long Beach, CA.


ACKNOWLEDGMENTS
SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are registered trademarks or trademarks of their respective companies.

CONTACT INFORMATION
Your comments and questions are valued and encouraged. Please contact the author at:

  John Cohen
  Jcohen1265@aol.com

-----------------------------------------------
APPENDIX 1 - SAMPLE PROGRAM CODE – BUILDING A SPREADSHEET FROM SCRATCH

Open Excel – because our SAS application will be communicating with Excel

```sas
options symbolgen mprint noxsync noxwait;
filename cmds dde "excel\system";

%macro check;
  %let fid=%sysfunc(fopen(cmds,S));
  %if &fid = 0 %then %do;
    x "C:\Program Files\Microsoft Office\Office\EXCEL.EXE" /e';
    data _null_;
    x=sleep(5);
    run;
  %end;
%mend check;
```

Open a Specific Worksheet – here a partially pre-formatted template report

```sas
/*** Report Template pre-formatted with two header rows ***/
data _null_;  
  file cmds;  
  put "[open("C:\NESUG\Report Template.xls")]";
run;
```

Identify the Specific Location within the Worksheet Where We Want to Place our Data

```sas
/*** Start location in Row 3 (after first two header rows) ***/  
/*** for another 120 rows (120+2), columns 1-10, in sheet1 ***/
filename ddeout dde "excel\$sheet1!r3c1:r122c10" notab;
```

Writing SAS Data to Specified Location

```sas
/*** Filename ddeout specifies starting row 3, 10 columns ***/
data _null_;  
  set final_data_set;  
  file ddeout lrecl=3000; /** up to another 3,000 rows **/
  put Col_1 '09'x Col_2 '09'x Col_3 '09'x Col_4 '09'x Col_5 '09'x Col_6 '09'x Col_7 '09'x Col_8 '09'x Col_9 '09'x Col_10 '09'x;
run;
```
APPENDIX 1 - SAMPLE PROGRAM CODE (CONTINUED)

Format Instructions Within Spreadsheet – Set selected column widths, fonts, formats, and hide

```plaintext
data _null_;  
  file cmds;   /** activate sheet **/
  put '[workbook.activate("Sheet1")]';;

  /** specify column(s), set column width **/
  put '[select("r1c1:r122.c1")]';;
  put '[column.width(23)]';;

  /** specify cell(s), set font characteristics **/
  put '[select("r1c1:r2c10")]';;
  put '[font.properties("Arial","Bold",14)]';;

  /** specify cell(s), set special "picture" format **/
  put '[select("r3c8:r122c9")]';;
  put '[format.number("##.#0%")]';;

  /** specify column(s), "hide" column **/
  put '[select("r1c10:r122c10")]';;
```

More Format Instructions – You Can Do This?

```plaintext
data _null_;  
  file cmds;   /** activate sheet **/
  put '[workbook.activate("Sheet&i")]';;

  /** set Zoom to 85% (for entire spreadsheet) **/
  put '[zoom(85)]';;

  /** specify row(s)/column(s), set Freeze Pane **/
  put '[select("r2c1:r2c2")]';;
  put '[freeze.panes(true,1,3)]';;

  /** specify cell(s) set Filter **/
  put '[select("r3c1:r122.c10")]';;
  put '[filter()]';;
```

Save Spreadsheet – As “My New Spreadsheet.xls”

```plaintext
data _null_;  
  file cmds;   
  put "[save("C:\NESUG\My New Spreadsheet.xls")]";
run;
```