Paper SIB-105

Customized Excel Output Using the Excel Libname

Harry Droogendyk, Stratia Consulting Inc., Lynden, ON

ABSTRACT

The advent of the ODS ExcelXP tagset and its many features has afforded the SAS© user great flexibility in creating publishable Excel output. But what if the report must be delivered by populating pre-defined, customized Excel worksheets? DDE has been around for ages, and is a viable option, but often not a simple one to execute. This paper will demonstrate another method, the EXCEL libname engine. Using the EXCEL libname, SAS is able to write data into specific, pre-defined "named ranges" in customized Excel worksheets.

INTRODUCTION

There has always been ways to move SAS data to Excel, even if it was as simple as opening a SAS-created delimited text file in Excel. Since Excel quite happily opens HTML files, as enhancements were made to the SAS System, users began to utilize ODS HTML (or data step PUTs) to create customized output. When Excel XP/2003 added support for XML, an entirely new range of possibilities opened up with the creation of the ever-improving ExcelXP tagset.

Any of the methods already mentioned work well for displaying *tabular* data. Until the inclusion of the Excel libname in the SAS Access for PC Files software, if the Excel output required special or heavily customized formatting, the only choice was Dynamic Data Exchange, or DDE. While DDE is very powerful, it has a number of significant drawbacks. For anything beyond the simplest operations, DDE can be syntactically difficult, and worse, provides only rudimentary error handling and reporting when things go awry. In addition, Excel must actually be invoked on the machine, tying up the PC while the spreadsheet is manipulated / populated by DDE.

The SAS 9 Excel Libname statement allows the pushing and pulling of SAS data to/from Excel, ***even if Excel is not installed on the machine running SAS***. This paper will only focus on creating Excel output using the libname engine. See the References section at the end of the paper for additional information on pulling data from Excel into SAS.

The procedure for creating customized Excel Output using SAS's Excel libname engine:

1. Create a customized report layout in Excel
2. Define a range in Excel to receive the SAS data
3. Process the SAS data and massage it into the shape of the range
4. Use the Excel libname to load the Excel range and populate the report.

EXCEL LIBNAME

The Excel libname syntax is quite familiar:

```
libname xls <engine-name> physical-file-name;  eg.

libname xls excel 'c:\temp\report.xls';

    … SAS code …

libname xls clear;      * very important !! ;
```

If the physical file specified does not exist, SAS will create a workbook of that name. Once the libname statement is successfully executed, the Excel workbook and the sheets and named ranges defined in the workbook will be available in the SAS Explorer window.

The library can be treated like any other SAS libref with a couple of caveats:

- row and column limitations as defined by Excel, eg. 64K rows, 256 columns for Excel 2003
- Excel sheet and column names can contain characters not normally considered valid by SAS, use of options validvarname=any may be necessary when reading from Excel using the libname.

To reference Excel sheet names and named ranges, the familiar libref.dataset syntax is used (sheet names are suffixed with $, necessitating the use of SAS Name Literals syntax, ie. 'sheet1$'n when referencing).

eg:

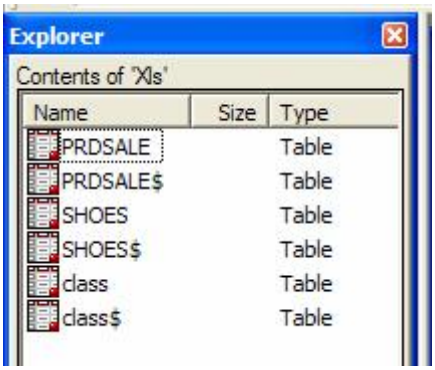
```
data xls.'sheet1$'n;        * referencing worksheet sheet1 ;

data xls.range_out;         * referencing named range range_out;
```

WRITING SAS DATA TO EXCEL

To populate Excel spreadsheets with the libname engine, most any DATA or PROC step can be used, eg.

```
data xls.class;
   set sashelp.class;
run;

proc copy in     = sashelp
          out    = xls;
   select prdsale shoes;
run;
```

After the DATA and PROC COPY steps are run, the SAS Explorer can be used to display the contents of the Excel library XLS. Since the Excel spreadsheet did not exist before running this code, SAS created the appropriate Excel "containers" to hold the data.



Why do we see our Excel table names twice, once as coded and once suffixed with a $ character?

SAS has actually created two Excel entities for each table:

- a *named range*, bearing the same name as the dataset
- a *sheet name* suffixed with $, that contains the named range
- subsequent references to the sheet must use xls.'class$'n syntax
- named range is referenced via xls.class

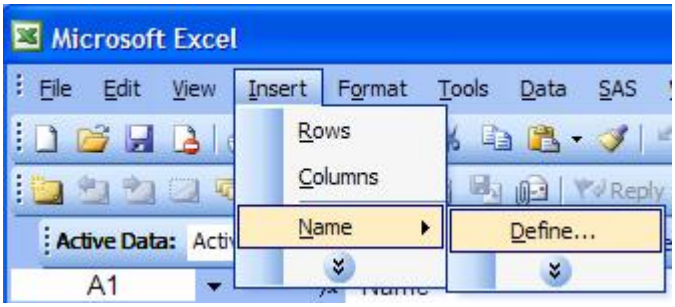
Why is the named range important? We'll begin to understand why when we clear the XLS libref and open the spreadsheet with Excel.

This doesn't look any different than the Excel output produced by PROC EXPORT !?

The sheet data starts in the **default, top, left-most cell, A1** and fills out the necessary columns and rows to contain the table data.

Let's dig a little deeper and examine named ranges…

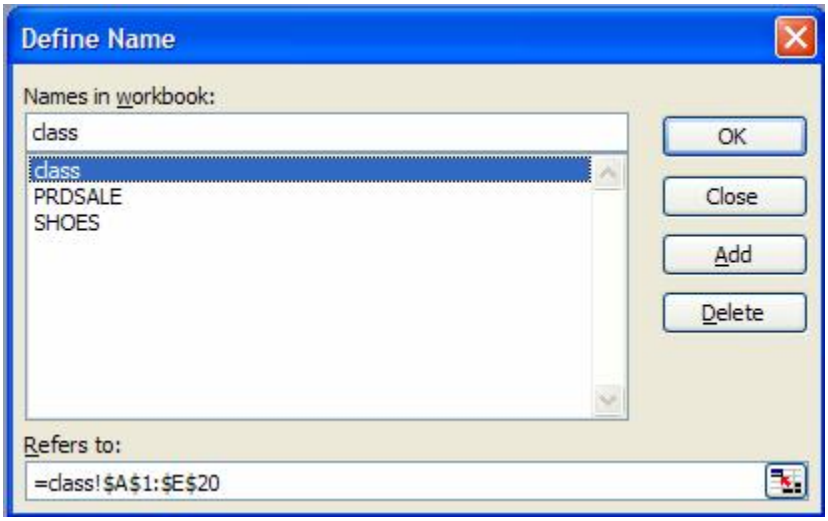
In Excel, we can look at the named ranges created by SAS. From the Excel menu bar, click Insert, Name and Define.



The three ranges defined by SAS are displayed.

eg. the **class** *named range* is defined as cells A1 - E20 in the **class** *sheet*.

Since named ranges are defined by declaring the top-left and bottom-right cells of the rectangular subset of sets, the usefulness of named ranges is beginning to emerge…



EXCEL LIBNAME LIMITATIONS

While the Excel libname is more powerful than PROC EXPORT, it does not have the full range of functionality provided by DDE.

Excel Libname Capabilities	Not Possible Via Excel Libname
• creating new workbooks • creating new sheets and named ranges • deleting existing data within a named range • populating an existing, empty named range • appending data to an existing named range • reading data from an existing sheet or named range	• formatting changes, eg. font, color • deleting an entire workbook, sheets or named ranges • deleting cells containing a formula • writing a formula into a cell

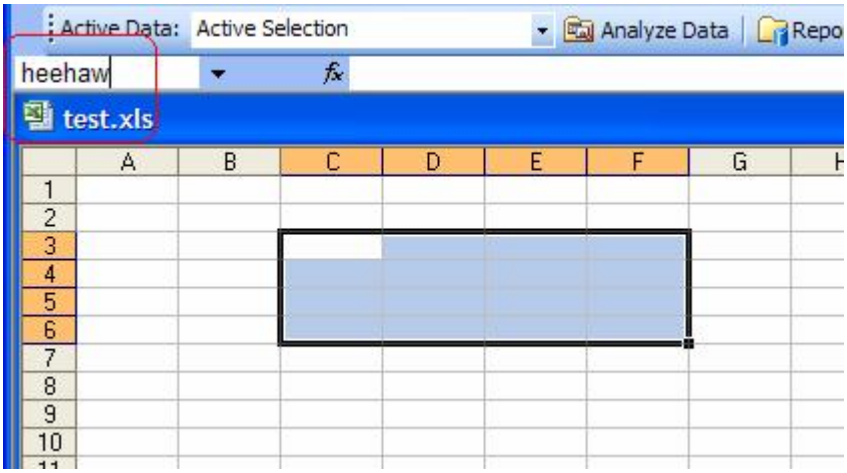
UTILIZING NAMED RANGES IN EXCEL

The advantages of the Excel libname begin to be realized when worksheets are created and formatted and named ranges are defined *in Excel* and the SAS Excel libname used to populate the formatted sheets. Separating the presentation of the data from the data itself allows the user to leverage the strengths of Excel's formatting ability to create very customized report "templates" and still take advantage of SAS's analytical power and processing options to create and load the data. Note that "template" is a generic term and is not referring to Excel's .xlt file type.

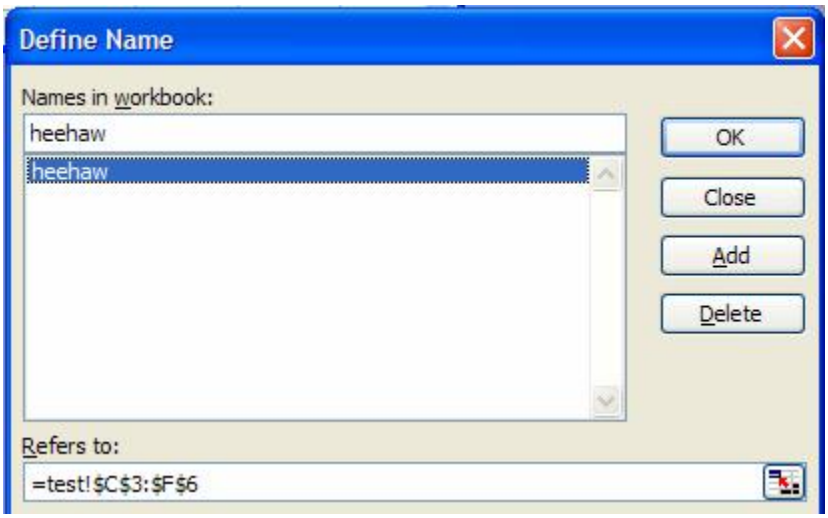
CREATING NAMED RANGES

There are two ways to create named ranges in Excel. The first utilizes the Excel menu bar choice that we've seen already (Insert, Name, Define) to provide a name for the range and the sheet and cell references that define the range. A simpler method follows:

Use the mouse to highlight the desired *rectangular* area and type the name of the range in the "Name Box" area and hit enter to define the range:



Once a range has been defined, the menu bar (Insert, Name, Define) method is the only way to delete it or change the size/shape of the rectangle of cells it includes. As we can see below, the named range "heehaw" has been defined as the highlighted group of cells specified above. Save and exit the Excel spreadsheet.

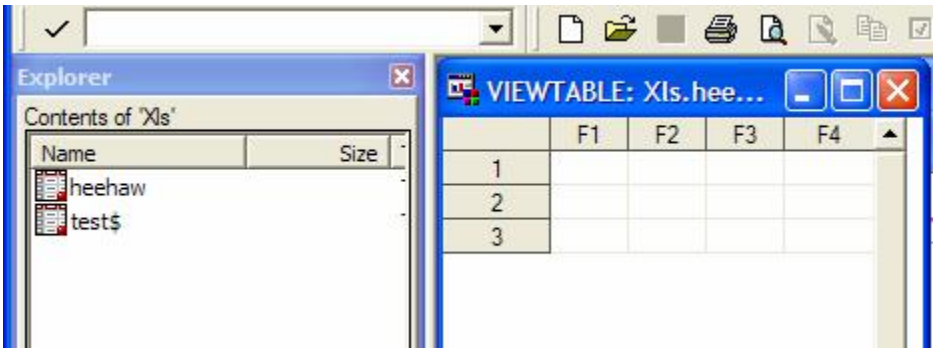


POPULATING NAMED RANGES

Once a named range has been defined, execute the Excel libname statement pointing to the spreadsheet file we just saved and exited.

```
libname xls excel 'c:\temp\test.xls' ver=2002;
```

When the contents of the XLS libname are viewed in SAS Explorer two "datasets" are present, a sheetname of "test$" and a range named "heehaw". When "heehaw" is opened, the dataset is found to contain four variables and **three** rows of data. Three rows of data…. hmmm… didn't the Excel range include rows 3-6, i.e. a total of four rows?



The Excel libname cannot replace the contents of a range. Even though we've done nothing but define the range in Excel, we can see in the SAS Explorer that rows and columns do exist. Before the range can be populated, it must first be deleted using PROC DATASETS or PROC SQL drop table:

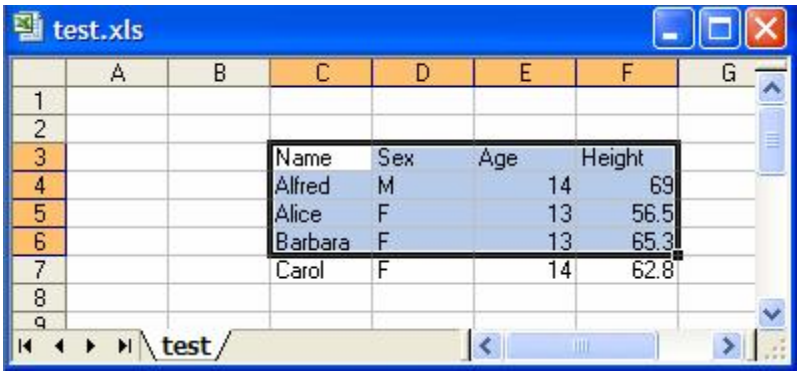
```
proc datasets lib = xls nolist;
    delete heehaw;
quit;
```

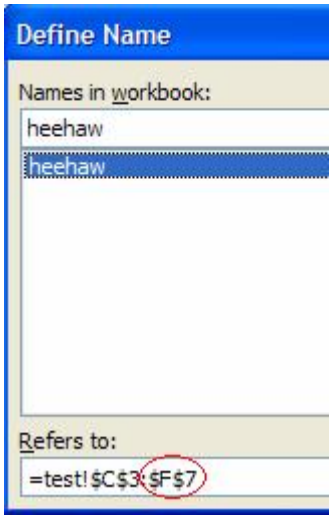
Once the "dataset" or range has been deleted, a simple data step will populate it. Once the range has been written, clear the Excel libname to complete the update.

```
data xls.heehaw;
    set sashelp.class ( obs  = 4
                        keep = name sex age height );
run;

libname xls clear;  * very important !! ;
```

When test.xls is opened, we can see that four observations have been written. However, because the header row containing the variable names is also written, the output has spilled over the bottom of the defined range. It's now apparent why the initial display of "heehaw" in SAS Explorer revealed only *three* rows; the first row of any range is reserved for the header record containing the variable names (or labels). Since the header row *will be generated automatically*, the definition of the Excel report template must take it into consideration. This also means that any named ranges we wish to populate via SAS must include at least two rows since the header row will always be present.

Note: the range has been expanded by SAS to include the extra row: was F6, now is F7.





DEALING WITH THE HEADER ROW

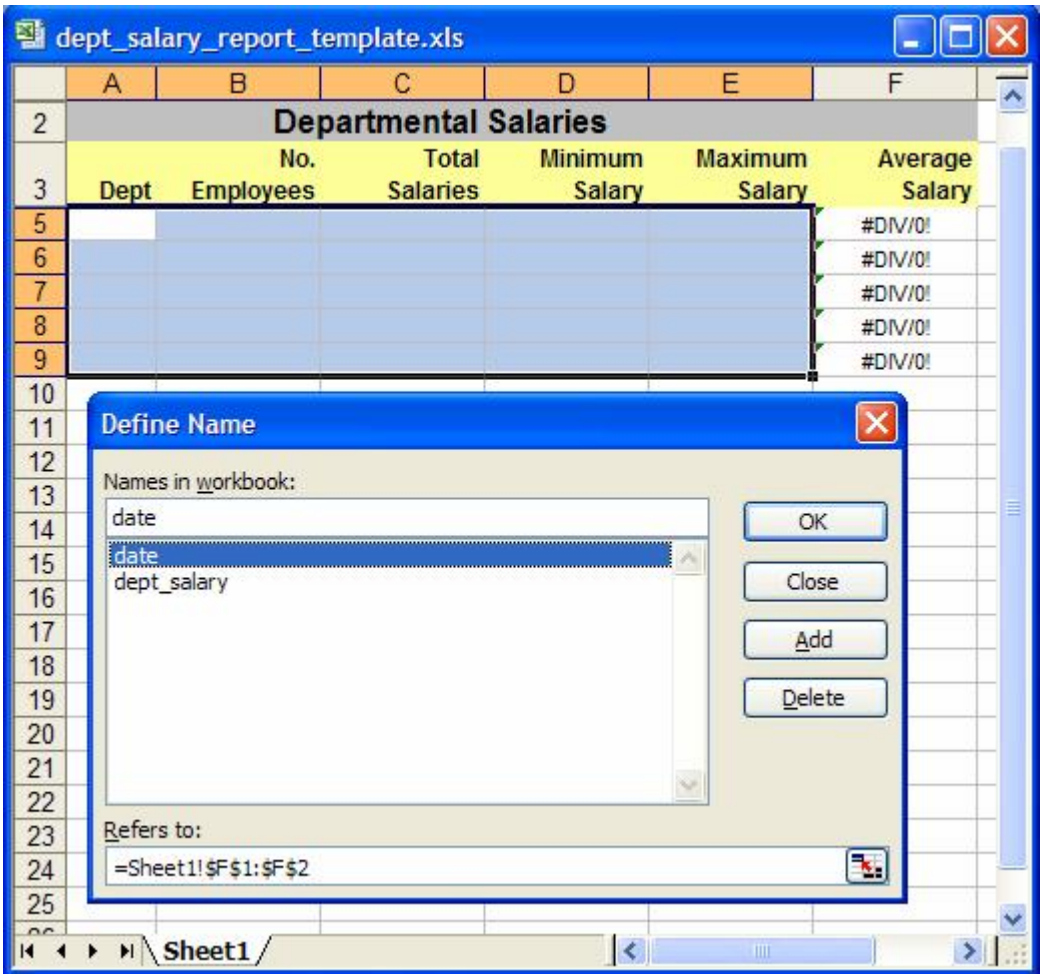
If the header row ought not to be part of the final Excel report, there are a couple of ways to deal with it. The easiest is to define the report template appropriately, accounting for the header row(s) and then hide the header row(s) in the template. A named range will happily include hidden rows. Note that specifying the DBLABEL=YES option will use SAS variable labels rather than variable names in the header row.

COMPREHENSIVE EXAMPLES - PUTTING IT ALL TOGETHER

CUSTOMIZED EXCEL REPORT

A summary query has created a SAS table containing employee counts, total, minimum and maximum salary metrics by department. A report template has been defined as pictured below.

Two ranges have been defined. The first, shown by the highlighted cells in the spreadsheet itself is called "dept_salary". It will be the destination of the summary table. In addition to surfacing the contents of the summary table, an additional column has been defined in the spreadsheet to calculate departmental average salaries (Total Salary / No. Employees). The second range, named "date", will record the report run date. Note the definition of "date", F1 - F2, i.e. the minimum two rows. Note as well, the 1st and 4th rows are hidden. Rows one and four will contain the SAS variable header information, rows we do not want displayed in our final, published report.



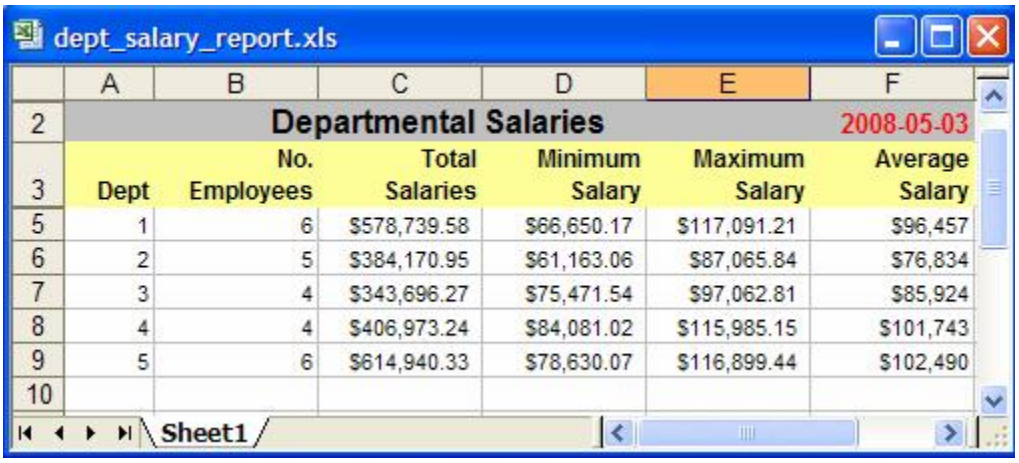
The code below will:

- copy the template file and create the final Excel report file, ie. preserving the template file
- define the Excel libname

- delete the two named ranges
- populate the two ranges
  - note the use of RETAIN in the second data step to ensure the PDV order is correct for the variables being output to Excel
- clear the Excel libname

```
options noxwait xsync;
x 'copy c:\temp\dept_salary_report_template.xls c:\temp\dept_salary_report.xls';

libname xls excel 'c:\temp\dept_salary_report.xls' ver=2002;

proc datasets lib = xls nolist;
      delete date dept_salary;   * case sensitive ;
quit;

data xls.date;
      format t yymmddd10.;
      t = today();
run;

* RETAIN used to ensure variables are in correct order for Excel report;
data xls.dept_salary;
      retain dept employees total_salary min_salary max_salary;
      set dept_salaries_sum;
run;

libname xls clear;
```

The report file looks exactly like the template that was used to define it. Report run date, the SAS departmental data and the calculated average salary column are all present and formatted as designed. While this example utilized only two named ranges, any number of rectangular ranges of varying sizes can be used when developing more complex reports.



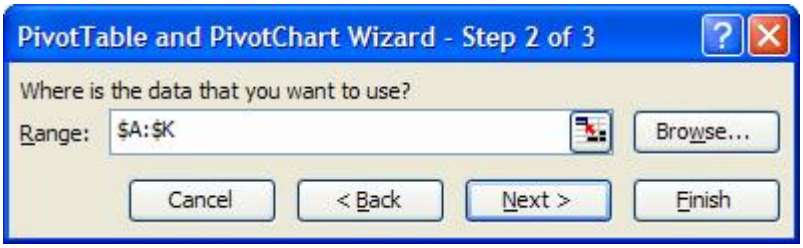
APPENDING PIVOT DATA

Report creation will not always start "from scratch". A monthly process may simply append the latest month's data to pre-existing Excel data from previous months. A pivot table or pivot chart sourced from the Excel data will be automatically refreshed with the additional data.

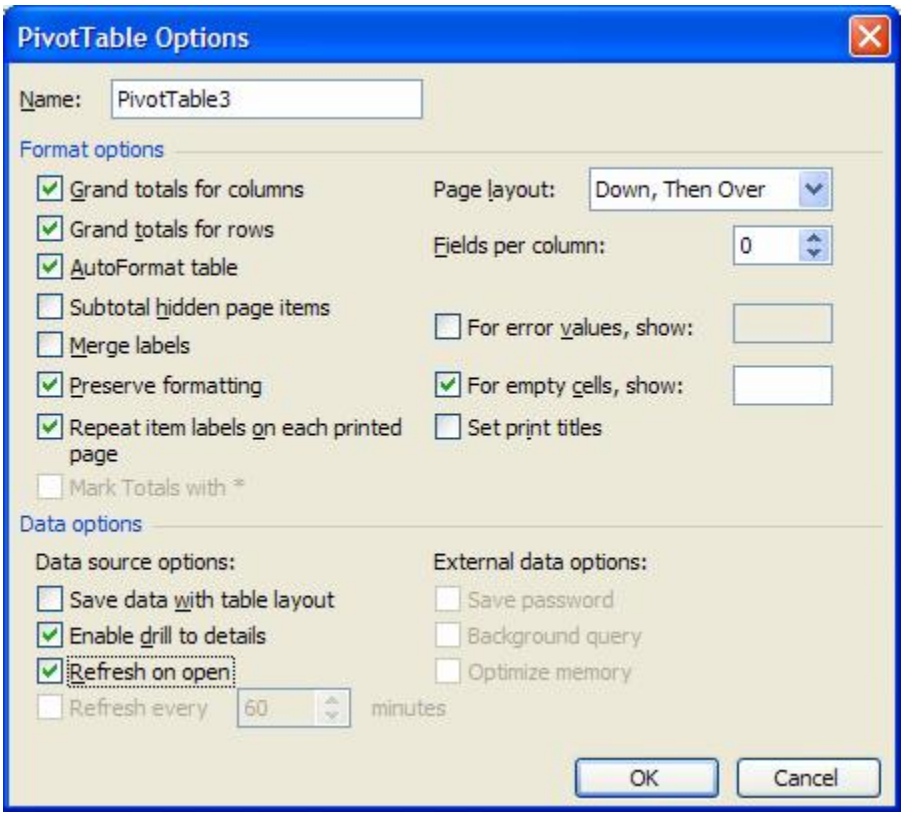
Create a new spreadsheet file with a range named "output" , defined as A1 : B2. The 2x2 cell range is certainly smaller than the data to be output, but the "heehaw" example on page 5 has demonstrated SAS's willingness to expand ranges to the size of the data actually sent to Excel. In the code below, the data from previous months is being loaded into the sheet. When the data step completes, the "output" range will be 10 columns wide by 1,381 rows deep.

```
libname xls excel 'c:\temp\append.xls' ver=2002;

proc datasets lib = xls nolist;
       delete output;
quit;

data xls.output;
       set sashelp.prdsale ( where = ( put(month,yymmn6.) ne '199412' ));
       mm = month(month);
run;

libname xls clear;
```

Open append.xls and define the pivot table/chart from the data that's available. Since the monthly process will append rows to the end of the current data, ensure that the pivot source does NOT specify a row specification, ie. only the columns containing the data, eg. $A:$K.



Using the Pivot Table Layout Wizard, define the page, row and column dimensions, metrics, formats etc… In addition, specify that the pivot table be automatically "Refresh on open" in the Options dialog:



Pivot Table after initial load:

append.xls

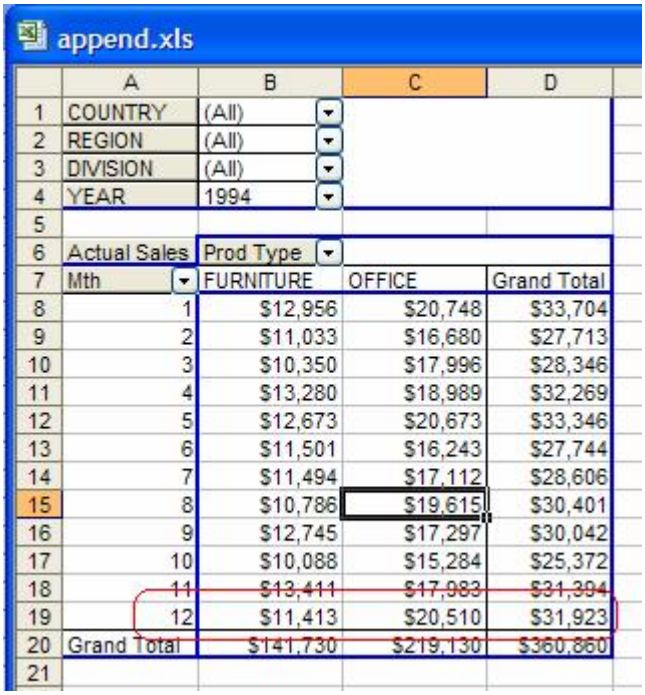
	A	B	C	D
1	COUNTRY	(All)		
2	REGION	(All)		
3	DIVISION	(All)		
4	YEAR	1994		
5				
6	Actual Sales	Prod Type		
7	Mth	FURNITURE	OFFICE	Grand Total
8	1	$12,956	$20,748	$33,704
9	2	$11,033	$16,680	$27,713
10	3	$10,350	$17,996	$28,346
11	4	$13,280	$18,989	$32,269
12	5	$12,673	$20,673	$33,346
13	6	$11,501	$16,243	$27,744
14	7	$11,494	$17,112	$28,606
15	8	$10,786	$19,615	$30,401
16	9	$12,745	$17,297	$30,042
17	10	$10,088	$15,284	$25,372
18	11	$13,411	$17,983	$31,394
19	Grand Total	$130,317	$198,620	$328,937
20				
21				

When appending data to a named range, one additional option, **scan_text = no**, must be specified in the Excel libname statement:

```
libname xls excel 'c:\temp\append.xls' scan_text = no ver=2002;

data this_month;
      set sashelp.prdsale ( where = ( put(month,yymmn6.) eq '199412' ));
      mm = month(month);
run;

proc append  base = xls.output
             data = this_month;
run;

libname xls clear;
```

After appending the 60 rows of data for 199412, the "output" range is now 1,441 rows deep.

When append.xls is opened and the pivot table refreshed, the additional 199412 data is present:



append.xls

	A	B	C	D
1	COUNTRY	(All)		
2	REGION	(All)		
3	DIVISION	(All)		
4	YEAR	1994		
5				
6	Actual Sales	Prod Type		
7	Mth	FURNITURE	OFFICE	Grand Total
8	1	$12,956	$20,748	$33,704
9	2	$11,033	$16,680	$27,713
10	3	$10,350	$17,996	$28,346
11	4	$13,280	$18,989	$32,269
12	5	$12,673	$20,673	$33,346
13	6	$11,501	$16,243	$27,744
14	7	$11,494	$17,112	$28,606
15	8	$10,786	$19,615	$30,401
16	9	$12,745	$17,297	$30,042
17	10	$10,088	$15,284	$25,372
18	11	$13,411	$17,983	$31,394
19	12	$11,413	$20,510	$31,923
20	Grand Total	$141,730	$219,130	$360,860
21				

ODDS AND SODS

A number of Excel engine libname options are available, eg. to display variable labels in the header row rather than variable names. See the Online Docs for those examples.

The Excel engine uses "Excel 97" as the default version. To ensure the latest functionality is available, specify the latest version available as a libname option. At the time of this writing: specify VER=2002.

Use of RETAIN *before* the SET statement in data steps used to populate Excel ranges is often required to order variables correctly.

CONCLUSION

The SAS Excel libname engine provides a versatile method of creating highly customized Excel reports even on machines where Excel is not installed. This functionality is even available on Unix if a PC File Server is available.

Since presentation customization (formatting, colors, fonts etc…) is performed in Excel, the user is not limited to the options available in ODS. Additionally, via creative use of multiple named ranges and appropriately massaged SAS data, it's possible to create reports that do not look rectangular.

REFERENCES

De-Mystifying the SAS® LIBNAME Engine in Microsoft Excel: A Practical Guide Paul A. Choate, Carol A. Martel , SUGI 31 Proceedings, **HTTP://WWW2.SAS.COM/PROCEEDINGS/SUGI31/024-31.PDF**.

Populating Custom Excel Spreadsheets: The Non-DDE Way, Winfried Jakob, TASS presentation, http://torsas.ca/downloads/Populating Custom Excel Spreadsheets (TASS 2007-09-21).pdf

SAS Online Docs - **HTTP://SUPPORT.SAS.COM/91DOC/GETDOC/ACPCREF.HLP/PCLIBNAME.HTM**

ACKNOWLEDGMENTS

My gratitude to Paul Choate, not only for the paper cited in the references, but also for his willingness to answer questions, provide additional examples of successful implementations and proof-reading this paper.

Additional thanks to Dianne Piaskoski for proof-reading this paper. I'll take credit for any errors that remain.

CONTACT INFORMATION

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

Harry Droogendyk
Stratia Consulting Inc.
PO Box 145
Lynden, ON, L0R 1T0
905-296-3595
conf@stratia.ca
www.stratia.ca

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 trademarks of their respective companies.