Paper 217-28

Filling Report Templates with the SAS® System and DDE

William C. Murphy
Howard M. Proskin & Associates, Inc., Rochester, NY

ABSTRACT

Delivering information to our clients is our only real goal. To achieve this goal, we generate data listings, create summaries, and report on statistical analyses. Most of this information is usually presented in tabular form. To accomplish this, we first design table templates in Microsoft Excel and then fill them with data from the SAS system. This combined approach gives us the power of the SAS system for data manipulation and management and the formatting capability of Microsoft Excel for presentation. For our clients it produces tables that are more attractive and allows them to 'play' with the data. To populate the tables, we use the dynamic data exchange (DDE) capabilities of the SAS language. Coupling DDE with macro programming, we are able to automate the process and readily revise tables if necessary. In the following, we will describe the table creation process from the template design to the data filling. In particular, we will detail a macro program that can readily be modified to fill almost any spreadsheet table, however convoluted.

INTRODUCTION

As a small statistical consulting firm in western New York, we analyze data supplied by laboratories and gathered in clinical studies on a variety of consumer and medical products. From these efforts, our clients expect detailed reports. A major part of these reports is summary tables and listings of the study data. The output of PROC TABULATE and PROC REPORT may produce most of the information that we need but it does not have the desired visual appearance. Furthermore, some of our clients occasionally want to 'play' with the data in the tables. Both of these problems can be solved by creating the tables in an Excel spreadsheet. Using the dynamic data exchange (DDE) abilities of the SAS system, we can readily transfer data from our SAS database into our spreadsheet tables.

QUICK DDE TUTORIAL

Dynamic data exchange (DDE) is a method of transferring data between Microsoft Windows programs. Implementation of this protocol between Excel and the SAS system is straightforward, being accomplished with FILENAME and PUT statements. First, we write the FILENAME linking us to Excel:

```
filename spread
     dde 'Excel|Sheet1! r8c1:r18c9' notab;
```

This creates a file reference called 'spread' for use by a DATA step. The file reference spread is linked by 'dde' (the third term on the line) to the program specified in the quotes. The first term in the quotes tells us that we are linking to 'Excel' and the term between the vertical bar and the exclamation mark indicates that the particular spreadsheet that we are linking to is 'Sheet1' (the name of the tab in Excel). The final term in the quotes is just a row and column specification. In our example, we are linking the file reference spread to the area of the spreadsheet with the upper right corner at row 8, column 1 and the lower left corner at row 18, column 9. It should be noted that to establish this link, the spreadsheet must be open at the time of execution of the SAS program. To write to this spreadsheet we would simply use a DATA step with PUT statements:

```
data _null_ ;
   set one;
   file spread;
   dlm='09'x;
   put  vara dlm +(-1)
        varb dlm +(-1) dlm +(-1)
        varc ;
   run;
```

The variable 'dlm' contains the hexadecimal code for a tab, and instructs the PUT statement to move the output to the next column. The +(-1) is a carriage control used to deal with an idiosyncrasy of the SAS language: a PUT statement inserts a space automatically between variables and the +(-1) does away with this space. This DATA step coupled with the FILENAME written above will insert the content of variable 'vara' into column 1, 'varb' into column2, and 'varc' into column 4. This is all there is for writing SAS data to and Excel spreadsheet. You can also use the SAS system and DDE to input the content of a spreadsheet and control the spreadsheet appearance, but that is beyond the scope of this paper. An excellent tutorial for this process was previously given by Vyverman (2001).

TABLE DESIGN

The first step in creating a report table is to come up with the actual layout that is desired in the final report. A mock-up of the actually table is usually created by our administrative staff and then sent to the client for approval. Where data should appear in the table, x's are inserted with the proper formatting. Our clients are very pleased with this process because it gives them a view of the actual report before it is written, and allows them to make changes upfront even before the actual data may be available. This report template also serves as a

perfect programmer guide: the data that the programmer needs is what fills the table!

For most requests, the templates created are simple matrices of numbers that can easily be filled with the SAS system. Only one FILENAME statement is used to refer to the spreadsheet area that the table data will occupy and a series of PUT statements in a DATA step will fill the table, such as the above program example illustrated. In fact, this whole process can be readily automated via a SAS macro program. The code for such a program would be nearly identical as that for dumping a SAS data set into a spreadsheet (Murphy, 1999). However, every so often we are given a template that has no particular resemblance to what is normally thought of as a table. It appears to be a random distribution of numbers with words interleaving. For some reason the client wants the information this way and the client is always right. For example, some clients actually like a demographic table like that illustrated by the template spreadsheet in Figure 1.

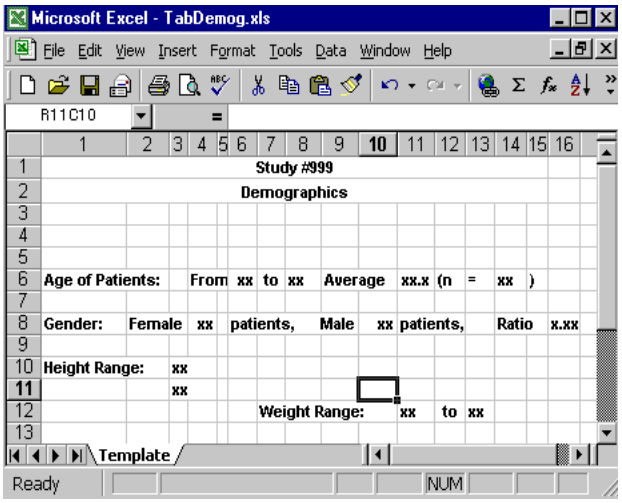

Figure 1. Sample spreadsheet template for demographic data.

To fill such a table template with data, you would have to specify a FILENAME for each cell containing information and write a separate DATA step to output the data there. On the other hand, you could automate the process with a SAS macro program!

DATA SET STRUCTURE

To construct an automation process, we will assume that we have a data set that contains two variables. The first variable will be called 'cell' and would contain a row and column specification. The second variable will be called 'value' and would contain the information that we wanted to write to the Excel spreadsheet row and column position specified by 'cell'. A sample of such a data set could be constructed with a DATALINES statement in a DATA step:

```
data one;
   infile datalines missover;
   input cell $ value ;
   datalines;
r6c6    18
r6c8    59
r6c11   31.1
r6c14   51
r8c4    27
r8c10   24
r8c16   1.12
r10c3   60
r11c3   75
r12c11  110
r12c13  280
   ;
   run;
```

Depending on our needs, this data set may be generated a little more automatically than using a DATALINES statement. For instance, the content of the desired variables could be directly assigned to the variable 'value'. A format could be created to link the desired variable name to a row and column position. A PUT statement could transfer this position information into the variable 'cell'.

AUTOMATION

Once we have the data set, we are ready to construct a macro to write our information to an Excel spreadsheet.

First, we need to know how many cells we are writing. This is the same as the number of observations in our data set. We can easily obtain this information using SAS Component Language (SCL) and %SYSFUNC. SCL is a group of SAS DATA step commands that allows you to access information about your data set. %SYSFUNC is a macro function that allows you to access commands that are not normally available to the macro processor. To get the number of cells we write

```
%let idData=%sysfunc(open(&data));
%let nCell=
    %sysfunc(attrn(&idData,NOBS));
```

The first line uses the SCL function OPEN to open the data set &data and assign it an identifier number that will be stored in &idData. The second line employs the SCL function ATTRN to determine the number of observations in the data set and stores that value in &nCell.

Once we have the number of cells to be filled, we then need the cell positions and values from the data set. We could obtain these variables by using a DATA _NULL_ step, followed by CALL SYMPUT to store the variable values into macro variables (Murphy, 2002). This would involve the creation of two macro variables for each cell to be written. Alternately, we could continue to use SCL and %SYSFUNC to read the variables we need directly from the data set into our macro.

To use the latter method, we must obtain the number that the SAS system assigns to the data set variables 'cell' and 'value':

```
%let numCell=
    %sysfunc(varnum(&idData,CELL));
%let numValue=
    %sysfunc(varnum(&idData,VALUE));
```

where we employ the SCL function VARNUM to find the variable number. Then we determine the variable type of 'value' (*i.e.* **C**haracter or **N**umeric):

```
%let typeValue=
    %sysfunc(vartype(&idData,&numValue));
```

Here we employ the SCL function VARTYPE. Next a macro %do loop is constructed that cycles through all of the cells to be filled. The first statement in the loop uses the SCL function FETCHOBS to load the appropriate data set observation, designated by the loop index &i:

```
%let rcObs=%sysfunc(fetchobs(&idData,&i));
```

Now we are ready to obtain the cell position and content from the variables 'cell' and 'value' and save the information into macro variables:

```
%let Cell=
    %sysfunc(getvarc(&idData,&numCell));
%if &typeValue=N %then
    %let Value=
    %sysfunc(getvarn(&idData,&numValue));
%else
    %let Value=
    %sysfunc(getvarc(&idData,&numValue));
```

where the first line uses the function GETVARC to read the value of variable number &numCell (*i.e.* the data set variable 'cell') and stores it in the macro variable &Cell. Next, using the previously determined type, &typeValue, we use the appropriate function, either GETVARN for numeric or GETVARC for character, to read the value of the variable number &numValue (*i.e.* the data set variable 'value') and store it in the macro variable &Value.

In the final section of our macro loop, the FILENAME statement links the SAS system to an Excel spreadsheet:

```
filename spread dde
          "excel|Template!&Cell " notab;
```

where the spreadsheet name 'Template' is our in-house standard name for the pre-designed spreadsheet tables. Double quotes are needed in order that the macro variable &Cell resolves to the proper value.

Finally, we write our data to the particular cell referenced in the FILENAME statement:

```
data _null_;
     file spread;
     put "&Value";
     stop;
     run;
```

where we use a DATA _NULL_ step since we do not wish to create an output data set. Therefore, within the loop, the program writes the content of &Value to the cell in the spreadsheet designated by &Cell. The next cycle of the loop performs the same function for the second observation in our data set. This keeps repeating until the final cell, is filled using the information in the last observation of our data set.

PUTTING IT ALL TOGETHER

Once this macro program is written, we can readily fill in the template given in Figure 1. We simply evoke the macro

```
%FillIt(data=one);
```

where we employ the sample data set one that was previously defined. The result is illustrated in Figure 2.

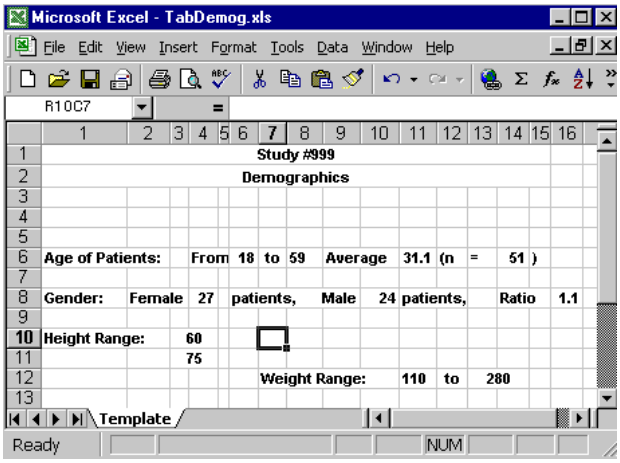

Figure 2. Spreadsheet template filled with macro program.

A complete listing of the macro described here can be found in the appendix.

CONCLUSION

Table templates designed in an Excel spreadsheet give a clear visual representation of the report to the client even before the study data is available. Once the programming is started, the templates provide a clear list of the needed numbers to the programmers. Filling these templates can easily be accomplished using DDE. Even unusual templates that involve data inserted in diverse spreadsheet cells can readily be filled using a simple macro routine.

APPENDIX

The following is a complete listing of the macro discussed in this paper:

```
%macro FillIt(data=);
```

```
%*** Declare Macro Variables Local ***;
%local idDate nCell numCell numValue i
   rcObs Cell Value;

%***** Open The Data Set *****;
%let idData=%sysfunc(open(&data));

%*** Number of Cells to be Filled ***;
%let nCell=
   %sysfunc(attrn(&idData,NOBS));

%***** Get the Variable Numbers for
   CELL and VALUE *****;
%let numCell=
   %sysfunc(varnum(&idData,CELL));
%let numValue=
   %sysfunc(varnum(&idData,VALUE));

%***** Is VALUE Numeric or Character?
   *****;
%let typeValue=
%sysfunc(vartype(&idData,&numValue));

%***** Loop Over All Cells to be
   Written *****;
%do i=1 %to &nCell;

   %***** Load the Selected
      Observation (Row) *****;
   %let rcObs=
      %sysfunc(fetchobs(&idData,&i));

   %***** Get the Value of CELL and
      VALUE from the Data Set *****;
   %let Cell=
%sysfunc(getvarc(&idData,&numCell));
   %if &typeValue=N %then
      %let Value=
%sysfunc(getvarn(&idData,&numValue));
   %else
      %let Value=
%sysfunc(getvarc(&idData,&numValue));

   %*** Setup DDE Link ***;
   filename spread dde
      "excel|Template!&Cell" notab;

   %*** Output to Excel ***;
   data _null_;
      file spread;
      put "&Value";
      stop;
      run;
   %end;

%***** Close Data Set *****;
%let idData=%sysfunc(close(&idData));

%*** CleanUp ***;
filename spread clear;

%mend;
```

REFERENCES

Murphy, W. C. (2000), "Everybody Wants Reports in a Spreadsheet: A SAS® Macro for Getting There ", *Proceedings of the Twenty-Fifth Annual SAS Users Group International Conference*, 25, 1163-1165.

Murphy, W. C. (2002), "Give Your Clients What They Want" Fill Report Tables with the SAS® System and DDE", *Proceedings of the Fifteenth Annual NorthEast SAS Users Group Conference*, 15, 715-717.

Vyverman, K. (2001), " Using Dynamic Data Exchange to Export Your SAS® Data to MS Excel: Against All ODS, Part I", *Proceedings of the Twenty-Sixth Annual SAS Users Group International Conference*, 26, Paper 11-26.

ACKNOWLEDGEMENTS

The author would like to thank Edith Hogan for her ever-inventive template designs, which keep the programmers busy.

TRADEMARKS

CONTACT

William C. Murphy
Howard M. Proskin & Associates, Inc.
2468 E. Henrietta Rd.
Rochester, NY 14623
Phone 585-359-2420
FAX 585-359-0465
Email wmurphy@hmproskin.com or wcmurphy@usa.net
Web www.hmproskin.com