Paper 229-2008

Using Dynamic Data Exchange (DDE) and Macro Variables to Create Custom Excel Workbooks with Multi-Sheets

Authors, Kathy Shepperson, Mathematica Policy Research, Inc., Princeton, NJ
Barbara Kolln, Mathematica Policy Research, Inc., Princeton, NJ
Shilpa Khambhati, Mathematica Policy Research, Inc., Princeton, NJ
Ronald Palanca, Mathematica Policy Research, Inc., Princeton, NJ

ABSTRACT

Dynamic Data Exchange (DDE) - why can't we seem to get away from it? If you are currently running SAS Version 9 and MSOffice 2002 or higher, then you can use XML in Base SAS 9.1 to create multi-sheet Microsoft Excel workbooks. If you are still using MSOffice 2000 or an earlier version, the easiest way to create multi-sheet workbooks is to use DDE. Although using SAS Proc Export is an option, DDE provides greater formatting and data placement control.

INTRODUCTION

Dynamic Data Exchange (DDE) allows you to exchange data between PC applications. DDE will not only read data from an Excel spreadsheet into an SAS data set but also place information from a SAS data set into an Excel spreadsheet. DDE provides more control over the placement of data and formats than is available with PROC EXPORT, including writing Excel commands to specific cells in a specified worksheet from the DATA step. DDE allows you to name the workbook and worksheets that are to receive the data from within SAS, and specify the data range through the upper-left to lower-right designated block of data. The worksheet cells can even be formatted with font, font size, and column width.

One thing to note here is that Excel must be running and the existing workbooks that you are reading from and writing to must be open. Once the data exchange is completed, the connection must be terminated.

In this paper, we will focus primarily on how to use DDE to export data from SAS to Excel. We will create an Excel workbook with custom-formatted reports on separate worksheets, using DDE and a single Excel template. This paper illustrates how to read the contents of a specified directory in SAS Version 9 and create macro variables using the names of each of the SAS data sets in the directory. The SAS program will use this information to generate an Excel workbook containing separate worksheets for each of the SAS files located in the directory.

The SAS program uses %SysFunc to identify the contents of the directory containing the input files, and cycles the file names through a macro loop. Using DDE, SAS populates the data from the input files into an existing file template (Shell.xls), generating a separate Excel file for each input file. These new Excel files have the same names as the input files. The individual Excel files are then combined into a master spreadsheet, with a worksheet for each Excel file. Finally, the individual Excel files are deleted using the X command, leaving the master spreadsheet as the only spreadsheet in the directory with all the data from the original input data sets.

Through our example, this paper will demonstrate various DDE features including the launching of an Excel application, opening an Excel file, populating data from SAS to Excel, moving data from various Excel files into a one master workbook, generating a separate sheet for each individual file, and renaming the worksheets and workbook.

For the purpose of this presentation, eight data sets are read in from our sample directory and an Excel workbook with eight worksheets is created.

STEP 1: ASSIGN MACRO VARIABLES AND LIB REF

First, initialize all macro variables related to the Excel file path, Excel file name, and output file path. Assign the library references to the data set input and output folders.

```
%Let Dir = C:\SGF2008\Posters;
%Let Table = States;
Libname in "&dir\Input";
Libname indd “&dir.\raw”;
Libname indd3 “&dir.”;
```

STEP 2: MACRO TO READ INPUT DATA SET NAMES INTO SEPARATE MACRO VARIABLES
This step reads all the input files names from the input directory and creates an individual macro variable for each of the file names. For our example, the input SAS data set names are by state, such as NJ.sas7bdat, CA.sas7bdat.

The macro GetNames(), generates a text file called AllFiles.txt, specified as a macro parameter outfile in our program. The Allfiles.txt contains the list of input files from the directory path specified as a macro parameter, filedir.

The Null data step generates macro variables called FileNm1 to FileNm&NumFiles. These Macro variables hold the name of the each input data set in the AllFiles.txt. The macro variable &NumFiels holds the number of input data sets in the AllFiles.txt.

```
%MACRO GetNames(filedir,outfile);

   %Let Fref=x;

   %Let rc=%sysfunc(filename(fref,&filedir));
   %If &rc ne 0 %Then %Put %sysfunc(sysmsg())

**Open the Directory with the input data sets;
   %Let Dirid=%sysfunc(dopen(&fref));
   %Let NewFile = newfile;

   %Let rc = %sysfunc(filename(newfile,&outfile));
   %If &rc ne  0 %then %put  %sysfunc(sysmsg());

**Open Allfiles.txt;
   %Let FileId = %sysfunc(fopen(&newfile,a));
   %If &FileId eq 0 %Then   %Put %sysfunc(sysmsg());

**Get number of files in the directory;
%Let NumFiles = %sysfunc(dnum(&dirid));

    %Do I = 1 %to &NumFiles;
      **Read the names of the files in the directory;
        %Let FileNm= %sysfunc(dread(&dirid,&I));

      **Put the name of each file to the file buffer;
        %Let rc = %sysfunc(fput(&fileid,&filenm));

     **Write the name to the file;
        %Let rc = %sysfunc(fwrite(&fileid));
        %If &rc ne 0 %Ten %Put %sysfunc(sysmsg());
   %End;

    **Close the directory and the file;
   %Let rc=%sysfunc(dclose(&dirid));
   %Let rc=%sysfunc(fclose(&fileid));
   %Let rc=%sysfunc(filename(fref));

%MEND GetNames ;

 %GetNames(&DIR.\Input\,AllFiles.txt);


**Generates Macro variables FileNm and NumFiles for the files in the Raw directory;
Data _NULL_;
 Infile  "&Dir.\AllFiles.txt" End=eof;
 Input  @001 varname :$50.;
 Length nVarName $50.;
**Remove  SAS data set extension,  .SAS7bdat from the SAS file name;
 nVarName=SUBSTR(STRIP(varname),1,(INDEXC(STRIP(varName),".")-1));

 **Set a counter for each file in the AllFiles.txt;
 c+1;
   **Get name of files and create  macro variable &FileNm;
   Call Sympute('filenm' || left(put(c,8.)),upcase(strip(nVarName)));
```

****Get total number of files and create macro variable &NumFiles.;**

```
      if eof then call symput('numFiles',left(put(c,8.)));
   Run;
```

****Once the Macro variables are generated, the AllFiles.txt is deleted;**

```
      OPTIONS  noxwait noxsync;
      X  Del  "AllFiles.txt";
```

STEP 3: DATA PREPARATION

The third step generates the output data set whose layout matches the Excel template. The Macro DoFreq counts the number of MONTHS by TYPE. The count of MONTH by TYPE is then transformed into a data set with one observation (TYPE) with multiple columns (MONTH). The macro's Do-Loop cycles through all the data sets listed in AllFiles.txt, and executes PROC UNIVARIATE for each file to create an output data set for each with the reported values. **The data set <State>.sas7bdat has the same data structure or layout as the Excel template "Shell.xls."

```
   %MACRO DoFreq;
      %Do I = 1 % to &NumFiles.;
          PROC SORT DATA= Indd.&&FileNm&I. OUT= Tmp;
          BY Month Type Release;

          PROC UNIVARIATE DATA = Tmp NOPRINT ;
           VAR  Release;
            BY  Month Type Release;
            OUTPUT OUT=Tmp2 n=N ;
          PROC SORT;
          BY  Type Release Month ;

          Data Tmp2;
           Set Tmp2;
            By  Type Release Month;
              drop Month Release  ;
           retain Month1 - Month6;
           If first.Release then
           Do;
              %Do M = 1 %to 6;
                      Month&M=0;
              %End;
           End;
            %Do M = 1 %To 6;
              If Month=&M Then Month&M.=n;
            %End;
            If last.Release then output;
            ID2 = Type;
```

****Add data set that has three blank rows to create blank rows between observations, to match the layout in the Excel shell;**

```
        Data &&filenm&i.;
          Set Tmp2 Indd3.Empty;
          ID = _N_;   ID2 = Type;
          If ID = 15 Then ID2 = 5.5;
          If ID = 16 Then ID2 = 11.5;
          If ID = 17 Then ID2 = 11.6;
        Run;
        PROC SORT; By ID2;
```

****Create permanent data set that gets written into the input directory;**

```
         Data In.&&filenm&i.;
           Set &&filenm&i.;
           Drop ID2 ID Release;
         Run;
       %End;
%MEND DoFreq;

%DoFreq;
```

STEP 4: LAUNCH EXCEL AND OPEN EXCEL FILES

For Excel to be used, the Excel Application needs to be invoked before any transfer of data can occur between SAS and Excel. The Macro OpenExcel launches Excel.

```
    %MACRO OpnExcel;
         Filename  sas2xl dde 'excel|system';
         Data _NULL_;
            Length fid rc start stop time 8;
            fid = FOPEN('sas2xl','s');
             If (fid le 0) then
              Do;
                 rc = system('start excel');
                 start = datetime();
                 stop = start + 10;

                  Do while (fid le 0);
                   fid  =  FOPEN( 'sas2xl', 's' );
                   time = DATETIME();
                   If (time ge stop) then fid = 1;
                  End;
              End;
             Rc=FCLOSE(fid);
           Run;
     %MEND OpnExcel;
```

The Macro FileOpen passes the name of the file being opened as a parameter. This Macro is used in the subsequent steps to open Excel Template and the new Excel files generated from the input SAS data files.

```
   %MACRO FileOpen(fname);
       Data _NULL_;
        File sas2xl;
       Put '[ERROR(false)]';
       Put '[OPEN("'"&dir"'\'"&fname."'")]';
     Run;
   %MEND FileOpen;
```

STEP 5 POPULATING THE EXCEL SHELL.XLS

This part of the program does the following steps:

1. Launch Excel, and open the Excel template, Shell.xls
2. Populate the individual state data from the SAS file into the worksheet, by looping through each fileNm Macro variable
3. Save and close the worksheet for each state in the designated output directory.

The columns are mapped from the SAS file to the Excel template. The file name assigned to each worksheet will be the corresponding state name. The steps are repeated automatically for each state by the macro Do loop, and Macro variables fileNm (the name of the files) and Numfiles (the Number of files).

The option NOXSYNC indicates that after executing any operating system command the control is returned to the SAS system. For example, after executing an X command you can return to your SAS session without closing the process spawned by the X command.

Similarly, upon completion of the system command, the NOXWAIT option tells SAS to automatically close the command prompt window, without having to type Exit. The NOXSYNC is usually submitted with the NOXWAIT option to speed submitting and returning to the SAS System session.

```
%MACRO InsertData;
  **Loop through each file;
  %Do I = 1  %to &NumFiles.;
    OPTIONS noxwait noxsync;
   **Create new Excel workbook, insert data, and save state files;
   **Launch Excel;
    %OpnExcel;
  **Open Template Shell.xls;
    %FileOpen(Shell);

   **Filename sas3xl points to the template from the cell at row 12 and column 2 to row 29 and column 8;
    Filename sas3xl dde 'excel|[Shell.xls]State!r12c2:r29c8' notab;

   **Map datafile columns to the template;
    Data _NULL_;
       Set in.&&FileNm&i.;
       File sas3xl;
       **The columns in the SAS file mapped to the template;
       PutN '09'x LTC0 '09'x '09'x LTC1 '09'x LTCTYPE1 '09'x LTCTYPE2 '09'x LTCTYPE3 ;
    Run;

    Data _NULL_;
      File sas2xl;
      **Activate sheet called "State" and select row 2, column 1 in Shell.xls;
      Put '[workbook.activate("State")]';
      put '[select("r2c1:r2c1")]';

      **Populate selected cell with the title;
      Put "State Utilization In 2001, By DDE Long-Term Care Use: &&FileNm&I." '0D'x;
      Put '[select("r2c1:r2c1")]';

      **Save updated Shell.xls as individual state Excel file;
      Put '[save.as("'"&dir"'\'"&&FILENM&i."'")]';
      Put '[file.close(false)]';
      Put '[quit()]';
    Run;
 %End;
%MEND InsertData;
%InsertData;
```

STEP 6: RENAME THE WORKSHEETS

The macro RenameSheets changes the generic name of the worksheet, currently “State,” to the individual state initials. This is accomplished by generating a Rename Excel Macro Code in a blank Excel sheet, and then calling this macro to rename the sheet.

```
%MACRO RenameSheets;
   %Do I  = 1 % to &NumFiles.;

     OPTIONS noxwait noxsync;
     **Launch Excel by calling openExcel Macro;
     %OpnExcel;

      **Open file by calling FileOpen Macro;
     %FileOpen(&&FileNm&i.);
```

We need a blank macro-sheet to be available in the current Excel workbook. This sheet will be named Macro1 by default; it will hold the Excel Macro code for renaming the sheet. We want this blank sheet to sit on top of all the other sheets in the workbook. This is important in what follows.

```
   Data _NULL_;
     File sas2xl;

     **The code below makes sure that the single sheet is selected;
     Put '[workbook.next()]';
     **Create the blank Macro1-sheet in front of the currently selected sheet;
     Put '[workbook.insert(3)]';

    ** Move the Macro1-sheet to the top of the workbook;
     Put '[workbook.move("macro1","'"&&FileNm&i...xls"'",1)]';
```

****Next we define a range in the first column of the macro-sheet, large enough to hold loads of Excel macro code. Here, it is 100 cells;**

```
 Filename xlmacro dde 'excel|macro1!r1c1:r100c1' notab lrecl = 200;
```

****The subsequent steps write (and run) an Excel macro in the Macro1-sheet that will rename the default worksheet 'State' to the desired name &&FilenNm$i. ;**

```
   Data _NULL_;
    File xlmacro;
    Put '=workbook.name("State","'"&&FileNm&i."'")';
    Put '=halt(true)';
    Put '!dde_flush';

    File sas2xl;
    Put '[run("macro1!r1c1")]';
    Put '[error(false)]';
  Run;
```

****Clear the Macro1-sheet;**

```
   Data  _NULL_;
    File sas2xl;
    Put '[error(false)]';
    Put '[workbook.delete("Macro1")]';
    Put '[save.as("'"&dir"'\'"&&FileNm&i."'")]';
    Put '[file.close(false)]';
    Put '[quit()]';
   Run;
  %End;
%MEND RenameSheets;
%RenameSheets;
```

STEP 7: COMPILE ALL INDIVIDUAL FILES INTO ONE WORKBOOK

The following program is used to automatically collect data from all the individual state Excel files and append into one Master Workbook, called State.xls.

The following steps are taken to create one master Excel file:

1. Create new blank Excel file, State.xls.
2. Open an individual state file.
3. Copy the data from the individual file to the master Excel spreadsheet.
4. Save the master Excel spreadsheet.

These steps are repeated with the Macro Do loop for every existing file of state data.

```
 OPTIONS noxwait noxsync;
```

****Launch Excel;**

```
 %opnExcel;
```

****Create new blank Excel file, States.xls, to insert all the individual state files;**

```
Data _NULL_;
 File sas2xl;
  Put '[error(false)]';
  Put '[new(1)]';
  Put '[save.as("'"&dir"'\'"&Table."'")]';
Run;

%MACRO InsertSheets;
 %Do I  = 1  % to &NumFiles.;
    Data  _NULL_;
     File sas2xl;
    **Open individual state Excel files, created in Step 6;
     Put '[open("'"&dir"'\'"&&FileNm&i."'")]';

    **Copy the data from individual state files to the workbook as separate sheets;
     Put '[workbook.move("'"&&FileNm&i."'","'"&Table..xls"'",1)]';
     Put '[save.as("'"&dir"'\'"&Table."'")]';
    Run;
 %End;
%MEND InsertSheets;
%InsertSheets;
```

STEP 8: FINAL CLEANUP

The Macro Removefiles deletes unwanted individual state files after the multi-sheet Excel file is generated.

```
  %MACRO RemoveFiles;
   %Do I = 1 % to &NUMFILES.;
      x del "&dir\&&FILENM&I….xls";
   %End;
%MEND RemoveFiles;
%RemoveFiles;
```

LIMITATIONS

The directory path where the input files are located cannot contain any other files, since all files in the directory are input to our program. The Getname Macro does not look for any pattern in the names, such as .txt, .csv, or .sas7bdat.

REFERENCES

Denslow, Russell, and Li,Yan. 2001. "Using DDE with Microsoft Excel and SAS to Collect Data from Hundreds of Users." Proceedings of the 26th Annual SAS User Group International Conference, Paper 29-26.

Qi, H. 2003. "A Practical Approach to Transferring Data from Microsoft Excel to SAS in Pharmaceutical Research." Proceedings of the 29th Annual SAS Users Group International Conference, Paper 082-29.

Vyverman, K. 2001. "Using Dynamic Data Exchange to Export Your SAS Data to MS Excel—Against All ODS, Part I. " Proceedings of the 26th Annual SAS Users Group International Conference, Paper 011-26.

Yindra, Chris. 1998. "%SYSFUNC—The Brave New Macro World." Proceedings of the 23rd Annual SAS User Group International Conference.

ABOUT THE AUTHORS

Kathy Shepperson (Senior Programmer, Mathematica Policy Research, Inc. [MPR], Princeton, New Jersey) has been working with MPR for 3 years and has been a SAS programmer for more than 10 years.

Barbara Kolln (Senior Systems Analyst, Mathematica Policy Research, Inc., Princeton, New Jersey) has been working with MPR for more than 20 years and supervises the SAS programmers in the Survey Information System division.

Shilpa Khambhati (Systems Analyst, Mathematica Policy Research, Inc., Princeton, New Jersey) has been working with MPR for more than 7 years and has been a SAS programmer for more than 10 years.

Ronald Palanca (Systems Analyst, Mathematica Policy Research, Inc., Princeton, New Jersey) has been working with MPR for more than 7 years and has been a SAS programmer for more than 10 years.