Paper AD01

Managing very large EXCEL files using the XLS engine

John H. Adams, Boehringer Ingelheim Pharmaceutical , Inc., Ridgefield, CT

ABSTRACT

The use of EXCEL spreadsheets is very common in SAS applications, especially in the pharmaceutical industry. EXCEL sheets are fairly easy to manipulate and easy to edit. Also, most users are relatively at ease with using their EXCEL skills.

It is quite common for SAS applications to use EXCEL spreadsheets to hold and maintain metadata in their operational model. Given the excellent user interface and user skill levels in EXCEL, it is an excellent candidate for this role.

This use of EXCEL spreadsheets does come with some limitations, however. Spreadsheets are limited to 256 columns and about 64K rows. For large applications, this can be a 'show-stopper'. However, there are ways to work around these limitations. By using these 'work-around' solutions in two SAS interface macros (to EXCEL) we wind up with a simple implementation.

There are, of course, several methods to handle the task of splitting the large data into multiple sheets and combining them, when needed. These multiple sheets could be saved as multiple files or as a workbook. A recent paper by this author at the 2008 PharmaSug conference discussed the solution of using multiple files in SAS 8.2.

With SAS 9.x and the new XLS engine, there is a much more elegant and transparent solution possible. This paper describes the solution of storing data in EXCEL sheets and retrieving data from EXCEL sheets no matter how large the data is.

INTRODUCTION

In essence, we're talking about handling large amounts of data (or metadata) in our SAS application and wanting to use EXCEL as the place holder for that data. As the user has a simple interface to view or edit this data IN EXCEL, it becomes a great way to affect the behavior of that SAS application.

When the data gets too large for EXCEL, we need to break it up into chunks or sheets. Each physical sheet would represent one piece of a large 'virtual' sheet. This 'virtual' sheet would be stored as a workbook. We also need to keep track, however, how to put these sheets back together. All of this activity of splitting and re-assembling should be automatic and transparent to the user. Whenever the user needs to read from or write to an EXCEL spreadsheet, he or she uses the 'READXLS' or 'WRITEXLS' interface macros. The user does not have to be concerned whether this would involve a 'multi-sheet' operation or not.

The 'WRITEXLS' macro automatically determines whether the SAS dataset it is supposed to write to EXCEL needs to be split vertically (columns), horizontally (rows) or both. It then would create the separate physical sheets, if needed, and the metadata for putting them back together.

The 'READXLS' macro, on the other hand, determines if the EXCEL sheet it is supposed to load is a 'multi-sheet workbook with accompanying metadata. If so, it reads the metadata and uses that information to re-assemble all sheets in the appropriate order.

This paper describes the methodology and the coding of the Excel interface macros

1. THE APPLICATION

Any application program that uses Excel spreadsheets to store data or meta data should not have to worry about the underlying limitations of Excel or program around these. By using two Excel interface macros that handle the actual storage and retrieval of the data to and from Excel, the main application program can focus on higher level tasks.

This paper describes a methodology for automatically breaking up the SAS dataset into smaller pieces (sheets) to store them in a workbook - if needed. It also describes the methodology for automatically stitching these sheets together again when importing the data back to SAS.

The concept behind this methodology is that of a 'virtual' spreadsheet To the application program it looks like one Spreadsheet. Under the covers, however, it is really an Excel workbook, containing a one or more sheets.

The following map shows the physical sheets (and their names) in a sample 'virtual' spreadsheet that could contain up to 1014 columns (4x256) and 192k rows (3x64k):

Vertical column set sheets →

Horizontal Row set sheets ↓

VS_1_1	VS_1_2	VS_1_3	VS_1_4
VS_2_1	VS_2_2	VS_2_3	VS_2_4
VS_3_1	VS_3_2	VS_3_3	VS_3_4

Of course, we also need to keep some metadata for the workbook, i.e. how many sheets are stored, the key variables used for merging, etc. Without this information we wouldn't know how to properly re-assemble the dataset during the import to SAS. The WRITEXLS macro stores this metadata in an additional sheet that the READXLS macro looks for.

In previous versions of SAS it was possible to import from individual sheets in an Excel workbook type file, but not write to individual sheets. The new XLS (Excel) engine in SAS 9 not only makes it possible but also very easy to deal with the individual sheets in a workbook, just as if they were SAS datasets. That means that we can use functions that deal with datasets on individual sheets, i.e. exist, delete, etc. The one consideration to keep in mind when using the XLS engine is that it cannot replace existing sheets, so we must first check to see if a sheet exists and delete it if it does.

The Excel interface macros use an internal dynamic sheet naming convention: **VS_*R*_*C***, where VS is the common pre-fix, *R*=the row set number and *C*=the column set number. One or more digits could be used for *R* and *C* by these macros, depending on the amount of splitting needed. This means that the size of the dataset set to be exported to Excel is only limited by Excel's workbook specifications and space / performance considerations for splitting the data into multiple data sheets

To re-assemble the sheets in the proper order, the READXLS macro (in the above sample) would first merge the data from all sheets in the row set, e.g. Row_1_Dataset = VS_1_1← → VS_1_4. In order to be able to merge these sheets from the row set, each sheet must contain the same key variables. If there are multiple row sets in the 'virtual' sheet layout, the merged Row datasets would be appended in ascending row set number order.

2. THE MACRO INTERFACE

Following is the simple interface for the writeXLS macro with default values shown for some arguments.

```
%macro writexls (dsname=, IDvars=, libref=work,file=,
                 maxrows=60000,maxcols=250,
                 path=c:\windows\temp,workref=WORK, debug=N) ;
```

Only the Dsname, Idvars, File and Path arguments are required for the writeXLS macro..

Following is the simple interface for the readXLS macro with default values shown for some arguments.

```
%macro readxls(file=, sheet=SHEET1,path=,libref=work,dsname=,debug=N);
```

Only the Dsname, File and Path arguments are required for the readXLS macro.If no 'Virtual' Metadata sheet is found in the workbook and a sheet name is specified with the 'sheet' argument, that spreadsheet will be imported

4. THE MACRO CODE

3.1 The WriteXLS macro

The following code is functional and well documented to decribe what is being done in each section. As is the case in most macros, the amount of code of doing the actual desired work, i.e. saving the individual sheets in Excel, is quite small compared to the rest of the supporting code, i.e. error checking, setting up control variables, housecleaning, etc.

```
/*===================================================================|
|  Macro Program :            WRITEXLS                               |
|              by:            John Adams     11 February 2009        |
|-------------------------------------------------------------------|
|  Description : Create an EXCEL spreadsheet from a SAS dataset      |
|                ( Using the XLS engine )                           |
|  Arguments   : dsname  --> dataset name                  (REQ)     |
|                IDvars  --> id variables                  (REQ)     |
|                libref  --> lib name for d/s (def=work)   (OPT)     |
|                file    --> xls filename     (def= d/s name)(OPT)   |
|                            (filename w/o .xls extension)          |
|                path    --> path for xls file (w/o quoits)  (OPT)   |
|                            (def=c\windows\temp)                   |
|                maxrows  --> maximum # rows per sheet       (OPT)   |
|                            (def=63900)                            |
|                maxcols  --> maximum # columns per sheet    (OPT)   |
|                            (def=250)                              |
|  Output      : An EXCEL spreadsheet containing contents of d/s     |
|  Note        : The Excel sheet will be saved as Excel 2002 workbook|
|                file type.                                         |
|  Example     : %writexls(dsname=test,file=MYDATA,path=c:\myfolder);|
|===================================================================*/
```

```
%macro writexls (dsname=, IDvars=,
libref=work,file=,maxrows=60000,maxcols=250,path=c:\windows\temp,workref=WORK, debug=N)
         /des='sub-macro for writing a EXCEL sheet(V1.2)';
     %local file2 totcolcnt idcolcnt setcolcnt rowsetcnt colsetcnt pagecnt allvarcnt dommapcnt
                  allvar lst dommap lst domvar lst prefix lst idvar lst1 idvar_lst2
                  setvar_lst1 setvar_lst2 setvar_lst3 setvar_lst4 setvar_lst5;

   %if %length(|&dsname)=1  %then                                    /* Error checking */
       %put %str(ERR)OR: You must specify a dataset name;
   %else %if %sysfunc(exist(&libref..&dsname))^=1  %then
       %put %str(ERR)OR: Dataset &libref..&dsname was NOT found ;
   %else %if %sysfunc(fileexist(&path))^=1  %then
       %put %str(ERR)OR:Path &path does NOT exist ;
   %else %if %length(|&IDvars)=1  %then
       %put %str(ERR)OR: You must specify ID variable name(s);
```

```
%else %if &maxcols > 255  %then
    %put %str(ERR)OR: You cannot have more than 255 columns;
%else %if &maxrows > 63999  %then
    %put %str(ERR)OR: You cannot have more than 63999 rows;

%else %do;                                              /* No errors - process data block*/
     %let libref =%upcase(&libref);
     %let dsname =%upcase(&dsname);

      %let idvar lst1=%upcase("%scan(&IDvars,1,' ')");
      %let idvar lst2=%upcase(%scan(&IDvars,1,' '));
      %let i=2; %let wd1=%scan(&IDvars,&i,' ');

   %do %while(&wd1 ne );                              /* make a quoted commaseparated IDvar_list*/
      %let idvar lst1 =&idvar lst1 , %upcase("&wd1");
      %let idvar lst2 =&idvar_lst2 , %upcase(&wd1);
      %let i=%eval(&i+1);
      %let wd1=%upcase(%scan(&IDvars,&i,' '));
   %end;

  %if %length(|&file)=1 %then %let file=&dsname;            /* default filename assignment */

 %let totcolcnt=0;  %let idcolcnt=0;     %let _pagecnt=0;

  data   colinf1(keep=rowno name vartyp) ;  /*create variable name list and assign var type*/
    set sashelp.vcolumn;
      where upcase(libname)="&libref" and upcase(memname)="&dsname"  ;
      rowno =  n ;
    if  upcase(name)  in(&idvar lst1)  then vartyp = 0;
    else                                   vartyp = 1;
  run;

  proc sort data= _colinf1 ; by vartyp; run;

  proc sql noprint;                                    /* characterize the dataset to export */
     select nobs,nvar into:  rowcnt,                      /*count of obs and vars in dataset*/
                          :  varcnt
         from sashelp.vtable where upcase(libname)="&libref" and upcase(memname)="&dsname"
;

     select count(name) into: totcolcnt from  colinf1  ;                /*count all columns */
     select count(name) into: idcolcnt  from  colinf1 where vartyp=0;     /*count ID columns*/
     select count(name) into: othcolcnt from _colinf1 where vartyp=1; /*count non-ID columns*/

     %let colsetcnt= &totcolcnt / &maxcols;

   create table  colinf2 as
     select name, vartyp, &maxcols as maxcols,
                 case                                     /*create vertical column set number*/
                 when vartyp = 0 then 0
                 else ceil((&idcolcnt + (vartyp * rowno)) / &maxcols)
                 end as page_num
        from _colinf1  ;

     select max(page_num)    into: _pagecnt from _colinf2;      /* # of vert sheets  needed */

     %let rowsetcnt= %sysfunc(ceil(&_rowcnt/&maxrows));       /* # of  horiz. sheets needed */

   %do i=1 %to & pagecnt;                     /*make non-id varname list for each column set */
     select name into: setvar_lst&i separated by ' , ' from _colinf2 where page_num=&i;
   %end;
  quit;

    %let _rowno  = 1;

    %let file2  =&path.\&file..xls;                                  /* Work Book name name*/
    libname xlsdata "&file2"  version=2002;                           /*open the workbook */


  %if & pagecnt > 1 or &rowsetcnt >1 %then %do;           /* show log message for splitting */
      %put +---------------------------------------------------------------------------+;
      %put | SPECIAL NOTE : The dataset has too many columns to fit in one sheet !    |
      %put |                It will be split into several sheets within a workbook. |;
      %put +---------------------------------------------------------------------------+;
      %put;
  %end;
```

```
    %do i=1 %to &_pagecnt;                          /*Start of Loop through vert. column sets*/

         proc sql noprint;                        /*create a temp dataset with only selected cols*/
          create table tempdat
            as select &idvar_lst2, &&&setvar_lst&i from &libref..&dsname;
         quit;


       %do j=1 %to &rowsetcnt;                      /*Start of Loop through horiz. row sets*/

           %let sheetnam = %str(VS_)%trim(%left(&j))%str(_)%trim(%left(&i));  /* sheet name*/

           %let startobs = %eval(1+(&j-1)*&maxrows) ;            /* for horizontal splitting*/
           %let endobs   = %sysfunc(min(&_rowcnt, &startobs -1 + &maxrows));

         %if %sysfunc(exist(xlsdata.&sheetnam))=1  %then %do;  /* delete sheet if it exists*/
           proc datasets library=xlsdata nolist;  delete %trim(&sheetnam) ;  quit;
                                                  /*Note: XLS engine cant replace sheet*/
         %end;

         data xlsdata.%trim(&sheetnam) ;     /*create a Excel data sheet with selected rows*/
           set tempdat(firstobs=&startobs obs=&endobs);
         run;

       %end;                                        /*End of Loop through horiz. row sets*/

    %end;                                          /*End of Loop through vert. column sets*/

    %if %sysfunc(exist(xlsdata.meta dat))=1  %then %do;        /* delete sheet if it exists*/
         proc datasets library=xlsdata nolist;  delete meta_dat ;  quit;
     %end;

     data xlsdata.meta dat;            /*save the metadata in a new sheet called 'meta_dat'*/
       length  idvars  $5000;
         Nhsheets=&rowsetcnt;  Nvsheet=&_pagecnt ;   idvars= "&idvar_lst2";
       output;
     run;

    %if %upcase(&debug) NE Y and %upcase(&debug) NE YES %then %do;         /*housecleaning*/
       proc datasets library=work nolist;  delete _colinf1 _colinf2 tempdat ;  quit;
       libname xlsdata  ;
    %end;
  %end;

%mend writexls;
```

3.2 The ReadXLS macro

```
  /*===================================================================|
   |  Macro Program :            READXLS                               |
   |              by:            John Adams              11 February 2009 |
   |-------------------------------------------------------------------|
   |  Description : Create a SAS dataset or view from an Excel spreadsheet |
   |                ( No ODBC connection required )                    |
   |  Arguments   : file    --> xls filename (w/o .xls extension)   (REQ) |
   |                sheet   --> Sheet name/number (eg. Sheet1)      (OPT) |
   |                path    --> path to file (w/o quotes)           (REQ) |
   |                libref  --> lib name for storing d/s            (OPT) |
   |  Output      : A SAS dataset                                      |
   |  Note        : The Excel sheet must be saved as Excel 2002 workbook |
   |                file type.                                         |
   |  Example     : %readxls(file=MYDATA,path=c:\myfolder,dsname=test);  |
   |===================================================================*/

%macro readxls(file=, sheet=SHEET1,path=,libref=work,dsname=,debug=N)
     /des='sub-macro for reading a EXCEL sheet (V1.1)';
     %local  name2 file2  setcolcnt rowsetcnt _pagecnt
       idvar lst1 idvar lst2  filemode  ;
     %let filemode=SINGLE;
     %let file    =%left(&file);
     %let libref  =%upcase(&libref);
     %let name2   =%upcase(&dsname);
     %let file2   =%trim(&path)\%trim(&file).xls;

/********************************* Error checking ******************************/
     %if %length(|&file)=1 %then
       %put ERROR: You must specify an Excel FILE name;
     %else %if %length(|&path)=1 %then
       %put ERROR: You must specify a path;
     %else %if %sysfunc(fileexist("&file2"))^=1  %then
       %put ERROR: File &file..xls was NOT found in &path;



     %else %do;    /*************** no errors, continue processing block **********/
         libname xlsdata "&file2" getnames=yes scantext=yes scan textsize=yes version=2002;
                                                                  /*open the workbook */
        %if %sysfunc(exist(xlsdata.meta dat))=1  %then %let filemode=VIRTUAL;
                                                        /* Metadata sheet does  exist */

        %if &filemode eq SINGLE and  %sysfunc(exist(xlsdata.%trim(&sheet)))^=1 %then %do;
           %put %str(ERR)OR: "&sheet" sheet was NOT found in file &file2   ;
        %end;

        %else %do;                                /* Start- found the sheet in the workbook*/

          %if filemode eq SINGLE %then %do;                 /*selected single sheet mode*/
              proc sql noprint;
                 create table &libref..dsname as
                   select * from xlsdata.%trim(&sheet);         /* import selected sheet */
              quit;
          %end;

          %else %do;                                          /*Start of Virtual sheet mode*/
             proc sql noprint;                                          /*get Metadata */
                select Nhsheets,Nvsheet, idvars     /*row and col set counts, ID varname */
                   into: rowsetcnt,
                       :  pagecnt,
                       : idvar lst2
                from xlsdata.meta_dat  ;
             quit;

             %let idvar lst2 = %sysfunc(translate(&idvar_lst2,' ',',')); /* take out commas*/
             %let errflag=0;

            %if %sysfunc(exist(%trim(&libref).%trim(&dsname)))=1  %then %do;
                                                  /* delete output datasset if it exists*/
              proc datasets library=%trim(&libref) nolist;  delete %trim(&dsname) ;  quit;
            %end;
```

```
            %do j=1 %to &rowsetcnt;                 /*Start of Loop through horiz. row sets*/

               %do i=1 %to &_pagecnt;             /*Start of Loop through vert. column sets*/

                   %let sheetnam = %str(VS )%trim(%left(&j))%str( )%trim(%left(&i));
                                                                          /* sheet name*/
                  %if %sysfunc(exist(xlsdata.&sheetnam))^=1 %then %do;
                     %let errflag=1;                                 /* sheet doesnt  exist*/
                     %put %str(ERR)OR: "&sheetnam" sheet was NOT found in file &file2   ;
                  %end;

                  %else %do;
                    proc sql noprint;                                 /*get data from a sheet */
                     create table  Ctmp&i as
                        select * from xlsdata.%trim(&sheetnam)
                        order by &idvar_lst2;
                    quit;
                  %end;
                %end;                                 /*End of Loop through vert. column sets*/

                %if &errflag=0 %then  %do;
                    data   Rtmp;                                  /*merge the vert. datasets*/
                      merge
                  %do i=1 %to &_pagecnt;
                      Ctmp&i
                  %end;
                   ;
                     by  &idvar_lst1;
                   run;

                   proc append force base=%trim(&libref).%trim(&dsname) data=  Rtmp;  run;
                                                          /*append the horiz.. datasets*/
                %end;

              %end;                                    /*End of Loop through horiz. row sets*/

            %end;                                                /*End of Virtual sheet mode*/

          %end;                                     /* End - found the sheet in the workbook*/

          %if %upcase(&debug) NE Y and %upcase(&debug) NE YES %then %do;   /*housecleaning*/
              proc datasets library=work nolist;  delete _Rtmp
            %do i=1 %to &_pagecnt;
                 _Ctmp&i
            %end;
            ;  quit;
               libname xlsdata  ;
          %end;

       %end;                                                 /* End - No Errors found block*/

%mend readxls;
```

5. SAMPLE RUNS

5.1. The WriteXLS macro

The sample dataset TEST1 has 100 obs and 302 columns. The sample call set the max number of columns per physical sheet at 200 and the max number of rows per physical sheet at 50 in order to demonstarte the splitting capability. That means that this sample produced a workbook named ‘c:\windows\temp\test2.xls’ that contains 5 sheets, i.e. 4 data sheets and 1 sheet with the metadata. Please see the boxed comments in the log, describing the function of the sections.

The Call:

```
 %writexls (dsname=test1,        file=TEST2,  IDvars=Study pt,
            path=c:\windows\temp,maxcols=200, maxrows=50);
```

The Log (with comments):

```
NOTE: There were 302 observations read from the data set SASHELP.VCOLUMN.
      WHERE (UPCASE(libname)='WORK') and (UPCASE(memname)='TEST1');
NOTE: The data set WORK._COLINF1 has 302 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time           0.21 seconds
      cpu time            0.21 seconds


NOTE: There were 302 observations read from the data set WORK._COLINF1.
NOTE: The data set WORK._COLINF1 has 302 observations and 3 variables.
NOTE: PROCEDURE SORT used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


NOTE: Table WORK._COLINF2 created, with 302 rows and 4 columns.

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.21 seconds
      cpu time            0.23 seconds


NOTE: Libref XLSDATA was successfully assigned as follows:
      Engine:        EXCEL
      Physical Name: c:\windows\temp\TEST2.xls
+-------------------------------------------------------------------------+
| SPECIAL NOTE : The dataset has too many columns to fit in one sheet !   |
|                It will be split into several sheets within a workbook.  |
+-------------------------------------------------------------------------+

NOTE: Table WORK.TEMPDAT created, with 100 rows and 198 columns.

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


NOTE: Deleting XLSDATA.VS_1_1 (memtype=DATA).
NOTE: PROCEDURE DATASETS used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


NOTE: There were 50 observations read from the data set WORK.TEMPDAT.
NOTE: The data set XLSDATA.VS_1_1 has 50 observations and 198 variables.
NOTE: DATA statement used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds


NOTE: Deleting XLSDATA.VS_2_1 (memtype=DATA).
NOTE: PROCEDURE DATASETS used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


NOTE: There were 50 observations read from the data set WORK.TEMPDAT.
NOTE: The data set XLSDATA.VS_2_1 has 50 observations and 198 variables.
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds


NOTE: Table WORK.TEMPDAT created, with 100 rows and 106 columns.

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds


NOTE: Deleting XLSDATA.VS_1_2 (memtype=DATA).
NOTE: PROCEDURE DATASETS used (Total process time):
```

Characterize the dataset to be exported

Needs splitting

Temporayy dataset with with 1st column set

Delete the physical sheet if it already exists.
XLS engine can't do replace.

Store physical sheet with 1st column set and 1st row set

Delete the physical sheet if it already exists.
XLS engine can't do replace.

Store physical sheet with 1st column set and 2nd row set

Temporary dataset with with 2nd column set

Delete the physical sheet if it already exists.
XLS engine can't do replace.

```
      real time             0.00 seconds
      cpu time              0.00 seconds


NOTE: There were 50 observations read from the data set WORK.TEMPDAT.
NOTE: The data set XLSDATA.VS_1_2 has 50 observations and 106 variables.
NOTE: DATA statement used (Total process time):
      real time             0.03 seconds
      cpu time              0.03 seconds


NOTE: Deleting XLSDATA.VS_2_2 (memtype=DATA).
NOTE: PROCEDURE DATASETS used (Total process time):
      real time             0.01 seconds
      cpu time              0.01 seconds


NOTE: There were 50 observations read from the data set WORK.TEMPDAT.
NOTE: The data set XLSDATA.VS_2_2 has 50 observations and 106 variables.
NOTE: DATA statement used (Total process time):
      real time             0.03 seconds
      cpu time              0.03 seconds


NOTE: Deleting XLSDATA.meta_dat (memtype=DATA).
NOTE: PROCEDURE DATASETS used (Total process time):
      real time             0.00 seconds
      cpu time              0.00 seconds


NOTE: The data set XLSDATA.meta_dat has 1 observations and 3 variables.
NOTE: DATA statement used (Total process time):
      real time             0.01 seconds
      cpu time              0.01 seconds


NOTE: Deleting WORK._COLINF1 (memtype=DATA).
NOTE: Deleting WORK._COLINF2 (memtype=DATA).
NOTE: Deleting WORK.TEMPDAT (memtype=DATA).
NOTE: PROCEDURE DATASETS used (Total process time):
      real time             0.00 seconds
      cpu time              0.00 seconds


NOTE: Libref XLSDATA has been deassigned.
```

Store physical sheet with 2nd column set and 1st row set

Delete the physical sheet if it already exists.
XLS engine can't do replace.

Store physical sheet with 2nd column set and 2nd row set

Delete the physical sheet if it already exists.
XLS engine can't do replace.

Store metadata sheet

Clean up

5.2. The ReadXLS macro

The sample file TEST2 used here is the output file from the writeXLS sample of section 5.1. As you'll remember, we wound up with four sheets in the workbook, representing 100 obs and 302 columns from the original input dataset to the writeXLS macro.. This readXLS sample call restores the dataset from the multiple sheets in the workbook. Please see the boxed comments in the log, describing the function of the sections.

The Call:

```
 %readxls(file=TEST2, path=c:\windows\temp,libref=work,
          dsname=imptest, debug=y)                      ;
```

The Log (with comments):

```
NOTE: Libref XLSDATA was successfully assigned as follows:
      Engine:        EXCEL
      Physical Name: c:\windows\temp\TEST2.xls
NOTE: PROCEDURE SQL used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Assign XLS libname

```
NOTE: Deleting WORK.IMPTEST (memtype=DATA).
NOTE: PROCEDURE DATASETS used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Delete the output dataset if it already exists from prior runs

```
NOTE: Table WORK._CTMP1 created, with 50 rows and 198 columns.

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.07 seconds
      cpu time            0.07 seconds
```

Import 1st vert. sheet into temporary dataset (1st row)

```
NOTE: Table WORK._CTMP2 created, with 50 rows and 106 columns.

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds
```

Import 2nd vert. sheet into temporary dataset (1st row)

```
NOTE: There were 50 observations read from the data set WORK._CTMP1.
NOTE: There were 50 observations read from the data set WORK._CTMP2.
NOTE: The data set WORK._RTMP has 50 observations and 302 variables.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Merge 1st and 2nd vert. datasets (1st row)

```
NOTE: Appending WORK._RTMP to WORK.IMPTEST.
NOTE: BASE data set does not exist. DATA file is being copied to BASE file.
NOTE: There were 50 observations read from the data set WORK._RTMP.
NOTE: The data set WORK.IMPTEST has 50 observations and 302 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Append combined vert. dataset to output D/S

```
NOTE: Table WORK._CTMP1 created, with 50 rows and 198 columns.

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.09 seconds
      cpu time            0.09 seconds
```

Import 1st vert. sheet into temporary dataset (2nd row)

```
NOTE: Table WORK._CTMP2 created, with 50 rows and 106 columns.

NOTE: PROCEDURE SQL used (Total process time):
      real time           0.04 seconds
      cpu time            0.04 seconds
```

Import 2nd vert. sheet into temporary dataset (2nd row)

```
NOTE: There were 50 observations read from the data set WORK._CTMP1.
NOTE: There were 50 observations read from the data set WORK._CTMP2.
NOTE: The data set WORK._RTMP has 50 observations and 302 variables.
NOTE: DATA statement used (Total process time):
      real time           0.01 seconds
      cpu time            0.01 seconds
```

Merge 1st and 2nd vert. datasets (2nd row)

```
NOTE: Appending WORK._RTMP to WORK.IMPTEST.
NOTE: There were 50 observations read from the data set WORK._RTMP.
NOTE: 50 observations added.
NOTE: The data set WORK.IMPTEST has 100 observations and 302 variables.
NOTE: PROCEDURE APPEND used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds
```

Append combined vert. dataset to output D/S

CONCLUSIONS

The readXLS and writeXLS macros are extremely useful macro tools to help automate and simplify the importing and exporting SAS data from and to Excel without being concerned about Excel limitations. .These Interface macros can be used by any application programs that need to deal with very large amounts of data stored in Excel.

CONTACT INFORMATION

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

John H Adams
900 Ridgebury Road
Ridgefield, CT, 06877-0368
Work Phone: 203-778-7820
Fax: 203-837-4413
Email: john.adams@boehringer-ingelheim.com
adamsjh@mindspring.com