

**Smart Import/Append Data in Excel Sheets**  
Zhengping Ma, Eli Lilly and Company, Indianapolis, IN  
Liping Liu, i3 StatProbe, Indianapolis, IN

**ABSTRACT**

It is common that we have data stored in various forms in clinical research. One commonly used format is Microsoft Excel spreadsheet. However, data analysis is often conducted through SAS software. There are many SAS procedures, such as PROC import, PROC download, etc, which can be used to load Excel files into SAS. Another option is to use DDE (Dynamic Data Exchange), especially to load Excel files with multiple sheets. We can also use SAS®9 LIBNAME Engine to get sheet names dynamically.

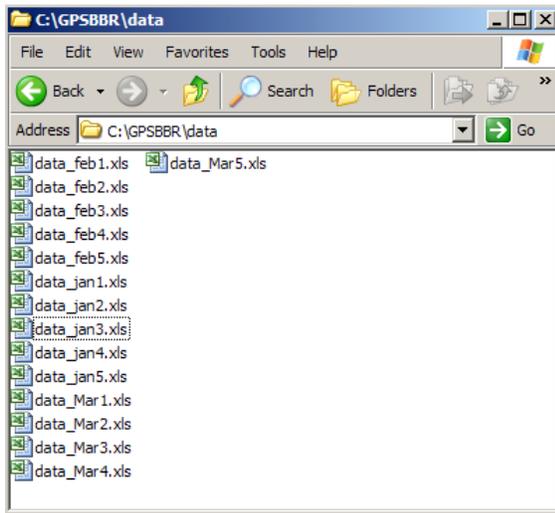
In several projects we worked on, periodic data transfers (e.g. weekly, monthly, etc) were scheduled so that Excel files accumulated over time. In other circumstances, similar data files accumulated over multiple studies. There is a need for automating the loading of source data files and combining them either into study specific datasets or into an IDB across several studies. In this paper, we present macro definitions that can load all Excel files with multiple sheets into SAS datasets, either using a user specified name, or using the combination of file name and sheet name as the corresponding dataset name. The smartness of the macros include: 1) Ability to handle an unspecified number of data files with multiple sheets, while maintaining the relationship of SAS datasets with their source; 2) Ability to handle special characters to create valid SAS dataset names based on file name/sheet name combination, and to truncate the name to 32 characters, if necessary; 3) Ability to load selected data sheets into consolidated SAS datasets (e.g., load all drug use data stored in sheet 'drug' across several Excel files into SAS dataset 'drug'); 4) Ability to append longer character values in new data instead of having it truncated as when using proc append.

**INTRODUCTION**

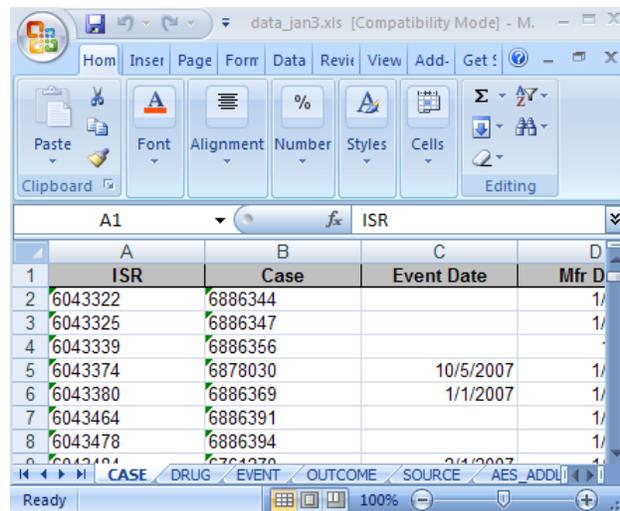
Microsoft Excel spreadsheet is often used to store clinical research data (such as lab data from vendors). On the other hand, data analysis is often conducted using SAS software. Therefore, we need to import Excel data sheets into SAS datasets before we conduct any data analysis. To import a single file, we can simply use PROC import, or use the import wizard in SAS. To load Excel files with multiple sheets, Helen Sun, et al. (2005) presented a macro %xl2sas based on DDE (Dynamic Data Exchange) technology. In addition, Fan (2006) presented a multi-book, multi-sheet importation solution based on the SAS®9 LIBNAME Engine. In Fan's macro, data from different sheets with different number of variables are forced to be appended into one consolidated SAS dataset, which may not necessarily be what is needed.

**THE PROBLEM**

In several projects we worked on, data files were accumulated over time. In other circumstances, similar data files were accumulated over several studies. For example Figure 1a illustrates a data folder with multiple data files accumulated in the first quarter of the year. Figure 1b shows a sample data file with several sheets.



**Figure 2a. Data folder with multiple files**



**Figure 1b. a sample Excel file with multi-sheets**

We need to load three selected sheets (case, drug, and event) into SAS datasets DEMO, DRUGS and EVENTS, respectively. All datasets are stored in a user specified location, with all the case data from the input files stored in dataset DEMO, and similarly for drug and events data. In other cases, we may need to import all Excel data files with varying content or number of data sheets into SAS datasets. To trace the relationship between SAS datasets and their original source, we need to use the combination of file name and sheet name as the corresponding dataset name. Since SAS does not allow some special characters as part of a valid SAS name, for each combination, we need to replace some special characters in the file name/sheet names before they can be used as dataset names; also, SAS does not allow names exceeding 32 characters. We need to truncate long names to 32 characters.

We present macro definitions which are capable of all the necessary functions. Since we need to append an unknown number of new datasets into existing datasets in the first use case above, for categorical variables, it is very likely that new values in the new data file exceed the length of the variables in the existing datasets. Thus, the values will be truncated if PROC APPEND is used. We include a solution to avoid truncating new values in this paper.

## LOADING ALL FILES WITHIN A SPECIFIED LOCATION

Given some location information, without knowing names of the files and number of files, the first thing we need is to create a list of file names to process. The code below creates such a list of all the files with .xls as its extension within a pre-specified location:

```
filename dir pipe "dir &fileLoc.\ /b *.xls";
%if &outlib. eq %then %let outlib=work;

data temp;
  infile dir;
  input;
  fn = _infile_;
  dotPos = index(_infile_,'.xls')-1;

  if dotPos > 0;
  dsm = substr(_infile_, 1, dotPos);
run;

filename dir clear;

proc sql noprint;
select dsm into :dsms SEPARATED BY ' ' from temp;
select fn into :fns SEPARATED BY ' ' from temp;
quit;
```

It first creates a pipe based on the folder location provided by the user. The pipe is used as input to create a temporary dataset with the name of a file as an observation. It also picks up the file name without extension, which will be used as the first half of SAS dataset names. All the complete file names are stored in macro &fns as strings separated by a blank space, and all the file names without extension are similarly stored in macro &dsms.

The next step is to load all the data sheets into SAS datasets, processing one file at a time. We use a %do %while loop to work through the strings stored in the macro variables, as illustrated below:

```

%let delim=%str( );
%let j=1;
%let fn=%scan(&fns,&j,&delim);
%let dspfx=%scan(&dsms,&j,&delim);

data _null_;
%do %while (&fn ne );
%if %sysfunc(fileexist(&fileLoc.\&fn)) %then %do;
/* read sheets with no dataset/sheet names specified*/
%if &sheetNM eq %then %do;
%put INFO: Process file &fn with no sheet/dataset name specified;
%ReadXls(path=&fileLoc,inf=&fn, dspfx=&dspfx);
%end;
/* user provided sheet/dataset names, use it as specified*/
%else %do;
%let i=1;
%let snm=%scan(&sheetNM,&i,&delim);
%if &dsnm eq %then %do; %let dsn=&snm; %end;
%else %let dsn=%scan(&dsnm,&i,&delim);

%let dspfx=%scan(&dsms,&j,&delim);

%do %while (&snm ne );
%importXL(inFile=&fileLoc.\&fn., sheetNM=&snm, outDS=&dspfx.&dsn);
%if %sysfunc(exist(&outlib..&dsn)) %then %do;
%append_adjlength(ds1=&outlib..&dsn, ds2=&dspfx.&dsn, out=&outlib..&dsn);
%end;
%else %do;
data &outlib..&dsn;
set &dspfx.&dsn;
run;
%end;
%let i=%eval(&i+1);
%let snm=%scan(&sheetNM,&i,&delim);

%if &dsnm eq %then %let dsn=&snm;
%else %let dsn=%scan(&dsnm,&i,&delim);

%end;
%end;
/* end if file exist */

%else %put File &fn does NOT exist, go to next file ...;

%let j=%eval(&j+1);
%let fn=%scan(&fns,&j,&delim);
%let dspfx=%scan(&dsms,&j,&delim);

```

The code first checks the existence of the input file, and then checks whether sheet names are provided. If not, it calls macro %ReadXls to read a multi-sheet Excel file, using a combination of file name and sheet names as SAS dataset names. When sheet names are provided, it assumes that only those data sheets with their names specified are needed. Therefore, the macro walks through all the provided sheet names and creates a temporary dataset for each file/sheet combination. It also checks whether dataset names are provided, and if not, the sheet names are used for dataset names. Then, based on the specified output library, it checks whether a dataset with the specified name exists, and if not, creates a permanent copy into the library. Otherwise, if a dataset with the same name exists, it appends the new dataset to the existing dataset.

## IMPORTING FILES WITHOUT SHEET /DATASET NAME SPECIFIED

The details of importing files with no sheet names and datasets provided are implemented in macro %ReadXls. The first step is to extract all the sheet names within an Excel file. From Fan's (2006) implementation based on SAS@9 LIBNAME Engine, we create a series of macro variables to hold all the data sheet names within a file, as illustrated below:

```

%macro ReadXls (path=,inf=, dspfx=);
libname excellib excel "&path.\&inf";
proc sql noprint;
  create table sheetname as
  select tranwrd(memname, "''", "'") as sheetname
  from sashelp.vstabvw
  where libname="EXCELLIB";

  select count(DISTINCT sheetname) into :cnt_sht
  from sheetname;

  select DISTINCT sheetname into :sheet1 - :sheet%left(&cnt_sht)
  from sheetname;

quit;

libname excellib clear;
...
%mend ReadXls;

```

Then, we use a loop to work through all the data sheets one at a time. Note that we need to make sure that the dataset names derived based on file names and sheet names are valid as a SAS name. For our project, for each file name/sheet name combination, we need to remove \$ and replace blank spaces with underscores, and then truncate to 32 characters, if necessary. Also, to deal with the possibility of duplicated dataset names based on file name and sheet name combination, when a new dataset name has been created, it is checked against existing datasets. If it's a duplicate, it's truncated to 31 characters with a number attached as its 32<sup>nd</sup> letter, starting from 1, with details as shown below:

```

%do i=1 %to &cnt_sht;
  %put INFO: importing file &inf., sheet name: &&sheet&i;

  /* truncate it to up to 32 chars */
  %let dsname=&dspfx._&ds&i;
  %let dsname = %replace(src=& dsname, oldvalue=$, newvalue=);
  %let dsname = %replace(src=& dsname, oldvalue=%str( ), newvalue=_);
  %if %length(&dsname)>32 %then %do;
    %let dsname=%substr(&dsname, 1, 32);
  %end;
  %let j=1;
  %do %while (%sysfunc(exist(&outlib..&dsname)));
    %let dsname=%substr(&dsname, 1, 31);
    %let dsname=&dsname.&j;
    %let j=%eval(&j+1);
  %end;
  %put the dataset name will be: &dsname;
  %importXL(inFile=&path.\&inf., sheetNM=&&sheet&i, outDS=&outlib..&dsname);
%end;

```

The details of macro %replace are as follows. It should be pointed out that the macro below has some limitations. For example, it does not work when a string contains characters such as '(' or ')'. Therefore, when such letters are used as part of file name/sheet name, a data step is needed to replace them with some other letters such as underscore, using function 'tranwrd'.

```

/* macro to used to replace special characters for dataset names */
%macro replace(src, oldvalue, newvalue);
  %let outs=&src;
  %do %while(%index(&outs, &oldvalue) > 0);
    %let pos = %index(&outs, &oldvalue);
    %if &pos > 0 %then %do;
      %let left = %substr(&outs, 1, %eval(&pos - 1));
      %if %length(&outs) > %eval(&pos + %length(&oldvalue)) %then %do;
        %let right = %substr(&outs, %eval(&pos + %length(&oldvalue)));
      %end;
      %else %let right=;
      %let outs=&left&newvalue&right;
    %end;
  %end;
  %let outs
%mend;

```

Macro %importXL implements a call to PROC import, as follows:

```
/* macro to import one sheet within an Excel file */
%macro importXL(inFile=, sheetNM=, range=, outDS=);
PROC IMPORT OUT= &outDS.
    DATAFILE= "&inFile."
    DBMS=EXCEL REPLACE;
    %if %length(&sheetNM.) ne %then %do;
        %if &range. ne %then %do;
            range="&sheetNM.$&range.";
        %end;

        %else %do;
            SHEET="&sheetNM.";
        %end;
    %end;
MIXED=yes;
SCANTEXT=YES;
GETNAMES=YES;
RUN;
%mend;
```

## AVOIDING TRUNCATION OF CHARACTER VALUES WHEN APPENDING NEW DATA

When we need to combine data from multiple input files but with the same structure into one SAS dataset, for categorical variables, we need to take special care so that new values longer than the defined length of the variable in pre-existing data are not truncated. When the dataset is created by importing from Excel file, the length of a character variable is usually defined by the longest values SAS scanned. To avoid truncating of longer values within an input sheet, we can force SAS to do a whole file scan. To do this, open Windows registry editor, then set the value of the key 'HKEY\_LOCAL\_MACHINE\Software\Microsoft\Jet4.0\Engines\Excel\TypeGuessRows' to 0. Note that even when you set mixed option to 'yes' during your import, it is still possible to see that SAS sets input character values as missing when a column contains mixed value types. A whole file scan prevents this from happening.

Typically, we use PROC append to append new data to an existing dataset. It works well when the new dataset shares the exact same structure as the existing dataset. However, as I pointed out above, when we deal with datasets created by import from Excel files, it is very likely to see that same character variable is defined with different lengths. If the variable in the new dataset comes with longer length, proc append will truncate the longer values in the new dataset. To prevent this, we define the macro %append\_adjlength to combine the new data with existing data.

```
%macro append_adjlength(ds1=, ds2=, outds=);
/* get information about the input datasets */
proc contents data=&ds1 noprint
    out=out1(keep=name type length where=(type=2));

proc contents data=&ds2 noprint
    out=out2(keep=name type length where=(type=2));
run;

/* pick up the longer length for the combined dataset*/
data _null_;
    file "mergedata.sas";
    merge out1 out2(rename=(length=length2)) end=last;
    by name;
    if _n_ = 1 then put "Data &outds.";
    newlength = max(length,length2);
    put "    length " name " $ " newlength 4. " ";";
    if last then do;
        put "    set &ds1 &ds2.";
        put "run.";
    end;
run;

%include " mergedata.sas";
```

```
%mend append_adjlength;
```

First, it calls PROC contents to find out the variable definitions. For character variables, it picks up the longer length as the length for the combined dataset. Datasets are combined with a data step instead of using APPEND procedure. This macro is called when sheet names are provided by a user. It expects that the user wants to combine datasets from multiple input files with the same sheet names.

## PUTTING THE PIECES TOGETHER

Macro %importXLfiles is the user interface for all macros discussed here. At minimum, it expects an input for data source file location. Users can also specify output library location. If it is not provided, work library will be used as default. Parameters sheetnm and dsnm are used to specify selected sheet names and corresponding dataset names for output datasets.

```
%macro importXLfiles(fileLoc=, outlib=work, sheetnm=, dsnm=);
  %if &fileLoc= %then %do;
    %put ERROR: No file location specified, exiting with no file imported;
    %goto %terminate;
  %end;

  filename dir pipe "dir &fileLoc.\ /b *.xls";
  %if &outlib. eq %then %let outlib=work;

  data temp;
    infile dir;
    input;
    fn = _infile_;
    dotPos = index(_infile_,'.xls')-1;

    if dotPos > 0 then dsm = substr(_infile_, 1, dotPos);
    if dotPos>0;
  run;

  filename dir clear;

  proc sql noprint;
  select dsm into :dsms SEPARATED BY ' ' from temp;
  select fn into :fns SEPARATED BY ' ' from temp;
  quit;
  %put Input file names: &fns;
  %let delim=%str( );
  %let j=1;
  %let fn=%scan(&fns,&j,&delim);
  %let dspfx=%scan(&dsms,&j,&delim);

  data _null_;
  %do %while (&fn ne );
  %if %sysfunc(fileexist(&fileLoc.\&fn)) %then %do;
    %put File &fn exists, processing...;
    %if &sheetNM eq %then %do; /* read all sheets in the file */
      %put INFO: process Excel file &fn without sheet name/dataset name specified;
      %ReadXls(path=&fileLoc,inf=&fn, dspfx=&dspfx);
    %end;
  %else %do;
    %let i=1;
    %let snm=%scan(&sheetNM,&i,&delim);

    %if &dsnM eq %then %do; %let dsn=&snm; %end;
    %else %let dsn=%scan(&dsnM,&i,&delim);

    %let dspfx=%scan(&dsms,&j,&delim);

    %do %while (&snm ne );
```

```

%importXL(infile=&fileLoc.\&fn., sheetNM=&snm, outDS=&dspfx.&dsn);
%if %sysfunc(exist(&outlib..&dsn)) %then %do;
    %append_adjlength(ds1=&outlib..&dsn, ds2=&dspfx.&dsn,out=&outlib..&dsn);
%end;
%else %do;
    data &outlib..&dsn;
        set &dspfx.&dsn;
    run;
%end;
%let i=%eval(&i+1);
%let snm=%scan(&sheetNM,&i,&delim);

%if &dsnm eq %then %let dsn=&snm;
%else %let dsn=%scan(&dsnm,&i,&delim);

%end;

%end;

%end; /* end if file exist */

%else %put File &fn does NOT exist, go to next file ...;

%let j=%eval(&j+1);
%let fn=%scan(&fns,&j,&delim);
%let dspfx=%scan(&dsms,&j,&delim);

%end;

Run;
%terminate;
%mend;

```

## SAMPLE CALLS FOR DIFFERENT USES

Below are two sample calls. The first call provides the location for data source files and output library for SAS output datasets. Since no sheet names/dataset names are provided, the macro loads all the input files and creates one dataset for each datasheet, using the concatenated file name/sheet name as dataset name to keep the link between the created SAS dataset and its source.

```

%importXLfiles(fileLoc=&srcLoc, outlib=xxx);
%importXLfiles(fileLoc=&srcLoc, outlib=xxx, sheetnm=case event drug, dsnm=demo
event drug);

```

The second call provides additional information for sheet names and dataset names. The macro loads all the input files within the specified location. For each input file, it imports the selected sheets (case, event and drug) and appends new data into SAS datasets (demo, event, and drug, respectively) stored in the output library.

## CONCLUSION

In this paper, we presented macro definitions to import all Excel files within a user specified source location. The macros can be used to load all data files into SAS without any information about the file names/data sheet names. By concatenating file names with sheet names, it keeps track of the relationship between created datasets and their source(s). It can also be used to load selected data sheets and combine all data from multiple source files into specified SAS datasets, without being given the source file names. In addition, we also presented some tips for avoiding setting character values to missing for columns with mixed values, truncating character values when new values are longer than variable length specification in the existing dataset during append, etc.

## REFERENCES

- 1) Helen Sun, et al. Robarts Clinical Trials, London, ON, CANADA. A Macro for Importing Multiple Excel Worksheets into SAS® Data Sets, Proceedings of the 30th Annual SAS Users Group International Conference, Paper 040-30, 2005
- 2) Zizhong Fan, Westat, Rockville, MD. Make the Invisible Visible: A Case Study of Importing Multiple Worksheet Files By Using the SAS®9 LIBNAME Engine in Microsoft Excel, Proceedings of the 31th Annual SAS Users Group International Conference, Paper 034-31, 2006

## ACKNOWLEDGMENTS

We'd like to thank Zhiyan Ma, Nancy Brucken, Colleen Benjamin and Elaine Dempsey for their review and suggestions for improvement.

## CONTACT INFORMATION

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

Name: Zhengping Ma  
Enterprise: Eli Lilly & Company  
Address: Lilly Corporate Center  
City, State ZIP: Indianapolis, IN 46285  
E-mail: mazh@lilly.com

Name: Liping Liu  
Enterprise: i3 Statprobe  
Address: 4745 Haven Point Boulevard  
City, State ZIP: Indianapolis, IN 46280  
E-mail: Liping.Liu@i3statprobe.com

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.