Paper 28

Checking and Tracking SAS® Programs Using SAS® Software

Keith M. Gregg, Ph.D., SCIREX Corporation, Chicago, IL
Yefim Gershteyn, Ph.D., SCIREX Corporation, Chicago, IL

ABSTRACT

Various checks on consistency and changes in the SAS programs in large scale projects can be efficiently implemented using the SAS language itself. This is done by obtaining lists of all files and their requisites in the project program and data directories and comparing these lists; extracting from the SAS log files information on permanent SAS data sets used in the corresponding SAS runs; and utilizing information from standardized comment fields in the headers of SAS programs. The information gathered in the course of this process is also useful for project tracking. The SAS language features that make this possible include interaction with DOS (Windows NT) command output using the PIPE device type in the FILENAME statement; information in the SAS log on the date and time of the SAS run and on physical names of LIBREFs; and the two-level name syntax for permanent data sets and the limited ways of referencing existing data sets in a SAS program (SET, MERGE and other statements in DATA step; DATA= option in procedures; etc.).

INTRODUCTION

Some projects concerning analysis of clinical trials data involve thousands of SAS programs and hundreds of permanent data sets. It is extremely important to keep track of changes in the programs and assure that the latest data are used. SAS can be used to perform some routine checks and to obtain valuable information on the SAS program, log, and output files. Specifically, two main tasks are performed by a SAS program we have developed:

- Check the consistency of SAS programs, logs, and listing files (correct location within the project directory structure, whether each of the SAS log or listing files has been produced by the corresponding version of the SAS program, etc.) and obtain the date and time of creation (last modification) and the size of those files. Also, any other of the SAS output files, such as printer control language (.PCL) files or graphic (.GPH) files, can be included in the check.
- Extract information from standardized SAS program header comments such as project, date of creation (last modification), author, etc. and information about the time of creation (last modification) of permanent data sets used in the program to ensure the most recent version of the data set was used. By comparing modification dates and times of SAS programs, their inputs, and their outputs in the manner described above, a form of version control is achieved.

SAS FEATURES USED

The program and involved macros were developed using SAS version 6.12 under Microsoft Windows NT 4.0. Four points should be mentioned which made it possible to obtain the necessary information.

First, interaction between SAS and MS-DOS (i.e., the Windows NT Command Prompt) using the PIPE device type in a FILENAME statement allows the creation of a SAS data set with the necessary information about the files in a directory and it subdirectories.

Second, information is extracted from the SAS log, such as the date and time of the SAS run and LIBREF names, which are located in certain fixed places and worded in standard ways.

Third, the SAS syntax for permanent data set names (two-level name with a dot dividing the LIBREF and the data set name), allows for organizing a search in a SAS log for permanent data sets used in the SAS program.

Finally, there are limited ways to reference permanent data sets in a SAS program. Taking into consideration the specifics of the projects regarding statistical processing of clinical trials data, those data sets can be used in DATA step SET and MERGE statements, in FROM clauses in the SQL procedure, and in procedures with the option DATA= (or BASE= in the APPEND procedure). The program employs various searches in the SAS log for these statements and procedures to extract the permanent data set names used in the SAS run.

PROGRAM DESIGN

The program we are discussing is relatively long and complicated. Below we emphasize its most important and interesting parts. Some code given here was simplified to better express the main goal and to not concentrate on trivial features. In general, the program:

- obtains the list of all files in the project program and data directories with file names (including extension), size, and date and time of creation (last modification);
- compares files within directories to find matching names with different extensions, then maps the files to extract information from the different kinds of files (e.g., a .SAS file modified more recently than the corresponding .LOG file, indicating that the program has been changed but not re-run);
- searches each .LOG file in the project program directory for names of permanent SAS data sets used in the corresponding SAS runs, and then compares the time of the creation (last modification) of the data sets to the time of the SAS run and finds cases of using non-updated versions of the data sets;
- reads information from the fields in standardized comment headers of a SAS program and obtains the project name, date, name of the author of the program, etc., and simultaneously finds incorrect or blank fields;
- creates various data sets containing the information described above, including errors and inconsistencies, and, if necessary, reports problems and produces various statistics on the SAS programs and outputs of the project.

The program had three general design goals: sufficient flexibility to allow it to be used for different projects; logical order of program code; and the possibility of applying, if needed, some program fragments independently. The program has the following seven parts:

1. Execute system options, initializations, and %INCLUDE statements which help to share code among projects.

```
%include 'proj.sas';
%include 'util.sas';
%include 'os.sas';
%include 'filinf.sas';
%include 'sashead.sas';
%include 'dateset.sas';
```

The *%proj* macro simply sets macro variables and library names that must be customized for each project. The macros in the *util.sas* file perform some routine tasks such as checking for missing values and creating variables for file names. Macro *%oscheck*, residing in the *os.sas* file, detects the operating system (OS) under which the program is running and makes sure that the OS is a suitable one, otherwise an error message in the log is generated. The OS is determining by parsing the output of the command prompt VER command:

```
Windows NT Version 4.0
```

The filename statement

```
filename _vercmd pipe 'ver';
```

allows one to read the output from the DOS VER command into a SAS data set.

Macros *%fileinf*, *%sashead*, and *%dateset* perform some major program tasks and are discussed below.

2. Create two SAS data sets with the names, sizes, and dates and times of creation (last modification) of all permanent data sets in the project data directories and programs in the project SAS program directories.

This is performed by calling the *%filinf* macro twice. The macro goes through the directories, containing respectively SAS data sets and programs (with *&home* and *&dir* being macro parameters specifying the directory), forming a command prompt DIR command to list the files in the directory and all its subdirectories along with the date, time, and size of the files:

```
filename _dircmd pipe "dir /a-d /s -p
&home\&dir";
```

Then, using as a basis the standard DOS DIR command output layout which looks as follows

```
 Directory of D:\MYDIR\MYSUBDIR

01/01/98  12:00a          19,999  MYFILE.SAS
01/01/98  12:01a          29,999  HISFILE.SAS
               2 File(s)   49,998  bytes
```

the macro creates a SAS data set of filenames, sizes, and dates and times of creation (last modification). For example, the line containing the text

```
               2 File(s)   49,998  bytes
```

indicates the end of a list of files in a certain directory, which can be used in SAS code as an indicator to end the loop reading in file names and their information in that directory.

```
** READ THE OUTPUT OF THE DIR COMMAND INTO A
** DATA SET. ;
data _filein1;
  format fdttm datetime13. fsz comma21.;
  infile _dircmd  length=len;
  input @;          * GET LINE LENGTH ;
  input line $varying200. len;
** KEEP ONLY LINES WITH FILE AND DIRECTORY
** NAMES;
  if line ne '' and not
  index (line, 'Volume') and not
  index(line, 'File(s)') and not
  index(line, 'Total Files Listed:') and not
  index(line, 'bytes free');
** OBTAIN DIRECTORY NAME AND KEEP IT FOR ALL
** FILES IN THAT DIRECTORY;
  if index(line, 'Directory of') then do;
       direct =scan(upcase(line), 3, ' ');
       retain direct;
  end;
** NOW KEEP ONLY LINES WITH FILE NAMES;
  if not index(upcase(line), 'DIRECTORY OF');
** OBTAIN INFORMATION ON FILES;
  date_= input(substr(line,1,8),  mmddyy8.);
  timestr= scan(line, 2, ' ');
  time = input(substr(timestr, 1, 5),time5.);
** TIME WITHOUT AM/PM;
   select (substr(timestr, 6, 1));
** FIX AM/PM..;
     when ('p')
     if time < '12:00't then time = time +
       '12:00't;
     when ('a')
     if time >= '12:00't then time = time
     - '12:00't;
      otherwise
      error 'ERROR: Modification time string '
      timestr 'did not end with ''a'' or
      ''p''.';
    end;
    fdttm=dhms(date_, hour(time),
          minute(time), second(time));
    filen = upcase(scan(line, 4, ' '));
    fnvar = compress(direct||'\'||filen);
    fsz = input(scan(line, 3, ' '),
         comma21.);
    keep fnvar fdttm fsz;
run;
```

3. Extract the extensions and the file names without the extensions in the two lists of files. First, the *%fnparts* macro call creates two variables for each file containing the full file name without and with the extension. Then the files are sorted by the filenames without the extensions and various types of files are split off.

```
data files;
    set files;
    %fnparts(fn, fnnoext, fnext);
proc sort data=files;
     by fnnoext;
data sasfiles logfiles lstfiles
     etcfiles;
     set files;
     select (fnext);
        when ('.SAS') output sasfiles;
        when ('.LOG') output logfiles;
        when ('.LST') output lstfiles;
        otherwise     output etcfiles;
     end;
     drop fnext;
run;
```

4. Obtain information about permanent data sets used in a program and compare the time of the data sets' creation (last modification) with the time of the SAS run to find runs with non-updated data sets. This step calls the *%dateset* macro for each log file detected in the project directories. The macro does the following:

- Reads in and then outputs to the data set named *daytime* the time and date of the run from the first line of the log, which looks like (due to the layout of this paper the actual line occupies three lines)**:**

  ```
  1
  The SAS System           20:22 Saturday,
  August 2, 1998
  ```

- Extracts the names of permanent data sets used in the program from all possible pieces of code, e.g., for code like

  ```
  data= ooh.oh;
  ```

  coming from a procedure

  ```
  if index(upcase(compress(line)), 'DATA=')
  and index(upcase(line), '.') and not
  index(upcase(line), ' .') nd not
  index(upcase(line), '. ') and not
  index(line, '*') and not
  index(upcase(line), '.SAS') and not
  index(upcase(line), '.SD2') and not
  index(line, '&') and not
  index(line, '.0') and not
  index(line, '.1') and not
  ```

```
index(line, '.2') and not
index(line, '.3') and not
index(line, '.4') and not
index(line, '.5') and not
index(line, '.6') and not
index(line, '.7') and not
index(line, '.8') and not
index(line, '.9')  then do;
   if indexc(scan(substr(line,
   index(upcase(compress(line)),
   'DATA=')), 2, '='), '.') then do;
       mystrind = substr(compress(line),
       index(upcase(compress(line)),
       'DATA=');
       if index(mystrind, '.') and not
       index(mystrind, 'format') then do;
          libd = scan(upcase(mystrind),2
          ,'= .');
          setd = upcase(scan(substr(line,
          index(upcase(line),
          compress(libd))),
          2,'= . :(;)'));
          output datast;
        end;
   end;
end;
```

We have to include several conditions to avoid reading in other lines from the SAS log that coincidentally contain a dot and DATA= code. Thus, we have to make a provision for comments, numbers with decimal places, macro code, and so on. The library name *(libd*) and set name (*setd*) are output to a special data set named *datast.*

In a similar but sometimes more complicated manner, the macro reads in names of data sets which are in fragments of code like:

```
proc append base=ooh.oh;
set ooh.aaah1 oooh.aaah2;
merge ooh.aaah1 oooh.aaah2;
from ooh.aaah1.;
```

Also, 'loose' code from data steps such as

```
data a1;
set a1.b1 a2.b2  a3.b3;
```

has been accounted for.

The program correctly reads every SAS log with which we have tested it. There is, however, no guarantee that some code does not exist, which the program could not read properly and extract library and data set names correctly. One thing that helps identify code that might be mistakenly read in as a permanent data set name is putting in the log of our program the error message *'Libref physical name log message not found'*, which indicates 'suspicious' spots in the log being read.

On the other hand, with the complexity of the syntax of the SAS language (e.g., %INCLUDE statements, comments, macros and macro variables, etc.), the variety of ways a data set can be addressed (various DATA step statements and procedure options and clauses), the most solid way to find all the data set references with absolute certainty would be to write a SAS language (including the macro language) interpreter. Also, the possibility of using the RTRACE option has been considered for this purpose. This option generates a list of all the files that the SAS System opens and closes during a particular session. The option, however, must be specified when each program runs, which requires that all the programs in the project contain this option. Also, the output file produced by the RTRACE option can be very large, which complicates a search for the names of the permanent SAS data sets.

Meanwhile, our method will find all or almost all permanent data sets used in a particular program with the aid of knowledge of the specifics of the programs, purposes, and structure. Also, the performance could be improved by establishing reasonable standards for programming style.

- Extracts physical names of LIBREFs used in the log. This is done by reading in the third line of the log following each line of the form:

  ```
  NOTE: Libref <NAME> was successfully
  assigned as follows:
  ```

  Also, provisions have been made to accommodate cases when there is a page break in the log between those lines.

```
** OBTAIN PHYSICAL NAMES OF LIBREFS ;
data librefs;
    infile _logfile ls=200 pad;
    input line $char200.;
    if index(line, 'NOTE: Libref')
    then do;
       sppos = index(substr(line, 14),
       ' ');   ** SPACE AFTER LIBREF ;
       if sppos then do;
           libref = substr(line, 14,
           sppos);  ** THE LIBREF ;
           if substr(line, sppos + 13)
            = ' was successfully
            assigned as follows: '
            then do;
** SKIP OVER THE ENGINE MESSAGE ;
              input line $char200.;
** FOR SAS VERSIONS 6.11 OR 6.12. FOR
** OTHERS, THIS PART SHOULD BE CHANGED;
             if line = '      Engine:
             V611'
             or line = '      Engine:
             V612' then do;
** GET THE PHYSICAL NAME ;
               input line $char200.;
               if index(line, '
               Physical Name: ') = 1
               then do;
                 physname =
                 upcase(substr(line,22));
                 output;
               end;
** IF A PAGE BREAK IN THE LOG AFTER
** SECOND LINE;
                else if index(line,
                'The SAS System')
                then do;
** BLANK LINE;
                  input;
                  input line
                  $char200.;
                  if index(line, '
                  Physical Name: ') = 1
                  then do;
                    physname =
                    upcase(substr
                    (line, 22));
                    output;
                  end;
                  else error 'ERROR:
                  Libref physical name
                  log message not
                  found.';
                end;
               end;
** IF A PAGE BREAK IN THE LOG AFTER
** FIRST LINE;
       < code similar to the above>
               else error 'ERROR:
               Expected libref log
               message ending not
               found.';
```

```
            end;
         else error 'ERROR: Space
         following libref in libref log
         message not found.';
         end;
      end;
      keep libref physname;
   run;
```

- Finally, as an output from the macro, a data set is created with the full physical names of the permanent data sets used in the SAS run. This information includes flagging programs that reference non-updated data sets.

5. Create a data set with information about the permanent data sets used in all log files (SAS runs) for the project.

```
proc sort
data=allset(rename=(filenam=fndat));
proc sort data=datfiles;
data logdat;
    merge allset(in=a) datfiles(in=b);
    by fndat;
    if a;
    if not b then nodatset = 1;
    else nodatset = 0;
** WHETHER PERMANENT DATA SETS FROM STANDARD
** DATA DIRECTORIES WERE USED;
    if fdattm > daytime_ then noupdate =
    1;
    else noupdate = 0;
run;
** WHETHER THE UPDATED VERSION OF A
** PERMANENT DATA SET WERE USED;
data logprobl;
     set logdat (drop=nodatset);
     where noupdate=1;
proc sort data=logprobl;
     by logfn;
** TRANSFORM TO ONE OBSERVATION FOR EACH LOG
** FILE ;
proc sql;
     create table logdata
     as select logfn,
               sum(nodatset) as nodatset,
               sum(noupdate) as noupdate
    from logdat
    group by logfn
    order by logfn;
quit;
```

6. Obtain information about SAS program header standard comment fields and find any errors. This step invokes the *%sashead* macro with the following arguments:

- *fn* - filename (unquoted text string);
- *outds* - name of output data set.,
- *fnvar* - name of variable in data set *&outds* holding filenames;
- *pgmvar*, *authvar* , *datevar*, etc. - names of variables in the data set *&outds* to hold, respectively, program name, author, modification date, and other standard fields in the header.

The macro reads in the header of the SAS program (assuming that the first 50 lines of the program is the maximum number of lines to hold the header information), and then using standard field names like *'Sponsor'*, *'Project'*, etc. finds the value of character variables representing the contents of these fields.

```
filename _sashd1 "&fn";
data _sashd1(drop=line);
   length &fnvar &spnsrvar &projvar &pgmvar
   &authvar &vervar $ 200;
   retain &spnsrvar &projvar &pgmvar &authvar
   &datevar &vervar;
   &fnvar = "&fn";
   infile _sashd1 length=len obs=50;
   input @;            ** GET LINE LENGTH ;
     input line $varying200. len;
** REMOVE LEADING SPACES ;
     line = left(line);
** REMOVE LEADING ASTERISKS ;
line = substr(line, verify(line, '*'));
** REMOVE LEADING SPACES ;
line = left(line);
**REMOVE TRAILING *S & SEMI-COLONS ;
line = substr(line, 1, length(line) -
       verify(reverse(trim(line)), '*;') +1);
     if upcase(line) =: 'SPONSOR' then
        &spnsrvar = left(substr(line, 8));
     else if upcase(line) =: 'PROJECT' then
        &projvar  = left(substr(line, 8));
   <etc., until creating macro varibales
    for all standard header fields>
     else delete;
data _sashd1;
      set _sashd1 end=lastobs;
      if lastobs;
proc append base=&outds data=_sashd1
      force;
```

7. Create data sets to analyze. First, the data set containing information on SAS files is created. Error codes are created in cases where, e.g., the SAS program has no log.

```
data library.files;
      length prgright $8;
      merge sasfiles(in=a rename=(fn=sasfn
               fdttm=sasdttm fsz=sassz))
            logfiles(rename=(fn=logfn
               fdttm=logdttm fsz=logsz))
            lstfiles(rename=(fn=lstfn
               fdttm=lstdttm fsz=lstsz));
      by fnnoext;
      if not a then do;
            sashdspn = 'NoSASF';
            sashdprj = 'NoSASF';
            sashdpgm = 'NoSASF';
            sashdaut = 'NoSASF';
            sashdver = 'NoSASF';
            sashddt_  = '.A';
            prgright  = 'NoSASF';
      end;
** CHECK WHETHER THE PROGRAM NAME IN THE
** HEADER MATCHES WITH THE FILE NAME;
      if sasfn ne '' then do;
         if sashdpgm ='' then prgright =
         '';
         else if index(sasfn,
         upcase(compress(sashdpgm))) then
           prgright='NameR';
         else if not index(sasfn,
         upcase(compress(sashdpgm))) then
            prgright='NameW';
       end;
       format sashdspn  sashdprj sashdpgm
              sashdaut sashdver prgright
              $nofield. sashddt_ putdate.;
run;
```

Then, using this data set, another data set is created containing the discrepancies between .SAS, .LOG, and .LST files, mainly different dates and times and locations. Macro *%missing* simply checks for missing values of a variable.

```
data probl;
    set library.files(keep=sasfn sasdttm
        sassz logfn logdttm logsz lstfn
        lstdttm lstsz);
    length problem $ 9;
```

```
    format problem $problem.;
    if sasfn = '' then do;
       if logfn = '' then do;
          problem = 'NOSASLOG';
          output;
       end;
       else do;
          problem = 'NOSAS';
          output;
       end;
     end;
** IF SAS FILE IS A MACRO OR INCLUDE FILE, IT
** IS NOT SUPPOSED TO HAVE A  LOG;
     else if sasfn ne '' and not
     (index(sasfn, '\MACRO') or
      index(sasfn, '\SASMACRO') or
       index(sasfn, '\INCLUDE')) and
      logfn = '' then do;
        problem = 'NOLOG';
        output;
      end;
      if not %missing(sasdttm) and not
      %missing(logdttm) and sasdttm >
      logdttm  then do;
         problem = 'LOGOLD';
         output;
      end;
      if not %missing(sasdttm) and not
      %missing(lstdttm) and sasdttm >
      lstdttm then do;
         problem = 'LSTOLD';
         output;
      end;
run;
** IF BOTH LOG AND LIST OLDER, WE HAVE TWO
** OBSERVATIONS, CONVERT TO ONE;
proc sort data=probl;
    by sasfn;
data library.fileprb;
    set probl;
    by sasfn;
    if sasfn = '' then output;
    else do;
       if first.sasfn and last.sasfn then
       output;
       else if first.sasfn and not
       last.sasfn then do;
           problem = 'LOGLSTOLD';
           output;
        end;
    end;
    format problem $problem.;
run;
```

With this information about permanent data sets used in a SAS run, two more data sets are created: one lists the number of updated and non-updated permanent data sets in a SAS run, and another lists the names of non-updated data sets for a particular SAS run. Also, a number of other data sets could be constructed reflecting, e.g., changes in project programs and table structure, in the permanent data set names used in the programs, etc.

Finally, we create an integrated data set combining information from all the previously created permanent data sets.

CONCLUSION

The program and macros introduced in this paper can be used, with some appropriate changes, as a utility program for various large-scale projects involving SAS programs run under Windows NT. Also, parts of the programs can be applied to perform such tasks as producing, in SAS, listings of files in certain directories, permanent data sets used in a SAS run, statistics on the project, etc. Additional features can be added to facilitate analysis of, for example, program and output quality, such as detecting possible 'dangerous' notes, warnings, and error messages in SAS logs, non-labeled or different-labeled variables throughout the project programs, etc. All this can be done by using the above mentioned approach of reading standard keywords and syntax groups in a SAS log. These possibilities prove that the SAS language is not only a powerful tool for data analysis and statistical procedures, but a convenient means of developing useful utility and project management instruments.

CONTACT INFORMATION

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

Keith Gregg / Yefim Gershteyn
SCIREX Corporation
255 East Lake St.,
Bloomingdale, IL 60108
Phohe: (630) 307-1112
Fax: (630) 924-0402
Email: kgregg@scirex.com /
ygershteyn@scirex.com

TRADEMARKS