Paper 82-25

Dynamic data set selection and project management using SAS 6.12® and the Windows NT 4.0 file system

Matt Downs and Heidi Christ-Schmidt
Statistics Collaborative, Inc., Washington, D.C.

ABSTRACT

A common task of statistical coordinating centers is the production of interim reports to a Data and Safety Monitoring Board (DSMB) in order to present the status of an ongoing clinical trial. Frequently, the data summarized in these reports arise from multiple sources and come at varying schedules. Managing the receipt of data for complex trials can require a large organizational effort on the part of the coordinating center to ensure that each report uses the most current data.

In this paper we present a system of dynamic data set selection and project management that uses SAS 6.12 and Windows NT 4.0 file system. In this system, programs used in the production of DSMB reports include at their beginnings a central utility program. This program automatically assigns LIBNAMEs to the most recent source data and provides a macro that checks whether we have updated analysis files with the most recent source data, thereby alleviating us of the tedious task of manually updating every analysis program. Although the code is somewhat complicated, SAS users at any level can use this program.

THE PROBLEM

During the course of many randomized clinical trials, the independent statistical coordinating center frequently is called upon to review data quality and provide unblinded analyses of study data to a Data and Safety Monitoring Board (DSMB) chartered to review the safety and efficacy of the study agent. Often, the DSMB reviews frequent small reports that present updated accrual and abbreviated safety data supplemented at periodic intervals during the study by more extensive reports of safety that may also include analyses of efficacy, data integrity, or treatment compliance.

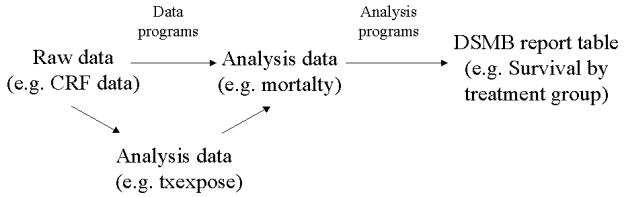
Throughout the trial data often come from different sources; for example, the trial's data management center may periodically send updated case report form (CRF) data that it has entered into the clinical database. Because of regulatory requirements, the sponsor may maintain a serious adverse event (SAE) database that is more current at any point during the trial than the SAE data stored in the CRF database. Furthermore, if the trial uses an adaptive randomization method to allocate patients, a randomization center may send an updated data set containing patient identifiers of randomized patients and their treatment assignments. Other potential players who may transmit data directly to the statistical coordinating center during a study include central laboratories that analyze safety parameters or plasma levels of the study agent as well as central reading centers or endpoint review committees that interpret subjective efficacy data.

Because so many different parties can be responsible for portions of the study data and because of the often short turn-around between data receipt and report production, the statistical coordinating center has to take great pains to tame the many-headed hydra of data management and ensure that its project team always uses the most current data in generating reports to the DSMB.

A SOLUTION TO ENSURE THAT DATA PROGRAMS USE THE LATEST SOURCE DATA

Often the analysis programs for DSMB reports do not directly use the raw data received from the various sources. Instead data programs written in SAS create and recode variables and restructure the data into analysis files, which are more suitable for analysis in SAS (Figure 1). Like the raw data, however, the study may have many analysis files, particularly if the study designers have cast a wide net in the scope of data the study collects.

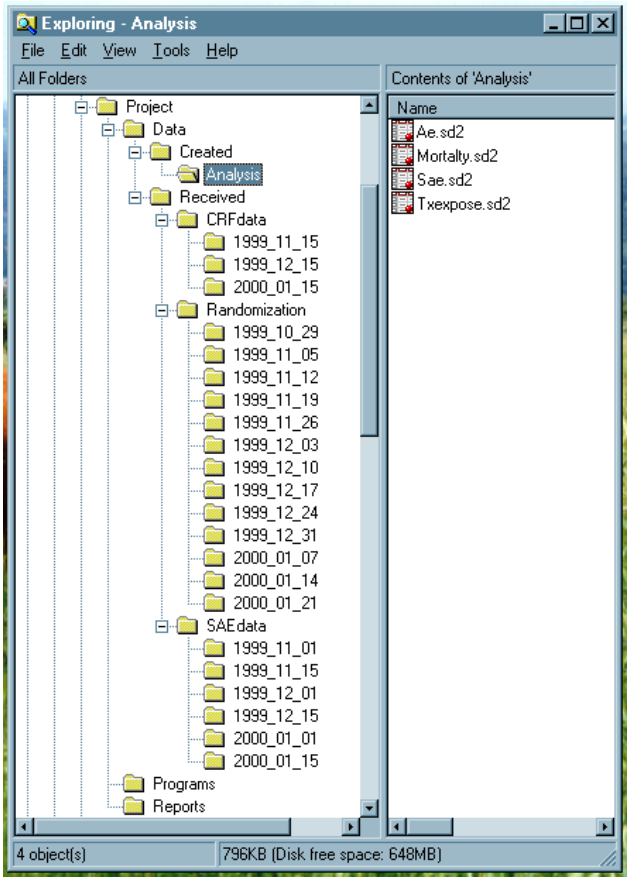
Figure 1. Path for creating analysis datasets



If each data program used for DSMB reports were to include LIBNAME statements that direct the program to the latest source data, the LIBNAME statements would require manual updates with each receipt of new data, a cumbersome task certainly easy to overlook in the rush of new report production. To avoid this, we have created a system in the management of our projects where each data program invokes at its beginning a central utility program. Aside from housing general-use macros for the project's programs and defining global study-specific macro variables, the utility program searches our received data directories for the most recent transfer of each data type and then dynamically assigns LIBNAMEs for use in the creation of updated analysis files.

To allow SAS this functionality in its hunt using the Windows NT 4.0 file system (NTFS), we must adhere to conventions in our organization of received data for projects. Figure 2 shows a prototype project directory on our file server. Notice that the E:\PROJECT\DATA\RECEIVED directory contains subdirectories for each data type we will receive during the course of the study. Within each data type subdirectory, we have created dated subdirectories named by the cut-off dates for the data transfers using a YYYY_MM_DD convention. For example, if we receive an updated randomization file from the randomization center, who informs us that the patient identifiers and treatment assignments it has provided are current as of January 21, 2000, we would copy the received data set to the directory E:\PROJECT\DATA\RECEIVED\RANDOMIZATION\2000_01_21.

Figure 2. Sample project folder on the network



Provided that we strictly adhere to this convention in our placement on the network of received data sets for the project, the utility program is able to extract the contents of each data type subdirectory to a SAS data set, convert the nested subdirectory names to data cut-off SAS dates, and, after sorting the dates, choose the latest data cut-off date for assignment in a LIBNAME statement. As shown in Figure 3, the link between the Windows NT directory command and the SAS data set is accomplished by creating an unnamed pipe with a dir call to the operating system.

Figure 3. Code to extract the subdirectory that contains the latest CRF data sets

```
filename CRFDATA pipe "dir
e:\project\data\received\CRFDATA";

data LISTCRF; * Output directory of CRF data
                transfers to a data set;

  infile CRFDATA missover;
  length DATE TIME SIZE FILENAME $20;
  input  DATE TIME SIZE FILENAME;

  if SIZE = "<DIR>" and
  /* Must be a directory and not a file */
  length(compress(FILENAME)) = 10 and
  /* Directory name must be 10 characters */
  (1999<=input(scan(FILENAME, 1, "_"), ?? 4.0)<=2003);
  * Must be a directory that follows our naming
    convention. That is, the first four digits
    of the directory represent the last year for
    which the file contains data. We have
    presumed that we will not receive data sets
    for the study after 2003.;

  CRFDATE = mdy(scan(FILENAME, 2, "_"),
                scan(FILENAME, 3, "_"),
                scan(FILENAME, 1, "_"));
  * Convert the directory name to a SAS date;

  if   CRFDATE <= .Z
  then do;

       %beep(5, FRSTDUR = 80, SECDUR = 80);
       * Macro defined earlier in central utility
         program. Just gives audible warning to
         user that there is a problem.;
       put / "ERROR: CRF directory has incorrectly
       named subdirectory: " FILENAME +(-1) "." /;

       end;
run;

proc sort data = LISTCRF;
  by CRFDATE; * Sort data set so latest data set
                directory appears last;
run;

data _null_; * Place directory name of latest CRF
               data set to the macro variable
               CRFDIR.;
             * Convert the directory name to the data
               cut-off date and place in the macro
               variable CRFCUT.;

  set LISTCRF end = LASTREC;
  if   LASTREC
  then do;

       call symput ("CRFDIR", compress(FILENAME));
       call symput ("CRFCUT", put(CRFDATE, mmddyy10.));
       put // "NOTE: The latest CRF data was received
       from the data management center on " CRFDATE
       :weekdate30. +(-1) "." //;

       end;
run;

libname CRFDATA
        "e:\project\data\received\CRFDATA\&CRFDIR";

* Directory for most recent CRF data sets received from
  the data management center;
```

Assuming, for example, that the directory tree for our project looks as it does in Figure 2, after running the code in Figure 3, the CRFDATA libname will be assigned to E:\project\data\received\CRFDATA\2000_01_15. As we place new CRF data on our server, executing the code will automatically reassign the libname to the latest dated directory containing the new data.

Note that the order of the input variables in the LISTCRF data step in Figure 3 is specific to the NTFS. When using this code on Windows 95/98 machines, we have found that the order of information output by the dir command differs from NT's, requiring us to reorder the input variables on these systems.

A SOLUTION TO ENSURE THAT DATA PROGRAMS HAVE, IN FACT, BEEN RUN

Although helpful, the code presented in Figure 3 is insufficient to guarantee that reports to the DSMB use the latest data. While ensuring that the data programs use the latest data when they are run, the code does nothing to safeguard that the project team has, in fact, run the data programs to create new analysis files. To achieve this, we include in the utility program for a project the declaration for DPNDCHEK, a macro that our programmers invoke whenever a data or analysis program first uses an analysis file we have created. DPNDCHEK checks the creation date of an analysis file and confirms that it is not before the date we placed the source data sets used as inputs for the analysis file onto our server.

The macro relies on a project data set, called DPNDCHEK.sd2, that lists the dependencies between the analysis files that we create and the source data. DPNDCHEK.sd2 is structured with one record per source data set for each analysis file and contains the three variables:

- SRCELIB, which specifies the library for the source data set (For example, if the source data is from a CRF data set, then we set SRCELIB to “CRFDATA”. Likewise, if the source data is from a previously created analysis file, we set SRCELIB to “ANALYSIS”),
- SRCEDATA, which specifies the name of the source data set, and
- DPNDDATA, which names the analysis file that is dependent upon the source data set specified by SRCELIB and SRCEDATA.

As an example, if the data program that creates the analysis file MORTALTY relies upon the CRF data set DEATH and the previously created analysis files named TXEXPOSE and SAE, DPNDCHEK.sd2 will contain the first three records as shown in Figure 4. The figure shows that the analysis file SAE also uses the analysis file TXEXPOSE as an input dataset, which, in turn, uses the DRUGADM CRF dataset.

Figure 4. PROC PRINT of DPNDCHEK.sd2 that shows the source data sets for the analysis files MORTALTY.sd2, SAE.sd2, and TXEXPOSE.sd2

OBS	dependent (i.e. analysis) data set (DPNDDATA)	Library for source data set (SRCELIB)	Source data set (SRCEDATA)
1	MORTALTY	CRFDATA	DEATH
2	MORTALTY	ANALYSIS	TXEXPOSE
3	MORTALTY	ANALYSIS	SAE
4	SAE	ANALYSIS	TXEXPOSE
5	TXEXPOSE	CRFDATA	DRUGADM

When invoking the macro in their programs, programmers must specify the name of the analysis file for which they wish to check the dependencies. For each source data set where the variable DPNDDATA matches the name of the analysis file specified in the macro call, the macro then either:

1) performs a call to the operating system, if we receive the source data externally, to confirm that the NTFS creation date of the source data set's directory is no later than the creation date of the dependent analysis file, or,
2) if the source data set is another analysis file created internally, checks through PROC DATASETS that the creation date of the source analysis file is on or before the creation date of the dependent analysis file.

We check the creation date of our server directories that contain external source datasets, rather than checking the creation dates of the files themselves, because we want to extract the date and time the files were placed on our server and not when the outside source created the data set.

Furthermore, if sources of an analysis file are themselves analysis files, the macro also checks that the source analysis files are up-to-date by calling itself, thereby ensuring that the project team has created analysis files in the proper order accounting for any dependencies between the data sets. If any source data set is newer than the analysis file, the macro returns an error message to the log, informing programmers that they must rerun the data program that creates the analysis file. Figure 5 shows the general structure of the DPNDCHEK macro. Note that sections denoted by “CODE REMOVED” were omitted from the presentation to conserve space. Also be aware that the formats of dates and times in directory calls are user-dependent, configured through NT’s “Regional Settings” control panel. This code assumes that times are in 24-hour format and that dates are in mm/dd/yy format.

Figure 5. The DPNDCHEK macro

```
*****> The DPNDCHEK macro should be invoked whenever
data and analysis programs use an analysis file that we
have created. Using the SAS data set DPNDCHEK.sd2,
which contains for each created data set a list of the
dependent data sets, the macro checks the creation date
and time of the analysis file and confirms that it is
after the date and time of each dependent source data
set. If the source files are themselves analysis files
that we create, the macro then calls itself for each
source analysis file to ensure that they need not be
rerun.;

%macro DPNDCHEK(ANALDATA,    /* Analysis file to be
                                checked */
                FRSTCALL = 1 /* Switch that tell macro
                                whether it has been
                                invoked from another
                                DPNDCHEK call */

%* Get a data set that lists the data dependencies for
each analysis data set. Count the number of
dependencies and output the count to macro variable
NUMCHEKS.;

data SOURCES;

  retain DEPENDNT 0; %* Count the number of
                        dependencies for analysis file;

  length SRCEPATH $200;
  set DPNDCHEK end = LASTREC;

  SRCEPATH = pathname(SRCELIB); %* Convert librefs to
                                   pathnames;

  if SRCEPATH = ""
  then do;                    %* Libname was undefined;

       %beep(5);
       put "WARNING: " SRCELIB "libname is undeclared."
           / "Check LIBNAMES and DPNDCHEK.sd2 for
           errors." //;

       end;

if   LASTREC
then do;
```

```
    if   compress(DPNDDATA) = compress(upcase
                                ("&ANALDATA"))
         and SRCEPATH ^=""
    then DEPENDNT + 1;
    call symput ("NUMCHEKS", put(DEPENDNT, 2.0));
    %* Output number of dependencies to macro variable
       NUMCHEKS;
    if DEPENDNT = 0
    then do;

         %BEEP(20);
         put "ERROR: Data set does not exist in the
         list of data dependencies (DPNDCHEK.sd2)" /
         "Check spelling of data set in macro call or
         update DPNDCHEK.sd2.";
         SLEEP = sleep(5);  %* Pause for 5 seconds to
                               allow programmer time
                               to note error.;
         end;
    end;

  if compress(DPNDDATA) = compress(upcase("&ANALDATA"))
     and SRCEPATH ^=""
  then do;

       DEPENDNT + (not(LASTREC));
       output;

       end;

  drop SLEEP;
run;

%if &NUMCHEKS > 0   /* Only continue with the macro if
                       at least one source dataset is
                       defined in DPNDCHEK dataset   */
%then %do;          /* for the dataset being checked.*/

      %* Get the creation date of the dependent dataset
         and assign it to the macro variable DPNDDATE;

      proc datasets   library = ANALYSIS
                      nodetails nolist;
        contents    data    = &ANALDATA
                    out     = WORK.MODATE noprint;
        %* MODATE contains the modified date extracted
           from the PROC CONTENTS as a variable also
           called MODATE. It is a DATETIME value.;
      run;
      quit;

      data _null_;
        set MODATE  (keep = MODATE)
            end = LASTOBS;

        if   LASTOBS
        then do;

             put / "NOTE: ANALYSIS.&ANALDATA was last
             modified on " MODATE :datetime40. //;
             call symput("DPNDDATE", compress(MODATE));

             end;
      run;
```

```
%* Cycle through the source data sets for the
analysis file and compare their creation dates
with the modification dates for the analysis file;

%do INDEX = 1 %to &NUMCHEKS;

    data _null_;
     set SOURCES (where = (DEPENDNT = &INDEX));

     call symput ("SRCELIB",  compress(SRCELIB));
     call symput ("SRCEDATA", compress(SRCEDATA));
    run;

    %* CODE REMOVED. We removed code here because
    we have presented similar code elsewhere in
    the paper. If the data source is externally
    created, the macro performs a NTFS call using
    an unnamed pipe, similar to that in Figure 3,
    to extract the creation date and time of the
    directory that contains the source data. The
    macro then used the DHMS function to create
    the datetime value MODATE in the dataset
    SRCEDATA with one record for the source
    dataset. Likewise, if the source dataset is
    created internally (i.e., it also is an
    analysis file created in-house), the macro
    performs a PROC DATASETS to extract MODATE to
    the dataset SRCEDATA.;

    data _null_;
      set SRCEDATA end = LASTREC;
      if   LASTREC
      then do;
           if   .Z < MODATE < &DPNDDATE
           then put "NOTE: &SRCELIB..&SRCEDATA
                created before ANALYSIS.&ANALDATA
                on " MODATE :DATETIME40.;
           else if MODATE >= &DPNDDATE
           then    do;
                   %beep(15);
                   put "ERROR: ANALYSIS.&ANALDATA
                   needs to be rerun." /
                   "&SRCELIB..&SRCEDATA created on
                   " MODATE :DATETIME40.;
                   SLEEP = sleep(5);
                   %* Pause for 5 seconds to allow
                   programmer adequate time to see
                   error.;
                   end;
           end;
    run;

    %* CODE REMOVED. Proc datasets to delete
       temporary datasets.;
%end;

%* Finally, the macro needs to call itself for
source data sets that are not primary, that is,
for source data sets that are themselves analysis
files. Only execute this code for the first
invocation of DPNDCHEK (FRSTCALL = 1);
```

```
%if   &FRSTCALL = 1
%then %do;

      %let ANALDATA = "&ANALDATA";

      %do %until (&MORE = 0);

          data  TEMP;
            length  NEWANAL $200;
            retain  MORE    0
                    NEWANAL;
            set     DPNDCHEK  end = LASTREC;

            if   compress(DPNDDATA) in (&ANALDATA)
                 and SRCELIB = "ANALYSIS"
            then  do;

                  if   MORE = 0
                  then NEWANAL = '"' ||
                               compress(SRCEDATA)
                               ||'"';
                  else NEWANAL = trim(NEWANAL) ||
                                 ', "'||
                               compress(SRCEDATA)
                               || '"';
                  output;
                  MORE + 1;

                  end;

            if    LASTREC
            then  do;

                  call symput ("MORE",
                       compress(put(MORE, 3.0)));
                  call symput ("ANALDATA",
                       trim(NEWANAL));

                  end;

            keep    SRCEDATA;
            rename  SRCEDATA = ANALDATA;

          run;

          proc append out  = AUTOCALL
                      data = TEMP force;
          run;

      %end;

      proc sort data = AUTOCALL nodupkey;
        by ANALDATA;
      run;

      data  _null_;
        set  AUTOCALL end = LASTREC;

        if   LASTREC
        then call symput("AUTOCALL",
                         compress(_N_));

      run;

      %do  INDEX2 = 1 %to &AUTOCALL;

            data _null_;
              set AUTOCALL;

              if   _N_ = &INDEX2
                then call symput ("NEWANAL",
                                  compress(ANALDATA));
            run;

            %DPNDCHEK(&NEWANAL, FRSTCALL = 0);
      %end;

      %* CODE REMOVED: PROC DATASETS to delete
         AUTOCALL;

  %end;

%end;

%mend DPNDCHEK;
```

As an example, consider that DPNDCHEK.sd2 has the structure shown in Figure 4. If we invoke the DPNDCHEK macro on the MORTALTY analysis dataset, the macro will check that before running the MORTALTY data program that we had 1) placed the CRF dataset DEATH on the server and 2) run the data programs that create the SAE and TXEXPOSE analysis files. In addition, to ensure that we had run TXEXPOSE and, by extension, SAE using the latest CRF data and, further, that we had run the SAE and TXEXPOSE data programs in the correct order, the DPNDCHEK macro would call itself for both analysis files.

TRADEMARKS

CONTACT INFORMATION

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

Matt Downs and Heidi Christ-Schmidt
Statistics Collaborative, Inc.
1710 Rhode Island Ave., NW
Suite 200
Washington, DC 20036
Work Phone: (202) 429-9267
Fax: (202) 429-9267
E-mail: matt@statcollab.com
E-mail: heidi@statcollab.com
Web: www.statcollab.com