Paper 106-28

Logicals From Libraries: Using Storage as a Bridge Between Sessions

Gary E. Schlegelmilch, U.S. Dept. of Commerce, Bureau of the Census, Suitland MD

ABSTRACT

In converting legacy systems into SAS, or consolidating systems into an integrated whole, new techniques are often developed just to accommodate the transition. In a case where a UNIX application needed to communicate file locations from inside the Base SAS program back to the calling script file, we used the common factor – physical storage – as the common link. This allowed the parent process to retrieve values for variables that were defined in subprocesses, and normally inaccessible by the parent process.

The technique uses several SAS functions, in common use in many SAS programs, to access the data from the calling process. The program then writes the library information to a UNIX script file, which can in turn be executed by the calling process to define the logicals at that level. Although this problem was solved for a UNIX platform; the technique used would lend itself to other platforms as well.

INTRODUCTION

The Bureau of the Census is currently involved in converting most of its Economic surveys into a central driving system. This system, the Standard Economic Processing System (StEPS), will save a tremendous amount of development and processing time by using common library routines. One of its features, designed to make the system more flexible, is a parameter dataset which stores the physical locations of data file directories. Those physical locations are expressed as macro variables and defined libraries, which in turn are passed by StEPS to its member surveys.

However, transitions and conversions to any new processing system are often complex. The users and analysts wish to lose as little functionality as possible, while taking advantage of faster processing time and less maintenance. In our particular case, an older system for reviewing and correcting Survey of Construction data used hard-coded file and directory names. We needed to transition the hard-coded names in both the on-line and batch processing to system logicals that would mirror the library names used for dataset and data file storage. But under UNIX, the session and sub-session structure made it difficult to return the values of those libraries and logicals to the calling programs.

The common ground was, in this case, the hard disk. By using it as a bridge, we could build all the existing library names we needed right into a script, and run that in the top-level session to establish our logicals for all subordinate sessions.

THE COMMAND FILE

Based on a date and a survey name, StEPS uses its proprietary processes to establish all the libraries needed for an on-line session. When processing interactively, these are entered by the user; in batch, we will have to provide them as arguments to the script.

In UNIX, system-level commands for batch processing are entered into a script file. This script needs to accomplish four things: (1) accept the user parameters (in this case, the name of the survey to be processed and a YYYYMM-formatted survey date, STATP00), (2) define some local working directories for the survey as STEPSDAT and STEPSCNT, (3) execute the SAS program that will establish the libraries and write that information to a script file, and (4) execute the newly-created script with the '.' (dot) command, so that it runs within the parent process.

SAS CODE TO CREATE THE SCRIPT

The SAS program, called SET_STEPS_LOGICALS.SAS, looks like this:

```
/*-----------------------------------
  SET_STEPS_LOGICALS.SAS

  A short SAS program that will use the
  libraries established by %setlibs,
  and build a script to create the
  environment variables required by the
  UNIX parent process.
-----------------------------------*/

/*  Pick up the environment variables
    defined in the calling script.   */

data _null_;
  length SURVEY $ 5 STATP00 $ 6
         USER $ 8;
  SURVEY=sysget('SURVEY');
  call symput('survey',SURVEY);
  STATP00=sysget('STATP00');
```

```
  call symput('statp00',STATP00);
  USER=sysget('USER');
  call symput('user',USER);
  STEPSDAT=sysget('STEPSDAT');
  call symput('stepsdat',STEPSDAT);
run;

/* The STEPSCNT macro variable must be
   defined with no trailing spaces.
*/

data _null_;
  x=sysget('STEPSCNT');
  ln=length(x);
  call symput('ln',ln);
run;
data _null_;
  length stepscnt $ &ln;
  stepscnt=sysget('STEPSCNT');
  call symput('stepscnt',stepscnt);
run;

/* The ALL_LIBS and SET_LIBS routines
   define all the libraries that will
   be used in this session.  The
   source is in the STEPS.SAS file.
*/

%include "&stepscnt/steps.sas";

%all_libs(survey=&survey,statp=&statp00
);

/* Since no documented routine provides
   the STATP01, et al, in logical form,
   this little routine retrieves them.
*/

data _null_;
  length statp01 statp02 $ 6;
  x=pathname('data01');
  ln=length(x);
  ln2=ln-5;
  statp01=substr(x,ln2);
  call symput('statp01',statp01);
  x=pathname('data02');
  ln=length(x);
  ln2=ln-5;
  statp02=substr(x,ln2);
  call symput('statp02',statp02);
run;

%setlibs(survey=&survey,statp00=&statp0
0,statp01=&statp01,statp02=&statp02);

/* Now build a script file which will
in
   turn be run by the originating
script
   file. */

data _null_;
  x=pathname('usertemp');
  x1=trimn(x); /* if blank, this field
will return a null string */
  call symput('usertemp',x1);
run;

/* Since the survey name might have
   imbedded blanks, this will eliminate
   them.  COMPRESS and TRIM build a
   resulting field that may have
   trailing blanks, which could make
   the file inaccessible to UNIX.
   This approach is taken to
   eliminate the possibility.    */

data _null_;
  filenm=compress
("&stepsdat/scripts/temp_steps.scr");
  filelen=length(filenm);  /* gives the
length of the string without blanks */
  call symput('filelen',filelen);
run;

data _null_;
  length filenm $ &filelen;
  filenm=compress
("&stepsdat/scripts/temp_steps.scr");
  call symput('filenm',filenm);
run;
filename OUTSCR "&filenm";

/* Now create the output, a script file
   which will set the StEPS variables
up
   for the UNIX environment.
*/

data _null_;
  file OUTSCR;
  x=pathname('datalib');
  put "export DATALIB=" x;
  put "export DATA00=&data00";
  put "export DATA01=&data01";
  put "export DATA02=&data02";
  x=pathname('userlib');
  put "export USERLIB=" x;
  x=pathname('central');
  put "export CENTRAL=" x;
  x=trim("&usertemp");
  put "export USERTEMP=" x;
  put "export STATP01=&statp01";
  put "export STATP02=&statp02";
run;
```

At the end of the SAS program, control returns to the calling script file, STEPSLIBS.SCR. (Local policy recommends that these files be coded with a ".scr" extension.) The script file that was built by the SAS program, TEMP_STEPS.SCR, is run with a "." in front of the script file name. In UNIX, a batch file is run in a subordinate or "child" session from the parent session, and as such, the logicals established would not be accessible by the parent session. By using the ".", it instructs the UNIX processor to run those commands within the parent session, making the logicals established accessible to all subordinate sessions.

For those familiar with VAX architecture, it would be the difference between running a command with a "@" command and a SUBMIT command. The "@" runs the command file within the current session, where the SUBMIT establishes a new session. In fact, this routine could easily be tailored to run in a

VAX environment, by changing the EXPORT commands to establish global VAX logicals.

RESULTS

Running this process with the parameters set to SOC as the input survey name and the survey date of 199904 (yyyymm format) would define STEPSDAT (the base directory for the survey) as /steps/$survey, or /steps/soc.

That information would be used by the StEPS library macro routines, and would create all necessary libraries for the session. As an example, DATA00 (the current library of data for the entered survey date) would be defined as /steps/soc/d199904. The SAS program would then retrieve that data, and build a line in the output script file:

```
export DATA00='/steps/soc/d199904'
```

Then, when the output script was run, any external program could access those datasets by defining the library as $DATA00.

CONCLUSIONS

The versatility of SAS is unbounded. But even the most powerful languages and applications cannot override the limitations of a hierarchical operating system. However, by taking a little time to examine all the tools in the SAS tool box - there is usually a way to get even the oddest jobs done.

REFERENCES

Aster, Rick, Professional SAS Programmer's Pocket Reference, 3nd Edition, 2000, Breakfast Books.
Gilly, Daniel, UNIX in a Nutshell, O'Reilly & Associates, 1998.

ACKNOWLEDGEMENTS

My thanks go to Sam Rozenel and Anne Linonis of ESMPD for their support in the writing of this paper.

CONTACT INFORMATION

Gary E. Schlegelmilch
U.S. Dept. of Commerce, Bureau of the Census, ESMPD/MCDIB
Suitland Federal Center, Rm. 1200-4
Washington DC 20233-6200
Email Gary.E.Schlegelmilch@census.gov