

Using Code that Writes Code to Generate External Flat Files

Jane Mudano, Pharmaceutical Product Development, Inc., Morrisville, NC

ABSTRACT

The creation of external flat files from existing SAS® data libraries can be simplified by instructing the SAS System to write the necessary code for you. If external flat file structure specifications (desired file name, column placement, and variable formats) are available in, or can be converted to, ASCII format, a SAS program can read in the specifications and construct a second SAS program containing the statements for creating the external files.

This method of data conversion reduces the possibility of coding errors when assigning column placement or variable formats. It also eliminates the need for code modification in response to changes in file structure specifications.

The data conversion process may be validated by using the same specifications file to create a companion program which writes SAS code containing the appropriate INPUT statements for reading the flat files and recreating the original data library. The original and the recreated SAS data libraries may then be compared to ensure that the algorithms employed in the data conversion process are error-free.

INTRODUCTION

The concept of writing code that generates code is not a new one. Applications based on the dynamic creation of SAS code have been developed to generate external flat files (e.g., ASCII) from SAS databases (Harris, M.C., 1995, Woolridge, G.M. and Winsor, H.B., 1994, Whitlock, H.I., 1993, Patterson, K., 1992). Such applications rely on some external source of information, such as a PROC CONTENTS output, to provide the necessary flat file layout information.

At Pharmaceutical Product Development, Inc. (PPD), we maintain clinical trials data predominantly in SAS libraries. Many of our clients who do not use SAS require their data to be converted to flat files for loading into their internal data management system. We routinely create ASCII files from our SAS databases according to client-specific file layout requirements. Typically, the clinical trials databases consist of many SAS data sets (we average

approximately 20 per study), some of which may contain up to one hundred variables. The number of ASCII files to be created is usually comparable to the number of data sets in the SAS database, and is frequently larger.

The SAS code required to convert these data to flat files can be quite extensive when there are many large files to be created. Although they may be straightforward, writing the PUT statements can be tedious and extremely time consuming. This, in addition to the potential for errors in assigning column placement or variable formats in the PUT statements, argues in support of an automated method for creating data conversion code. Such a method also ensures that minimal modifications to the data conversion code are needed when the file layout specifications change. For those situations where the specifications change frequently, this method can produce significant savings in time.

To guarantee that the integrity of the data is preserved, the ASCII files should be validated as a standard part of the data conversion process. At PPD, we commonly accomplish this by recreating the original SAS data sets from the appropriate ASCII file(s) and comparing the original and recreated data sets to detect differences using PROC COMPARE. The same file layout information used to drive the code generation for producing ASCII files is used as the basis for writing code to reconstruct the SAS data sets. In general, our approach to the data conversion process consists of the following steps:

1. Archive SAS database.
2. Input file layout specifications.
3. Generate SAS code.
4. Create ASCII files.
5. Reconstruct SAS database.
6. Compare original (archived) and reconstructed SAS data sets.
7. Examine compare output and correct any programming errors.

AN EXAMPLE

This section describes a typical situation in which we use code-generating-code to perform data conversion and validation.

We received electronic file layout specifications from our client in ASCII format listing the names of the flat files to be created, variable names, column placement, and variable format (Figure 1). We constructed our internal SAS database with the same variable names as those used by the client to facilitate data management and transfer. We began the conversion process by examining the specifications to identify variable formats that could not be interpreted by SAS (e.g., 0999, A3). Our code-generating-code program, MAKEPGMS.SAS (Figure 2), was designed to read the specifications file and to:

- a) construct the appropriate PUT statements for creating the ASCII files (Figure 3), and
- b) construct the appropriate INPUT statements for reading the ASCII files and recreate the SAS data sets (Figure 4).

The program incorporated code designed to translate the "non-SAS" formats found in the specifications into ones which SAS can recognize. As a result, no editing of the ASCII file creation program was needed before it was used.

It was necessary to archive a copy of the database to preserve it for comparison. Because data management and data transfer processes frequently occur simultaneously and the working database might still have been undergoing updates, the reconstructed data sets were compared to this archived "snapshot" of the original data sets. A sample compare program is shown in Figure 5.

To further facilitate the data conversion process, the entire suite of data conversion and validation programs could be packaged into one larger "driver" program which calls, via %INCLUDE, the creation, reconstruction, and validation programs. This would allow libnames and filenames to be defined in the main driver program, rather than in each component program as shown here, thus simplifying program maintenance.

CONCLUSION

The code presented here demonstrates one way to use the power of SAS to reduce programming time, minimize transformation error, and facilitate modification of the applications for electronic data interchange. Code that writes code produces a reliable, consistent, and efficient approach to this task.

REFERENCES

- Harris, Michael C. (1995), "Programs That Write Programs: Creating Code Generators in SAS," Presented at the 1995 PharmaSUG Conference, February 1995, St. Petersburg Beach, FL.
- Patterson, Karen. (1992), "Creating Comma-Delimited Files from SAS Data Sets for Downloading to Microcomputers," Proceedings of the Seventeenth Annual SAS Users Group International, Cary, NC: SAS Institute Inc., 1070-1072.
- Whitlock, H. Ian. (1993), "A Macro to Make External Flat Files," Proceedings of the Eighteenth Annual SAS Users Group International, Cary, NC: SAS Institute Inc., 258-263.
- Woolridge, Greg M. and Winsor, Henry B. (1994), "A Use of External Files in Dynamic Code Generation," Proceedings of the Nineteenth Annual SAS Users Group International, Cary, NC: SAS Institute Inc., 125-129.

ACKNOWLEDGEMENTS

The author thanks Bernadette Johnson, SAS Institute Inc., for her input in developing the application presented in this paper. Thanks also go to PPD staff members Rita Thissen and Marcia Murto for their editorial assistance.

SAS is a registered trademark or trademark of SAS Institute Inc. in the USA and other countries.
® indicates USA registration.

AUTHOR

The author may be contacted at:

Jane Mudano
Pharmaceutical Product Development, Inc.
1400 Perimeter Park Drive, Suite 100
Morrisville, NC 27560
(919) 380-2000
INTERNET: mudanoje%ppdrtp@mcimail.com

Figure 1. Sample File Layout Specifications.

DATABASENAME=SESUGDB				PROTOCOL=SESUG_EX	06/15/95 13:05:44
DRUGDISP	0001	INVNO	5	99999	INVESTIGATOR NO.
DRUGDISP	0006	PTNO	5	99999	PATIENT NO.
DRUGDISP	0011	PTINIT	3	A3	PATIENT INITIALS
DRUGDISP	0014	DGDATE	8	DATE'MM/DD/YY'	DATE DRUG DISPENSED
DRUGDISP	0022	DGVISNO	4	99.9	VISIT NO.
DRUGDISP	0026	DGLSTD	8	DATE'MM/DD/YY'	DATE LAST DOSE TAKEN
VITALS	0001	INVNO	5	99999	INVESTIGATOR NO.
VITALS	0006	PTNO	5	99999	PATIENT NO.
VITALS	0011	PTINIT	3	A3	PATIENT INITIALS
VITALS	0014	VIDT	8	DATE'MM/DD/YY'	VITAL SIGNS DATE
VITALS	0022	VITM	4	0999	VITAL SIGNS TIME
VITALS	0026	VIVISNUM	4	99.9	VITAL SIGNS VISIT NO.
VITALS	0030	VIBPS	5	999.9	BP SYSTOLIC
VITALS	0035	VIBPD	5	999.9	BP DIASTOLIC
VITALS	0040	VIPLS	3	999	PULSE
VITALS	0043	VIRESP	3	999	RESPIRATION RATE

Figure 2. MAKEPGMS.SAS

```

/*$START$*****
/*
/* PROGRAM: MAKEPGMS.SAS
/* DIRECTORY: sesug:[program.950701]
/* AUTHOR: mudanoje
/* DATE CREATED: 07/01/95
/* PURPOSE: Read the client data transfer specs file
/*           and create SAS programs to convert
/*           the SAS database into ascii files, and
/*           create SAS programs to reconstruct
/*           the SAS database from the ascii files.
/*
/*$STOP$*****

options source source2 mprint;

/* specify the number of observations from the original SAS
data to use when testing your code;
%let numobs=;

/* client data transmission specifications file;
%let specfile=sesug:[data.in]specfile.dat;

/* directory where original SAS database is stored;
%let origdir=sesug:[data.dm];

/* directory where original database will be archived;
%let arcdir=sesug:[data.&dirdate..archive];

/* specify dated subdirectories for storing created
programs, ascii files, and SAS data sets;
%let dirdate=950701;

/* directory where the ascii data files will be written;
%let ascidir=sesug:[data.&dirdate];

```

```

/* directory to store reconstructed SAS data sets;
%let recondir=sesug:[data.&dirdate..reconstr];

/* directory to store created SAS -> ascii programs;
%let makeascii=sesug:[program.&dirdate..makeascii];

/* directory to store created ascii -> SAS programs;
%let makesas=sesug:[program.&dirdate..makesas];

/* client specs file;
filename specs "&specfile";

** SAS program to recreate the SAS data sets;
filename makesas "&makesas._makesas.sas";

** clean out the directories where the programs will be
written;
x "delete &makeascii.*.*" /noconfirm";
x "delete &makesas.*.*" /noconfirm";

** archive a copy of the database to use later for comparison;
x "copy &origdir.*.* &arcdir";

data _null_;
* tempname contains the input value of outname;
* prevname contains the prev value of outname;
length outname $200
tempname $8
prevname $8
prefix $1
newform $15 ;

* set value of PREVNAME before first data step iteration;
retain prevname 'XXXXXXXX';

* skip the first 2 records;
infile specs lrecl=120 pad firstobs=3 end=eof;

```

```

* read in the output filename first;
* if different from the last file, complete previous
  file output before starting a new file;
input @1 tempname $8. @@;

* if the tempname and the previous name are not the same AND
it is not the first file processed (prevname^=XXXXXXXX) then
finish up the previous file;

if tempname ^= prevname then do;
  * finish up the last lines in the previous program;
  * if this is the first, it will just put semicolon in the log;
  put ";";

  * create the new filename;
  outname=compress("&MAKEASCII"||trim(tempname)||".SAS");

  * retrieve output filename from the variable OUTNAME;
  file makeascii filevar=outname new ;
  link ascpag;

  * reinitialize file to contain the code to recreate the data
  sets from the ascii file;

  file makesas old;
  link saspage;

  * reset the prevname for next iteration;
  prevname=tempname;

end;

do while (prevname=tempname and not(eof));

  input @11 column
    @16 varname $8.
    @32 format $15.
    @51 desc $char21.;

  * reformat client format specs into SAS format names;
  select;
  when(format="DATE'MM/DD/YY'")
    format='MMDDYY8.';
  when(format="A'")
    format=trim(translate(format,'$','A')) || ' ';
  when(format in: ('0','9')) do;
    if format="0" then do;
      prefix='Z';
    end;
    else prefix="";
    if ("=scan(format,2,' ') then dec=0;
    else dec=length(compress(scan(format,2,' ')));

    int=sum(length(compress(scan(format,1,' '))), (dec > 0), dec );
    format=compress(prefix||put(int,10.)||'.'||put(dec,10.));
  end; * when(format in: ('0','9'));

  otherwise do;
    put "##### OUTPUT ERROR " _all_;
  end;
end; *select;

newform=compress(format,'Z');

file makeascii mod;
put @1 "@ " column 4.-1 varname $9. format $9. @25
  " " " " " desc $21. " " " " ;

```

```

file makesas mod;
put @1 "@ " column 4.-1 varname $9. newform $9.
  @25 " " " " " desc $21. " " " " ;

```

```
if not(eof) then input @1 tempname $8. @@;
```

```

* enter here if we have either reached the end of the   specs
file file OR the data set/file name has changed;
* this code finishes off the previous file before starting
another file;
if (eof) or tempname ne prevname then do;
  file makeascii mod;
  put "#####";
  /run;
  /libname arcdata clear;run;;
  file makesas mod;
  put "#####";
  /run;
  /libname reconstr clear;run;;
end;
end; * do while;

```

```
RETURN;
```

```
ASCIPAGE:
```

```

put
/'**$START$*****'/
/ ' '
/ ' PROGRAM: ' TEMPNAME +(-1) '.SAS'
/ ' DIRECTORY: sesug:[program.&dirdate..makeascii]'
/ ' AUTHOR: mudanoje
/ ' DATE CREATED: &sysdate &systeme"
/ ' PURPOSE: create ASCII files from SAS database '
/ '
/'**$STOP$*****'/
/ 'libname arcdata '&arcdir';
/ 'filename datfile '&ascidir' tempname +(-1) '.DAT';
/ 'proc sort data=arcdata.' TEMPNAME " &numobs
out="NAME ";
/ 'BY INVNO PTNO ;'
/ 'RUN;'
// "data _NULL_;"
/ " set " TEMPNAME ";
/ " file datfile ;"
/ " put "
;;
RETURN;

```

```
SASPAGE:
```

```

put
/'**$START$*****'/
/ ' '
/ ' PROGRAM: ' TEMPNAME +(-1) '.SAS'
/ ' DIRECTORY: sesug:[Program.&dirdate..makesas]'
/ ' AUTHOR: mudanoje
/ ' DATE CREATED: &sysdate &systeme"
/ ' PURPOSE: reconstruct SAS data sets from the
  ASCII files'
/ '
/'**$STOP$*****'/
/ 'libname reconstr '&reconstr';
/ 'filename datfile '&ascidir' tempname +(-1) '.DAT';
/ 'data reconstr.' TEMPNAME ";
/ " infile datfile ;"
/ " input "
;;
RETURN;

```

Figure 3. Sample ASCII File Creation Program.

```

/*$START$-----/
/*
/* PROGRAM: VITALS.SAS
/* DIRECTORY: SESUG:[PROGRAM.950701.makeascii]
/* AUTHOR: mudanoje
/* DATE CREATED: 01JUL95 19:05
/* PURPOSE: create ASCII files from the SAS database
/*
/*$STOP$-----/
libname arcdata 'sesug:[data.archive]';
filename datfile sesug:[data.950701]VITALS.DAT;
proc sort data=arcdata.VITALS out=VITALS ;
BY INVNO PTNO ;
RUN;

data _NULL_;
set VITALS;
file datfile ;
put
@1 INVNO 5.0 /* INVESTIGATOR NO. */
@6 PTNO 5.0 /* PATIENT NO. */
@11 PTINIT $3. /* PATIENT INITIALS */
@14 VIDT MMDDYY8. /* VITAL SIGNS DATE */
@22 VITM 24.0 /* VITAL SIGNS TIME */
@26 VIVISNUM 4.1 /* VITAL SIGNS VISIT NO.*/
@30 VIBPS 5.1 /* BP SYSTOLIC */
@35 VIBPD 5.1 /* BP DIASTOLIC */
@40 VIPLS 3.0 /* PULSE */
;
run;
libname arcdata clear;run;

```

Figure 4. Sample SAS Data Set Reconstruction Program.

```

/*$START$-----/
/*
/* PROGRAM: VITALS.SAS
/* DIRECTORY: SESUG:[PROGRAM.950701.makesas]
/* AUTHOR: mudanoje
/* DATE CREATED: 01JUL95 19:24
/* PURPOSE: reconstruct SAS data sets from the
/*           ASCII files
/*
/*$STOP$-----/
libname reconstr 'sesug:[data.950701.reconstr]';
filename datfile 'sesug:[data.950701]VITALS.DAT';
data reconstr.VITALS ;
infile datfile ;
input
@1 INVNO 5.0 /* INVESTIGATOR NO. */
@6 PTNO 5.0 /* PATIENT NO. */
@11 PTINIT $3. /* PATIENT INITIALS */
@14 VIDT MMDDYY8. /* VITAL SIGNS DATE */
@22 VITM 4.0 /* VITAL SIGNS TIME */
@26 VIVISNUM 4.1 /* VITAL SIGNS VISIT NO.*/
@30 VIBPS 5.1 /* BP SYSTOLIC */
@35 VIBPD 5.1 /* BP DIASTOLIC */
@40 VIPLS 3.0 /* PULSE */
;
run;
libname reconstr clear;run;

```

Figure 5. Sample Conversion Validation Program.

```

/*$START$-----/
/*
/* PROGRAM: COMPARE.SAS
/* DIRECTORY: sesug:[program.950701]
/* AUTHOR: mudanoje
/* DATE CREATED: 07/01/95
/* PURPOSE: Compare the original SAS data sets
/*           with the reconstructed SAS datasets
/*$STOP$-----/

libname arcdata "sesug:[data.archive]";

%MACRO COMPARE(arc,recon);
/* arc is the name of the archived SAS data set;
/* recon is the name of the reconstructed SAS data set;

options formdlm='=';

/* create a test version of the archived data set for
comparison;
proc sort data=arcdata.&arc out=&arc;
by invno ptno;
run;

proc compare base=&arc
compare=RECONSTR.&recon
out=work.diffs
LISTBASE LISTCOMP NOSUMMARY
TRANPOSE maxprint=32000;
id invno ptno notsorted;
var _character_ _numeric_;
title "Comparing Reconstructed SAS Dataset &recon to Archived
Dataset &arc";
run;

%MEND COMPARE;

```