Using SAS to Write SAS – Automate Your Programming Tasks

Pam Reading
Rho, Inc., Chapel Hill, NC

Abstract: It doesn't always occur to a programmer to use SAS to write SAS code, but it can be a great time-saver. It's a reliable and efficient method for creating long, repetitive pieces of SAS code in which a similar set of operations is performed on a large number of variables. Once the code-generating program is functioning, the production code can be regenerated as needed and used as INCLUDE files in standardized programs. In a Windows environment, Excel spreadsheets are a convenient input format for the required data. The example presented in this paper is based on SAS Version 6.12 under Windows 98, but the principles presented apply to all operating systems. The paper addresses base SAS and the audience should be comfortable with basic DATA step operations.

Introduction: As a contract research organization working with a number of different clients, we often confront the problem of converting our standard SAS clinical trial data files to other formats. In the case presented here, our client required data in ASCII format for input into an ORACLE database system, with one record per 'question' or data value collected.

Figure 1 is a sample of demographics data for two subjects in the format required. Please note that all of the sample files illustrated in this paper have been abbreviated for clarity – the actual file structure of the ASCII file illustrated above included 18 variables and had a record length of 796.

PATIENT	EVENT	INSTRUMENT NAME	QUESTION GROUP	QUESTION NAME	SEQ	DATA VALUES
998	SCREENING	PATIENT	DEMOGRAPHICS	BIRTHDATE	1	19680401
998	SCREENING	PATIENT	DEMOGRAPHICS	RACE_CODE	1	C
998	SCREENING	PATIENT	DEMOGRAPHICS	RACE_OTHER	1	
999	SCREENING	PATIENT	DEMOGRAPHICS	BIRTHDATE	1	19700907
999	SCREENING	PATIENT	DEMOGRAPHICS	RACE_CODE	1	O
999	SCREENING	PATIENT	DEMOGRAPHICS	RACE_OTHER	1	MAORI

FIGURE 1

Our data management system produces SAS datasets, one record per logical page. Below is a subset of what our demography dataset looked like for the same two subjects.

ID	VISIT	SEQNO	INIT	BRTHDT	RACEC	ORACE
998	1000	00	ABC	01APR1968	C	
999	1000	00	XYZ	07SEP1970	O	MAORI

FIGURE 2

It was obviously a big and tedious job. In the first project, our data management system produced 36 different data files, which were destined to become 25 ORACLE formatted files. It was also quickly apparent that the specifications were going to change and evolve as the project progressed. Clearly, completely hard-coded conversion programs for each of the final delivery files would be nightmares to develop, debug and maintain. Furthermore, we wanted a system that could be modified easily to accommodate the multiple studies we were processing for this client.

Inspiration: The client sent us the specifications as a spreadsheet, an abbreviated version of which appears below as Figure 3.

Instrument Name	File Name	Question Group	Question Name	Data Type
PATIENT	PATIENT	DEMOGRAPHICS	BIRTHDATE	DATE
PATIENT	PATIENT	DEMOGRAPHICS	RACE_CODE	CHAR
PATIENT	PATIENT	DEMOGRAPHICS	RACE_OTHER	CHAR

FIGURE 3

Since the output records had a standard format, we realized we could handle a lot of the tedious coding by adding a few of our own data columns, then reading the spreadsheet information and writing SAS include files to convert the data.

Procedure: Figure 4 illustrates the spreadsheet after we added columns to 'translate' the client's variables to our own.

Instrument Name	File Name	Question Group	Question Name	Data Type	RHO NAME	RSEQ
PATIENT	PATIENT	DEMOGRAPHICS	BIRTHDATE	DATE	COMPRESS(PUT(BRTHDT,YYMMDD10.),'-')	1
PATIENT	PATIENT	DEMOGRAPHICS	RACE_CODE	CHAR	RACEC	1
PATIENT	PATIENT	DEMOGRAPHICS	RACE_OTHER	CHAR	ORACE	1

FIGURE 4

We wrote a SAS program to input the spreadsheet, using standard DDE syntax (Figure 5).

```
*************************
* Open Excel Spreadsheet *
*************************;
x "start &ROOT\RHOVARS.XLS";

**** THIS SECTION GETS FORMAT SPECIFICATIONS FROM AN EXCEL SPREADSHEET;

* r2c1:r999c7 means row 2 column 1 through row 999 column 7;
filename xls  DDE "excel|[rhovars.xls]sheet1!r2c1:r999c7";

****************************************
* Get Data From An Excel Spreadsheet      *
****************************************;
DATA FILEINF;
  informat INSTNAME  $VARYING30.   FILENAME  $VARYING15.  QGROUP $VARYING30.
           XQNAME $VARYING50. VARTYPE $VARYING10. RHONAME $varying200. XRSEQ $8. ;

 * this INFILE syntax makes commas in the data fields OK - see page 134 of SAS Companion for Windows
for source;
  infile xls dlm='09'x notab dsd missover;
  input  INSTNAME $  FILENAME $ QGROUP $ XQNAME $ VARTYPE $ RHONAME $ XRSEQ $  ;

RUN;
```

FIGURE 5

We next created a dataset of unique names, one per output file required, using PROC SORT NODUPKEY. This file was read into data *NULL* and macro variables are created, one per file name. Next, we implemented the following macro for each file in turn. The program wrote out two include files for each output file – one defining some standard information that appears on every record, and one defining the varying information for each 'question'.

The program code below creates the standard information file:

```
FILENAME INFO  "&ROOT\PROG\CONVERT\&STRM..INX";

** CREATE *.INX FILES FOR EACH DATASET;

DATA TEST;
 RETAIN BACKUP -1; * TO MOVE POINTER BACK ONE AND ELIMINATE IRRITATING SPACES;
  SET NODUPS;
  WHERE UPCASE(FILENAME)="&STRM";

   INSTNAME = UPCASE(INSTNAME);
   FILENAME = UPCASE(FILENAME);
   QGROUP   = UPCASE(QGROUP);

  FILE INFO;
       PUT @03 "**** &STRM  **** ; " / ;
       PUT @6 "INSTNAME   = '" INSTNAME +BACKUP "';";
       PUT @6 "FILENAME   = '" FILENAME +BACKUP "';";
       PUT @6 "QGROUP     = '" QGROUP +BACKUP "';" /;
  RUN;
```

FIGURE 6

The next section of the macro produces a second include file, setting up each specific output record and outputting it one by one.

```
** CREATE *.INC FILES FOR EACH DATASET;
FILENAME FMTS  "&ROOT\PROG\CONVERT\&STRM..INC";

DATA TEST2 ;
  SET FILEINF;
  FILE FMTS;
  WHERE UPCASE(FILENAME)="&STRM";

  LENGTH  RSEQ $8 XQVALUE MACCALL $200 CONSTANT 8  ;
  BACKUP=-1;   * TO MOVE POINTER BACK ONE AND ELIMINATE IRRITATING SPACES;

    IF XRSEQ='SEQNO' THEN RSEQ="SEQNO+1";
     ELSE RSEQ=XRSEQ;
    IF RSEQ='' THEN RSEQ='1';

    IF INDEX(RHONAME,'*')=1 THEN CONSTANT=1;  /* ASTERISK INDICATES LITERALS */
     ELSE IF INDEX(RHONAME,'%FIXDATE')=1 THEN DO;     * MACRO CALL FOR DATES;
       CONSTANT=2;
       MACCALL=RHONAME;
       XQVALUE=SCAN(RHONAME,2);
     END;

     ELSE XQVALUE=RHONAME;
```

```
    IF _N_=1 THEN  PUT @03 "**** &STRM **** ; " /
                       @03 "PATIENT = PUT(INPUT(PATIENT,8.),4.);"/;

    IF XQNAME>'' THEN DO;
      PUT @6 "QNAME  = '" XQNAME +BACKUP "';";
      PUT @6 "RSEQNO  = " RSEQ +BACKUP ";";

      IF XQVALUE>'' AND CONSTANT = . THEN PUT @6 "QVALUE = " XQVALUE ";" ;
       ELSE IF XQVALUE>'' AND CONSTANT = 1 THEN
         PUT @6 "QVALUE = ' " +BACKUP XQVALUE +BACKUP "';";
       ELSE IF XQVALUE>'' AND CONSTANT = 2  THEN
         PUT @6  MACCALL "; QVALUE =  "  XQVALUE ";";
       ELSE IF XQVALUE='' THEN  PUT @6 "QVALUE = ''; " ;

       PUT @6 'OUTPUT; ' /;

    END;

   RUN;
```

FIGURE 7

We then wrote a ‘standard’ conversion program that reads the data, loads the INCLUDE file, and writes the data out. The code in **BOLD** is the included code**.**

```
%LET PROG=PATIENT;

DATA &PROG (KEEP=PATIENT EVENT INSTNAME QGROUP QNAME RSEQNO QVALUE);
 SET DEMOFILE;

LENGTH PATIENT $8 EVENT $16 INSTNAME $12 QGROUP $16 QNAME $20 RSEQNO $4 QVALUE $200 ;

* THIS INCLUDE FILE SUPPLIES THE INSTNAME AND QUESTION GROUP NAME;
%INCLUDE "&ROOT\CONVERT\&PROG..INX";

**** PATIENT  **** ;
   INSTNAME  = 'PATIENT';
   QGROUP   = 'DEMOGRAPHICS';

*** THESE VALUE STAY CONSTANT ON EVERY RECORD IN THIS FILE;
PATIENT=ID;
EVENT=INPUT(VISIT,$VISIT.);


** FOR EACH ID, PUT OUT ONE RECORD PER QUESTION;
%INCLUDE "&ROOT\CONVERT\&PROG..INC" /SOURCE2;

**** PATIENT **** ;
 PATIENT = PUT(INPUT(PATIENT,8.),4.);

   QNAME  = 'BIRTHDATE';
   RSEQNO  = 1;
   QVALUE = COMPRESS(PUT(BRTHDT,YYMMDD10.),'-'); ;
   OUTPUT;

   QNAME  = 'RACE_CODE';
   RSEQNO  = 1;
   QVALUE = RACEC ;
   OUTPUT;
```

```
   QNAME  = 'RACE_OTHER';
   RSEQNO  = 1;
   QVALUE = ORACE ;
   OUTPUT;

RUN;

** NOW PUT VARIABLES OUT IN THE APPROPRIATE FORMAT;
FILENAME OUTFILE "&ROOT\&PROG..DAT";

DATA _NULL_;
SET &DNAME;
FILE OUTFILE LRECL=300 PAD;
PUT @1   PATIENT
    @10  EVENT
    @30  INSTNAME
    @45  QGROUP
    @65  QNAME
    @90  RSEQNO
    @95  QVALUE  ;
RUN;
```

FIGURE 8

The advantages to this kind of approach are many. If a variable name or type in our data management system is changed – we just change the spreadsheet. If the client adds, deletes or adjusts variables, we simply reflect those changes in the spreadsheet.

If we find we need to process the variable further, we can use SAS functions, FORMAT statements, etc. in the spreadsheet, as demonstrated. Within reason, we can put any SAS statements that can be used within the DATA step directly into the spreadsheet. It becomes part of the include file and is executed as usual when the conversion program is run. If the amount of code required becomes too great, we can create a custom macro to do the work, then modify the code writing program to recognize the macro call and set up the include code properly. See %FIXDATE section in Figure 5 above.

If whole files are added, deleted or renamed, we make appropriate adjustments to the spreadsheet, and the macro-driven procedure takes care of the inclusion files automatically. If files are added, we have to create another conversion program, the basic structure of which is standard across all files.

Conclusion: This approach to data file conversion has served us well for over 10 studies we have managed for this client. We also use a similar approach to generate data validation code, reading meta dataset descriptions from a spreadsheet into permanent SAS files, then producing code that checks each data point for missing values, invalid data, and range violations. Even if the reader never confronts a data conversion task like this one, the basic approach of using SAS to write SAS can be used to produce large amounts of reliable, easily modified code in a short time.

Contact: Pam Reading
Rho, Inc.
100 Eastowne Drive
Chapel Hill, NC 27514
(919) 408-8000 (voice)
(919) 408-0999 (FAX)
email: preading@rhoworld.com