Paper 248-29

Unusual Uses of SAS

Steven First, Systems Seminar Consultants, Madison, WI

ABSTRACT

Our company has used SAS almost exclusively for many years and we feel that it is clearly the best software for many applications. This paper will discuss uses of SAS that are not normally thought of, but for which SAS fits very well. This paper will be of interest to all SAS users.

This paper will discuss the following topics:

1. Conversion programs with SAS
2. Utility applications for programmers
3. Auditing other software packages with SAS
4. Production reporting systems with SAS.
5. Source control with SAS
6. Cross-referencing data sources with SAS

INTRODUCTION

In the history of SAS, there certainly have been huge changes in the industry, and I am constantly amazed at the versatility and staying power of the product. Being in a business where we need to do what clients ask for sometimes requires applications and solutions that are not at all what a person might expect at first glance.

CONVERSION PROGRAMS

With the versatility of SAS being able to run on most any platform and to manipulate data extensively, SAS is a perfect choice for many one time conversion programs.

Characteristics:

- SAS can read virtually any file format.
- SAS can also write virtually any file format.
- SAS can easily read and write from third party databases
- SAS allows access to system information.
- SAS provides ease of use as a programming language
- There is a general lack of other good tools in the industry.

A recent project to convert an older system to DB2 came to us because the original tables were SAS datasets. Obviously SAS needed to be used to read the data, but then after some clean up, the data was to be loaded to DB2. The system consisted of approximately 30 tables and originally the task was estimated at 40 hours.

The actual code went something like this:

```
libname oldsas 'old.sas.files';
libname sasdb2 db2 ssid=z1q1 location=z1q2
                 authid='z1q2001$';
data temp;
 set oldsas.xys;
  . . .   change as necessary
run;
proc append base=sasdb2.xysnewtable
             data=temp;
run;
```

While the project took much longer than originally estimated, it was data clean up that took the time. The SAS code was trivial. If the original input had not been stored as SAS files, SAS would not have been used at all, which would

have been a shame, because SAS did an excellent job of cleaning the data. In fact the new reporting for the system was written in COBOL which is a far inferior reporting language than SAS is. Other features of SAS such as MERGE or PROC SQL are clearly superior for many applications. One of the toughest things that we deal with, is that clients and especially IT staffs don't realize that SAS is an appropriate tool for these kinds of conversions.

This type of conversion has been done by our group many, many times with various input and output file types. In almost all cases, SAS could handle it with minimal programming.

Utility Applications

There are very few software packages that offer the flexibility of SAS for common utility applications. The ability to read, and write virtually any file, along with an extremely comprehensive report writer, make SAS a logical addition to any programmer's tool kit.

My very first SAS job was a rewrite of a COBOL program that transferred thousands of mostly blank records down a communications line and then unpacked them on the other end. Even though the COBOL code was written as a one-time program, it still took over 8 hours to write, compile, and test the two required programs. I knew there had to be a better way to accomplish this, and the two SAS programs listed below were written in much less time that did the same thing.

A SAS Solution – Two Small Programs

(JCL defined both files with LRECL=80).

```
DATA _NULL_;                          /* DON'T NEED DS     */
  INFILE IN;                          /* RAWFILE IN        */
  FILE OUT;                           /* RAWFILE OUT       */
  INPUT @1 TWENTY $CHAR20.;           /* READ 20 CHAR      */
  PUT      TWENTY $CHAR20. @;         /* 20 OUT, HOLD PTR  */
RUN;                                  /* END OF STEP       */


DATA _NULL_;                          /* DON'T NEED DATASET*/
 INFILE IN;                           /* RAW FILE IN       */
 FILE OUT LRECL=80;                   /* FILEOUT           */
 INPUT                                /* READ TWENTY/LOOP  */
     TWENTY  $CHAR20.                 /* FIXED PTRS CAUSE  */
    @@ ;                              /* LOOPS, BE CAREFUL */
 IF TWENTY NE ' ' THEN                /* IF NONBLANK ?     */
       PUT                            /* OUTPUT 20         */
           @1 TWENTY  $CHAR20.;       /* $CHAR SAVES BLANKS*/
 IF _N_ > 20 THEN                     /* A GOOD IDEA WHILE */
    STOP;                             /* TESTING           */
RUN;                                  /* END OF STEP       */
```

Below are listed a series of utility type applications. While they are in some ways trivial, they could be much more sophisticated, and even coded as they are, they are probably more advanced than what other languages and utilities provide.

Display Any File In A Hexadecimal Format.

```
DATA _NULL_;                  /* DONT NEED DATASET */
  INFILE IN;                  /* RAW FILE IN       */
  INPUT;                      /* READ A RECORD     */
  LIST;                       /* LIST BUFFER IN LOG*/
  IF _N_ > 50 THEN            /* STOP AFTER 50     */
     STOP;                    /* ADJUST AS NEEDED  */
```

```
RUN;                          /* END OF STEP       */
```

Copy any sequential file.

```
DATA _NULL_;                  /*DON'T NEED DATASET */
  INFILE IN;                  /* RAW FILE IN        */
  FILE OUT;                   /* RAW FILE OUT       */
  INPUT;                      /* READ A RECORD      */
  PUT _INFILE_;               /* WRITE IT OUT       */
RUN;                          /* END OF STEP        */
```

Changing DCB While Copying (Additional columns will be padded).

```
DATA _NULL_;                  /* DON'T NEED DATASET*/
    INFILE IN;                /* RAW FILE IN       */
    FILE OUT LRECL=90         /* INCREASE DCB AS   */
             BLKSIZE=9000     /* NEEDED            */
             RECFM=FB;
    INPUT;                    /* READ A RECORD     */
    PUT _INFILE_;             /* WRITE IT OUT      */
RUN;                          /* END OF STEP       */
```

Select part of a file.

```
DATA _NULL_;                  /* DON'T NEED DATASET  */
  INFILE IN;                  /* RAW FILE IN         */
  FILE OUT;                   /* RAW FILE OUT        */
  INPUT @5 ID $CHAR1.;        /* INPUT FIELDS NEEDED */
  IF ID='2';                  /* WANT THIS RECORD?   */
  PUT _INFILE_;               /* YEP, WRITE IT OUT   */
RUN;
```

Selecting a Random Subset

```
DATA _NULL_;                  /* NO DATASET NEEDED   */
  INFILE IN;                  /* RAW FILE IN         */
  FILE OUT;                   /* RAW FILE OUT        */
  INPUT;                      /* READ A RECORD       */
  IF RANUNI(0) LE .10;        /* TRUE FOR APP. 10%   */
  PUT _INFILE_;               /* WRITE OUT OBS       */
RUN;                          /* END OF STEP         */
```

Adding Sequence Numbers, write out a buffer, then overlay.

```
DATA _NULL_;                  /* NO DATASET NEEDED  */
  INFILE IN;                  /* RAW FILE IN        */
  FILE OUT;                   /* RAW FILE OUT       */
  INPUT;                      /* READ A RECORD      */
  SEQ=_N_*100;                /* COMPUTE SEQ NO     */
  PUT _INFILE_                /* OUTPUT INPUT REC   */
      @73 SEQ Z8.;            /* OVERLAY WITH SEQ   */
RUN;                          /* END OF STEP        */
```

Writing Literals in Every Record

```
DATA _NULL_;                  /* NO DATASET NEEDED  */
  INFILE IN;                  /* RAW FILE IN        */
  FILE OUT;                   /* RAW FILE OUT       */
  INPUT;                      /* READ A RECORD      */
  PUT _INFILE_                /* OUTPUT INPUT REC   */
      @10 'SSC';              /* OVERLAY WITH CONST.*/
RUN;                          /* END OF STEP        */
```

Correcting a Field on a File (Logic can be altered to match any situation.)

```
   DATA _NULL_;                    /* DON'T NEED DATASET  */
     INFILE IN;                    /* INPUT FILE IN       */
     FILE OUT;                     /* OUTPUT FILE         */
     INPUT @5 ID $CHAR1.;          /* INPUT FIELDS NEEDED */
     IF ID='2'                     /* CHANGE AS NEEDED    */
        THEN ID='3';
     PUT   _INFILE_                /* OUTPUT FILE         */
        @5 ID CHAR1.;              /* OVERLAY CHANGED ID  */
RUN;                               /* END OF STEP         */
```

Accessing System Control Blocks

In a OS/390 environment, SAS provides excellent access to several useful system control blocks. One particularly useful block is the Job File Control Block (JFCB). From it you can determine values of such things as datasetname, device type, catalog status, SYSIN or SYSOUT status, and label processing options Possible uses of the JFCB are to access dataset name from JCL for titles, determine whether it is reading a live VSAM file, a sequential backup diskfile, or a tape file, or to determine the date a dataset was created

A JFCB example: Determine the DSNAME and DSORG

```
   DATA _NULL_;                    /* DON'T NEED DATASET */
    INFILE IN JFCB=JFCBIN;         /* ASK FOR JFCB       */
    LENGTH TITLDSN $ 44;           /* SET LENGTHS AS     */
    LENGTH DSORG1 $1.;             /* REQUIRED           */
    IF _N_ = 1 THEN                /* FIRST TIME IN ?    */
     DO;                           /* YES, DO BLOCK      */
      TITLDSN=SUBSTR(JFCBIN,1,44);/* EXTRACT DSNAME     */
      DSORG1=SUBSTR(JFCBIN,99,1); /* AND DSORG BYTE 1   */
       IF DSORG1='.1......'B THEN /* BIT TEST AS NEEDED */
           DSORGOUT='PS';          /* MUST BE SEQUENTIAL */
     END;                          /* END OF BLOCK       */
    INPUT etc. ;                   /* REST OF PROGRAM    */
     . . .
    RETAIN TITLDSN DSORGOUT;       /* RETAIN             */
   RUN;                            /* END OF STEP        */
```

SAS can also read complicated file structures such as pds directories, VSAM catalogs, load modules. In earlier shipments of SAS a sample library was provide that showed examples of reading such files though I am not sure if those sample programs are still available.

Reading other program's output

Another alternative that is sometimes available to avoid reading overly complex files is to use an already available program that reads the file and creates an extract file that SAS can then read. Examples of such programs include programs that read VTOCs, source management systems, security systems, financial packages, system utilities, and virtually any program product.

A Panvalet® Example

The user wanted a different sort sequence than PANVALET provided.
Solution: Use PAN#3, then SAS

```
   // EXEC PGM=PAN#3
   //*************************************/
   //*   USE PAN#3 TO CREATE DIRECTORY    */
   //*************************************/
   //SYSPRINT DD SYSOUT=*
   //SYSPUNCH DD DSN=&&PANDIREC,DISP=(,PASS),
   //           UNIT=DISK,SPACE=(TRK,25)
```

```
//PANDD1   DD DSN=PANVALET.LIBRARY,DISP=SHR
++CONTROL ....
++PRINT 0-UP
//*************************************/
//*   CREATE REPORTS WITH SAS         */
//*************************************/
//SAS   EXEC SAS
//PANDD1   DD DSN=&&PANDIREC,DISP=(OLD,DELETE)
The Panvalet/SAS Source
DATA PANDIREC;
INFILE PANDD1 ;
INPUT @01 PNPGMNM  $10.
      @11 PNLEVEL  3.
      @19 PNPGMTYP $5.
      @27 PNMTDATE MMDDYY8.
      @35 PNACDATE MMDDYY8.
      @24 PNSTATUS $3.
      @43 PNNOBLK  4.
      @48 PNNOBLK  8.;
RUN;
PROC PRINT DATA=PANDIREC;
  TITLE 'PAN DIRECTORY';
RUN;
```

Programs that Write Other Programs

Another alternative may be to use SAS to construct instructions to another applicaton which could then be submitted to an internal reader or file.
Possible applications might be job schedulers, disk management programs, or many other applications. I have written programs that cleaned up disk space by deleting old unwanted programs as well as an automated job submission system. In the first case early bugs caused a logic problem which deleted far too many files and in the second case submit too many jobs. It is critical to test very thoroughly and always provide a backup.

A Downloading Example - Transfer All SAS Programs Under CMS.

```
X LISTFILE  SAS A (EXEC;                    /* CMS COMMAND    */
X FILEDEF FLIST DISK CMS EXEC A;            /* LISTFILE OUTPT */
X FILEDEF GENPRG DISK DOWNPRG SAS A;        /* TEMP PROGRAM   */
DATA _NULL_;                                /* BUILD FILEDEFS */
 INFILE FLIST;                              /* PROC DOWNLOADS */
 INPUT @8 FN $8. @17 FT $8.;
 FILE GENPRG;
 PUT 'X FILEDEF HOSTFILE DISK ' FN FT ' L;'
    /
    'PROC DOWNLOAD INFILE=HOSTFILE OUTFILE="C:\DOWNDIR\'
    FN '.SAS";'
     /  'RUN;';
RUN;
%INCLUDE GENPRG;                            /* INC GENED CODE  */
RUN;
```

A Mass Change Program

Another favorite program is an application that could do mass changes in thousands of members. Example: Change all SYSOUT=A to SYSOUT=* in a PROC Library
Issues are the large number of libraries and members, effect on production, change logic may be complex. Again, back up before, and be careful.

```
// EXEC SAS
//PDSIN  DD  DSN=SOMEPDS,DISP=SHR
//SEQIN  DD  UNIT=DISK,SPACE=(TRK,500)
//SEQOUT DD  DSN=&&SEQOUT,DISP=(,PASS),
```

```
//         UNIT=DISK,SPACE=(TRK,500)

PROC SOURCE INDD=PDSIN OUTDD=SEQIN
 SELECT . . .;    / MODIFY AS NEEDED /
RUN;

DATA _NULL_;
  INFILE SEQIN;
  INPUT @1 WHOLE $CHAR80.;
  COL=INDEX(WHOLE,'SYSOUT=A');
  IF COL NE 0 THEN
     SUBSTR(WHOLE,COL,8)='SYSOUT=*';
  FILE SEQOUT;
  PUT @1 WHOLE $CHAR80.;
RUN;

// EXEC PGM=IEBUPDTE
//SYSPRINT DD SYSOUT=*
//SYSUT2 DD DSN=SOMEOTHR.PDS,DISP=OLD
//SYSIN  DD DSN=&&SEQOUT,DISP=(OLD,DELETE)
```

Auditing other software packages with SAS

We have seen a increase in the last year of other very specialized purchased programs for customer systems which precipitated a very interesting use of SAS. Even though the client will use the new package for the application, we have written "throw_away" systems to help in the verification of such software. In the first case we read output from two releases of such a package, and used a series of simple SAS programs using data steps, procs like FREQ, MEANS, and COMPARE, to verify that results were identical between the releases. If results were not identical, adjustments were made until the new release could be implemented with complete confidence.

In another case, the software package had a series of IF like statements that needed to be coded from a set of business rules. Without seeing the statements, we were engaged to create a duplicate system in SAS again from the business rules. This provided a completely independent verification process that insured that the package was implemented correctly, at which time the SAS system can be discarded.

A third application was used to quickly paint out screens using PROC FSEDIT as a prototype for a more extensive online system. By developing the screens in real time alongside the client, screens were mocked up very quickly. The screens could then be developed in other languages for the production systems. In an interesting twist, there were a couple of instances where the prototype was adequate to solve the application requirements, and it was used as a complete solution, and no further production system was developed.

Production reporting systems with SAS.

A more obvious application that is nonetheless, often overlooked, is that SAS is in many if not all ways, one of the best production report writing systems. Countless packages have been written over the years that do a specific task and those systems may include a report writing module that is almost always inferior to SAS. Examples of these kind of applications are Financial systems, credit systems, security packages, disk management, tape management, and countless other vertical applications as well as custom written reporting systems.

In almost every package that I have evaluated, the reporting modules were inferior to SAS and in some cases very expensive. While I understand that COBOL is used because of the number of programmers familiar with it, it still seems very primitive to write reports using a second generation language when SAS is available.

The performance questions that come up are normally not a major issue, but instead we need to get IT staffs aware of what SAS has to offer in a production environment.

One recent large reporting system sent hundreds of pages of reporting to HTML files along with integrated graphs and a menu to navigate and archive the reports as a new set was generated. This system was

completed generated on the mainframe where the data was located, and the reports and graphs were automatically distributed to the web servers on different computers.

Miscellaneous Applications

Several other systems that I am familiar with are a bit unusual, but were very interesting and successful applications. The reason why SAS did a good job, was again SAS's flexibility to read very difficult file structures and a strong reporting capability.

Some examples are:

- Source control systems
- Cross referencing data sources from JCL, and libraries
- Reading Load modules to map out subroutine calls
- Reading partitioned data sets (PDS) directories
- Performance systems
- Report management systems

Excel Runs the World

More than any other computer development, spreadsheets probably have the most impact on users than anything else. Not only are spreadsheets used for analysis, but are in many ways the preferred universal report format. SAS can very easily produce spreadsheets using ods or a variety of other options.

Conclusions

In my opinion, SAS continues to be the premier analysis and reporting software system, and sometimes the applications are not obvious at first glance.

CONTACT INFORMATION

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

Author Name	Steven First
Company	Systems Seminar Consultants
Address	2997 Yarmouth Greenway Drive
City state ZIP	Madison, WI 53716
Work Phone:	608 278-9964 x302
Fax:	608 278-0065
Email:	sfirst@sys-seminar.com
Web:	sys-seminar.com

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.