

Hidden Gems in the SAS® System

Steven First
Systems Seminar Consultants
Madison, WI 53716 (608) 222-7081

Abstract

The SAS system has a wealth of features that makes it so unique, and sometimes those features are not obvious or easy to find in documentation. This paper will look at several new features, as well as some features that have been around for a while, that we have found to be useful.

Introduction

Our office's SAS documentation subscription brings us new manuals almost weekly, and our total library of SAS related material covers one wall. While our library is very thorough and the SAS documentation is of very high quality, one problem that sometimes occurs is that enhancements are often documented in technical reports that many users don't see, and then of course they don't know about those new features.

I have noticed that very often that features listed in technical reports are simple items that can save an enormous amount of work.

Raw Data Handling

Certainly one of the strongest features of the SAS system is the ability to read and write just about any file including some with very complex layouts. Examples of some difficult layouts are files with multiple record types, variable length records, files with imbedded data that needs to be scanned, data with lower case characters, short records, data with recurring values, and comma delimited data for import/export to other products. SAS can handle all of the above. INPUT and PUT of course offer a wealth of features, but here I am going to investigate just a few of them.

My favorite SAS feature

The trailing @ with INPUT allows us to read a record, examine it, and then "reread" the same record with a second INPUT if need be. Uses are to read multiple record types or to read some of the values in a record which are needed to read other data on the same record. The trailing @ is a very efficient feature because since the record is already in memory, the second INPUT is very fast.

An INPUT trailing @ example:

1. The following system file has 255 kinds of records each with a unique layout:

type 05 is a job record
type 15 is an dataset record
type 21 is a tape dismount record.

2. The records can appear in any order and are not necessarily related.
3. Build a separate SAS dataset for each of the three types listed above. Delete any other records found.

05	PRODPAYR	J1981	05/19/91	10:31:02	
15	PROD.MONTHLY.PAYROLL				3380
15	PROD.MONTHLY.BACKUP				3400
21	181	145664			
05	TSTCOMPL	J1982	05/19/91	10:32:02	
15	SYS1.LINKLIB				3380
15	XYZ.TESTLIB				3380

```
DATA JOBS(KEEP=JNAME JNUM JDATE JTIME)
  DSNS(KEEP=DSN UNIT)
  TAPE(KEEP=ADDR VOLUME);
INFILE RAWIN;
INPUT @1 TYPE $2. @;          /* INPUT AND HOLD */
IF TYPE='05' THEN             /* IS IT A 05 ? */
  DO;                          /* YES */
    INPUT @4 JNAME $8. @13 JNUM $5.
                                /* READ IT AGAIN */
                                @19 JDATE MMDDYY8. @28 JTIME TIME8.;
    OUTPUT JOBS;               /* WRITE TO JOBS */
    RETURN;                   /* GET OUT */
  END;
IF TYPE='15' THEN             /* IS IT A 15 ? */
  DO;                          /* YES */
    INPUT @4 DSN $44. @50 UNIT $4.;
                                /* REREAD */
    OUTPUT DSNS;
    RETURN;
  END;
IF TYPE='21' THEN             /* IS IT A 21 ? */
  DO;
    INPUT @4 ADDR $3. @10 VOLUME $6. ;
    OUTPUT TAPE;
    RETURN;
  END;
FORMAT JDATE MMDDYY8. JTIME TIME8.
RUN;
```

Trailing @ with PUT

Many features of INPUT can also be applied to PUT when a program needs to write raw data. The trailing @ can be coded after a PUT statement so that the next PUT statement doesn't write to a new record, but rather appends to the same record.

A PUT trailing @ example:

DATASET CUSTID		
OBS	CUSTID	SALESID
1	10001	900201
2	10003	900386
3	10003	900386
4	10003	900386

CUSTID and SALESID appear in a SAS dataset, print each CUSTID only once for each SALESID.

```
DATA NULL;
  TITLE 'PUTS With Trailing a';
  BY CUSTID;
  FILE PRINT;
  SET CUSTSALE;
  IF FIRST.CUSTID
    THEN PUT @11 CUSTID $5. a;    ** put and hold ptr;
  PUT @24 SALESID $6.;            ** put on same line;
RUN;
```

PUTS With Trailing a		
10001	900201	
10003	900386	
	900386	
	900386	

Scanning With INPUT

An INPUT pointer can be a constant or a character variable which causes INPUT to scan the record looking for the value anywhere in the record, and then continue reading from that point.

Example: Read the name after a title:

```
INPUT @'Mr.' NAME $CHAR20.;
```

New Informats With SAS 6.07

The \$QUOTEw. informat removes beginning and ending quotes during INPUT.

\$UPCASEw. translates any lower case characters to upper case.

Example: Remove the quotes around COMPANY and uppercase NAME.

```
data compfile;
  infile datalines;
  input @1 company $quote10.
        @20 name $upcase10.;
  datalines;
  "SSC"          Steve
  ;
proc print data=compfile;run;
```

OBS	COMPANY	NAME
1	SSC	STEVE

Variable Lists and Format Lists

When a table of recurring values needs to be read, both the variables and the formats to be used can be lists rather than specifying all values. This makes it as easy to read 500 values as it is to read 5.

Example: You have a file with a 2 character state followed by a 3 digit temperature. This sequence is repeated for all 50 states. You need to read all 50 state's data into one observation.

```
AK065 AZ101 AL092 AR078 . . . . . WY054
```

```
data temps;
  infile tempsin;
  input @1(state1-state50) ($2. +4)
        /* start in 1, read 2, skip 4 */
        @3(temp1-temp50) (3. +3)
        /* back to 3, read 3, skip 3 */
  ;
RUN;
```

User Written Informats

PROC FORMAT's INVALUE statement creates informats that edit and change values as they are read. This can do such things as translate character values to numeric, and edit values for correctness. This technique can be a fast and easy way to validate data on screens for online applications.

Example: translate letter grades to numerics for a grade point average and validate for correct departments.

```
proc format;
  invalue gradfmt 'A'=4 'B'=3
                  'C'=2 'D'=1 'F'=0;
  invalue deptfmt 101-888=_same_
                  other =_error_;
run;

data graddata;
  infile datalines;
  input @1 grade gradfmt1.
        @10 dept deptfmt3. ;
  datalines;
  A      707
  B      999
  ;
proc print data=graddata;run;
```

OBS	GRADE	DEPT
1	4	707
2	3	.

Special Variables

The special variables `_ALL_`, `_CHARACTER_`, `_NUMERIC_`, and `_INFILE_` can make some programming tasks easier. `PUT _ALL_` will display all data step variables and is an easy way to debug programs. `_CHARACTER_` and `_NUMERIC_` along with variable lists can be used to create a comma delimited file suitable for input to many software products. `_INFILE_` references the input buffer and it can be handy to use whenever most or all of the raw input record needs to be moved or changed.

Example: Make a copy of a raw file putting 'SSC' in columns 10-12 of every line.

```
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 */
```

Example: Create a file to import to a spreadsheet from any SAS dataset. All character variables will appear first followed by all numerics.

OBS	NAME	AGE	DEPT	RATE
1	STEVE	43	ACCT	12.22
2	DAVID	.	PAYR	11.21

```
options missing=' ';
data _null_;
  set addrdata;
  file rawfile;
  put (_character_) ('/'$'/'') ' '
      (_numeric_) (best.'/'');
run;
```

```
"STEVE ","ACCT ",      43,      12.22
"DAVID ","PAYR ",      ,      11.21
```

Reading Comma Delimited Data

The opposite of the last example is the case where we have a comma delimited file and need to read it into a data step. This type of file uses the double quote character to surround character values and commas between all fields, and of course all fields can be variable widths. This is very common and can be very difficult to read without some special `INFILE` options. The `DLM=` options allows us to specify the comma as the separator for list input, but that won't handle at least two special cases. First when a field is missing two consecutive commas are in the input file or when an imbedded comma is found in a character field is isn't handled correctly. In any case the `INFILE DSD` option has been a lifesaver that has handled all cases that I have encountered. Further details are found in

tech report P-222.

Example: Read a comma delimited file.

```
"STEVE ","ACCT ",      43,      12.22
"DAVID ","PAYR ",      ,      11.21
```

```
data addrdata;
  infile rawfile dsd;
  input name $
        dept $
        age
        rate ;
run;
proc print data=addrdata;run;
```

OBS	NAME	DEPT	AGE	RATE
1	STEVE	ACCT	43	12.22
2	DAVID	PAYR	.	11.21

Short Records

Another problem that comes up often, especially with ASCII text files is that some records are shorter than others and if the `INPUT` pointer goes beyond the right side of the record SAS flows into the next record or sets variables to missing. The `MISSOVER` and `TRUNCOVER` options can help with the flowing problems and the `INFILE LRECL` and `PAD` options can essentially make the records fixed length padded with spaces.

Example: Some cities are longer than others.

```

1
1234567890123456789
File Steve 32 Monona
RAWIN Tom 44 Milwaukee
Kim 25 Madison
```

```
DATA ADDRESS;
  INFILE RAWIN PAD LRECL=19;
  INPUT NAME $1-5 AGE 8-9 CITY $11-19;
RUN;
PROC PRINT;
  TITLE 'ADDRESS';
RUN;
```

ADDRESS			
OBS	NAME	AGE	CITY
1	Steve	32	Monona
2	Tom	44	Milwaukee
3	Kim	25	Madison

Reading or Writing Multiple Files

The `INFILE FILEVAR` option effectively allows a program to dynamically change the name of the file being read. `FILEVAR` names a dataset variable whose change in value causes the data

step to open a new file.

Uses might be to read all the members in an MVS PDS or all the files in a directory etc. This option is also available as a FILE option when writing multiple raw data files.

Example: Read every file listed after datalines.

```
data filesds;
  input filename $ 1-20;
  infile rawfile filevar=filename end=end;
  do until(end);
    input value $1-5;
    output;
  end;
datalines;
Q:\TESTDAT\FILE1.DAT
Q:\TESTDAT\FILE2.DAT
Q:\TESTDAT\FILEA.DAT
;
```

Transporting SAS Data

SAS/CONNECT certainly provides multiple ways to conveniently move SAS libraries and their entries across computer platforms. This does mean that SAS/CONNECT must be licensed on all used platforms, and that scripts are written and debugged. Another way to move data without SAS/CONNECT is to use the SAS XPORT engine to create a SAS XPORT file.

The steps in using this special binary file are to:

1. Create the XPORT file on one system using XPORT and PROC CPORT.
2. Move the XPORT file to the second system using a binary transfer program.
3. Construct a new SAS library on the second system using PROC CIMPORT.

We have moved data using this technique to and from several different SAS platforms and have found it to be especially valuable.

Example: Move a SAS library from MVS to WINDOWS via a XPORT file.

The MVS job:

```
//STEP02 EXEC SAS6
//SASDATA DD DSN=MY.SAS.LIBRARY,DISP=SHR
//OUT DD DSN=MY.XPORT.DAT,DISP=(,CATLG,DELETE),
// SPACE=(TRK,(100,100),RLSE),
// DCB=(LRECL=80,RECFM=FB,BLKSIZE=8000)
PROC CPORT LIBRARY=SASDATA FILE=OUT;
RUN;
```

The Windows job:

```
FILENAME FILEIN 'C:\TEMP\GRAPHDOWN';
LIBNAME SASDATA 'C:\SASDATA';
PROC CIMPORT LIBRARY=SASDATA
  INFILE=FILEIN;
RUN;
```

DATA Step Views

In release 6.07 DATA step views were introduced. "Introduction to DATA Step Views", POLZIN SUGI 18 along with Tech report P-222 describe this feature. The intention here is to make data appear *logically* as a SAS data set, even though *physically* the data may be in flat files, SAS data files, DBMS systems, or other sources. An input DATA step view can be virtually any DATA step that eventually outputs an observation. When a consumer of an observation (a proc or data step) requests an observation, the code in the view is executed and an observation is returned.

Two features of DATA step views are:

1. Simplification of code. Very complex programs can be stored once and when referenced everything appears a SAS dataset.
2. Multiple passes may be eliminated saving CPU, wall time and intermediate work space.

Example: Two large flat files that are already sorted by account number need to be merged.

Solution 1: The traditional method is to convert each flat file to a SAS data set, then in a third step merge them. The data from file one is passed three times, that from file two is passed three times, and finally the final dataset is created.

```
data file1;
  infile raw1 (recl=60 pad;
  input a1 field1 $10.
        a10 field2 $10.
        a20 acct $20.;
run;
data file2;
  infile raw2 (recl=60 pad;
  input a1 field3 $10.
        a10 field4 $10.
        a20 acct $20.;
run;
data both;
  merge file1
        file2;
  by acct;
run;
```

Solution 2: By using data step views, the flat files are not passed until the consumer (the merge step), and thus the data from the two raw files are only passed one time. Depending on file sizes, some significant time savings as well as significant intermediate space savings should be realized.

```
data file1/view=file1;
  infile raw1 (recl=60 pad;
```

```

input a1 field1 $10.
      a10 field2 $10.
      a20 acct $20.;
run;
data file2/view=file2;
infile raw2 lrecl=60 pad;
input a1 field3 $10.
      a10 field4 $10.
      a20 acct $20.;
run;
data both;
merge file1
      file2;
by acct;
run;

```

Dictionary Tables

As part of PROC SQL dictionary tables and corresponding SAS views have been set up to provide programs with names of SAS libraries allocated, members within the libraries, variables inside data members, entries within SAS catalogs, SAS options and much more.

A partial list of the dictionary tables is shown below; for a complete listing the documentation or examination of the SASHELP library can be used.

SASHELP.VMEMBER	SAS files and catalogs
SASHELP.VTABLE	SAS datafiles
SASHELP.VCOLUMN	SAS vars in all datasets
SASHELP.VCATALG	SAS catalogs and entries
SASHELP.VEXTFL	external files allocated
SASHELP.VVIEW	SAS views
SASHELP.VINDEX	SAS indexing information
SASHELP.VOPTION	SAS options
SASHELP.VSLIB	Sorted libname list
SASHELP.VSTABLE	Sorted table list
SASHELP.VSCATLG	Sorted catalog list
SASHELP.VSACCESS	Sorted access descriptors
SASHELP.VSVIEW	Sorted data views
SASHELP.VSTABVW	Sorted files and views

To use the views, just reference them like any other SAS dataset.

Example: List all the datasets and their number of observations created since 1 January.

```

proc print data=sashelp.vtable;
where crdate ge '01jan96:0:0'dt
and libname='WORK' ;
var memname crdate nobbs;
run;

```

OBS	MEMNAME	CRDATE	NOBS
1	ADDRDATA	13JAN96:10:19:58	2
2	BOTH	13JAN96:10:20:07	1730
3	COMPFILE	13JAN96:10:19:52	1
4	FILE1	13JAN96:10:19:59	1730

PROC SQL Pass-Through Facility

Originally access to a third party DBMS was done through SAS ACCESS and VIEW descriptors. In release 6.07 PROC SQL allows you to write native SQL code that is passed through relatively untouched to the DBMS for processing. This facility can be crucial to applications especially using large files. The only documentation for the facility is in the technical reports, thus users consulting only the SAS/ACCESS manuals might not be aware of its existence.

Example: Create an extract datafile from DB2.

```

PROC SQL;
CONNECT TO DB2(SSID=OP05);
CREATE TABLE ENROLL AS SELECT *
FROM CONNECTION TO DB2
(SELECT DOB,
      EFFDATE,
      NAME
FROM DBFILE.ENR
WHERE STATE='WI')
AS EXTRACT ( DOB,
      EFFDATE,
      NAME);
PROC PRINT DATA=ENROLL;RUN;

```

The SET KEY = Option

KEY = allows direct access to an observation by an index value. This allows what other systems call random access. KEY = can be used with with MODIFY or SET.

A KEY example: Read from MASTER, only the observations that are also in SELFIE.

MASTER		
OBS	OFFACCT	MATCHED
1	12345	
2	23456	
3	34567	

SELFIE	
OBS	OFFACCT
1	12345
2	67890

```

OPTIONS ERRORS=0;
DATA MATCHES;
SET SELFIE;
SET MASTER KEY=OFFACCT; * RAND READ MASTER;
IF _IORC_=0 THEN
  _OUTPUT; * FOUND IT;
RUN;
PROC PRINT DATA=MATCHES;RUN;
RUN;

```

OBS	OFFACCT	MATCHED
1	12345	

The MODIFY Statement

MODIFY works with some other SAS statements to update a SAS dataset in place rather than creating new SAS files as SET, MERGE, and UPDATE do. The following statements can be used in conjunction with MODIFY:

OUTPUT writes the current obs to the end of the dataset
REPLACE current observation is updated in place
REMOVE deletes the current observation

Example: Update in place any master records that matched with SELFIE.

```
DATA MASTER;
SET SELFIE;
MATCHED='';
MODIFY MASTER KEY=OFFACCT; * RAND READ TO MASTER;
IF _IORC_=0 THEN
DO;
MATCHED='Y';
REPLACE;
END;
RUN;

PROC PRINT DATA=MASTER;
RUN;
```

OBS	OFFACCT	MATCHED
1	12345	Y
2	23456	
3	34567	

New DATA Step Functions

Each release brings more and more functions. A pair that are fairly new are the QUOTE and DEQUOTE functions.

QUOTE adds double quotes around a value and DEQUOTE removes them. In the past a lot of substrings and concatenation was required to do the same thing.

Example:

```
data compfile;
infile datalines;
input @1 company $10.
@20 name $10.;
company=dequote(company);
name=quote(name);
datalines;
"SSC" Steve
proc print data=compfile;run;
```

OBS	COMPANY	NAME
1	SSC	"Steve"

Old DATA Step Functions

Mentioning just a few of the hundreds of SAS functions is difficult, but a few that are especially useful are MOD, RANUNI, PUT, and INPUT.

MOD returns a remainder from a division and is useful to select every nth record.
RANUNI gives a uniformly distributed random number and can be used to pull approximate size samples easily.

```
IF MOD(_N_,100)=0; * select every 100th rec;
IF RANUNI(0) GE .10; * select app. 10 %;
```

PUT and INPUT give very concise ways to convert between SAS character and numeric values without messages.

Examples:

```
DATA PUTINPUT;
INPUT NAME $1-10
RATE $12-18 /* RATE IS CHAR COMING IN */
HOURS 20-35; /* HOURS IS NUMERIC */
RATEN=INPUT(RATE,COMMA12.); /* RATEN IS NUMERIC */
HOURSC=PUT(HOURS,Z4.0); /* HOURSC WILL BE CHARACTER*/
DATALINES;
TOM $10,000 22
;
RUN;

PROC PRINT DATA=PUTINPUT;
RUN;
```

PUT AND INPUT FUNCTIONS					
OBS	NAME	RATE	HOURS	RATEN	HOURSC
1	TOM	\$10,000	22	10000	0022

The VNAME Function

VNAME can be used to return the variable name of data step variables. This can be useful when working with arrays where you may not know the actual underlying variable names.

Example: Modify the earlier program that exported to a spreadsheet to include a line containing the variable names.

OBS	NAME	AGE	DEPT	RATE
1	STEVE	43	ACCT	12.22
2	DAVID		PAYR	11.21

```
data _null_;
set addrdata end=eof;
file rawfile;
array chars(*) _character_;
array nums(*) _numeric_;
```

```
length varname $ 25;

if _n_ = 1 then
do;
  do j=1 to dim(chars);
    call vname(chars(j),varname);
    put ' ' varname ' ', 'a;
  end;
  do j=1 to dim(nums);
    call vname(nums(j),varname);
    put ' ' varname ' ', 'a;
  end;
  put +(-1) ' ';
end;
do j=1 to dim(chars);
  put ' ' chars(j) ' ', 'a;
end;
do j=1 to dim(nums);
  put nums(j) best9. ' ', 'a;
end;
put +(-1) ' ';
run;
```

```
"NAME " "DEPT " "AGE " "RATE "
"STEVE " "ACCT " 43, 12.22
"DAVID " "PAYR " , 11.21
```

Report Writing Gems

PROC FSEDIT in batch jobs can produce reports that display one observation per page much like the terminals display. Connecting it with PROC FSLETTER allows you to produce reports with constant text and one observation per page which is useful not only for form letters, but also for mailing labels or directories with one obs per page. We are all familiar with form letter programs, but here is one that can be used on a mainframe to process thousands of observations.

A form letter example:

```
PROC FSEDIT DATA=SASDATA.PERSON
  SCREEN=SASDATA.PERSON
  LETTER=SASDATA.LETCAT
  SEND=RETIRE;RUN;
```

PROC REPORT In Batch

This proc in a batch mode has numerous features that could take an entire day to talk about. Two very simple uses though can be discussed very briefly, with very little more effort than PROC PRINT. These programs can be built upon to compute new fields, control layout and headers and much more. Documentation is in Tech Report P-258.

Example: A very simple PROC REPORT.

```
proc report data=sales nowindows;
  title 'A Simple Proc Report';run;
```

A Simple Proc Report

DEPT	DATE	SALES
ACCTG	01/01/95	712.22
ACCTG	02/01/95	622.22
ACCTG	03/01/95	789.32
ACCTG	04/01/95	811.23
ACCTG	05/01/95	799.16
ACCTG	06/01/95	613.66
ACCTG	07/01/95	744.88
SALES	01/01/95	912.22
SALES	02/01/95	722.22
SALES	03/01/95	889.32
SALES	04/01/95	911.23
SALES	05/01/95	799.16
SALES	12/01/95	909.77

Example: A phone book style multi-column report.

```
proc report data=sales(keep=dept sales) nowindows
  panels=2;
  title 'A Multi Column Report';run;
```

A Multi Column Report

DEPT	SALES	DEPT	SALES
ACCTG	712.22	SALES	813.66
ACCTG	622.22	SALES	644.88
ACCTG	789.32	SALES	909.22
ACCTG	811.23	SALES	677.13
ACCTG	799.16	SALES	891.24
ACCTG	613.66	SALES	777.78
ACCTG	744.88	SALES	909.77
ACCTG	609.22		
ACCTG	677.13		
ACCTG	691.24		
ACCTG	777.78		
ACCTG	989.77		
SALES	912.22		
SALES	722.22		
SALES	889.32		
SALES	911.23		
SALES	799.16		

PROC PRINT

For those users who don't care for the way PROC PRINT rotates headers to save room or the way it breaks lines, two new options give you the control.

HEADING = V forces vertical headers
HEADING = H forces horizontal headers

ROWS = PAGE prints only one set of variables per page and continues on the next page with the remainder of variables.

```
proc print data=sales(obs=5) heading=v;
  title 'Vertical Headings';run;
```

Vertical Headings			
O B S	D E P T	D A T E	S A L E S
1	ACCTG	01/01/95	712.22
2	ACCTG	02/01/95	622.22
3	ACCTG	03/01/95	789.32
4	ACCTG	04/01/95	811.23
5	ACCTG	05/01/95	799.16

BY Variables in Titles

One of the simplest and most useful options from release 6.07 is the ability to reference BY variable names and values in TITLE statements, where in the past a macro or complicated annotation might have been required.

Coding `#BYVAR(byvariable)` substitutes the BY variable's name, and `#BYVAL(byvariable)` substitutes its value into a TITLE.

Note when using these features, `OPTIONS NOBYLINE;` should be specified to eliminate the default by lines.

Example:

```
options nobyline;
proc print data=sales;
  title 'Sales for Department #BYVAL(DEPT)';
  by dept;
  pageby dept;
run;
```

Sales for Department ACCTG		
OBS	DATE	SALES
1	01/01/95	712.22
2	02/01/95	622.22
3	03/01/95	789.32
4	04/01/95	811.23
5	05/01/95	799.16
12	12/01/95	989.77

Sales for Department SALES		
OBS	DATE	SALES
13	01/01/95	912.22
14	02/01/95	722.22
15	03/01/95	889.32
16	04/01/95	911.23
17	05/01/95	799.16
24	12/01/95	909.77

Data Step Reporting

`_BLANKPAGE_` and `_PAGE_@`

Some very subtle features of PUT can cause extra lines to appear when ejecting pages.

`PUT _BLANKPAGE_` ejects a blank page and moves the pointer to top of the next page.
`PUT _PAGE_` leaves the pointer on the next line on the same page.
`PUT _PAGE_@` eliminates the extra blank line.

The PUT statements alignment specifications of `-L`, `-C`, and `-R` can be specified after a format to left, center, or right justify values.

Example:

```
PUT X DOLLAR7.2 -C;
```

The `n*"character"` format modifier.

Example: Display 70 dashes in a report.

```
put @1 70*"-" ;
```

The SAS Macro Facility

One of the most difficult thing about debugging macros was knowing not only macro variable values, but also knowing in which macro referencing environment they were stored. One of the handiest features in the DATA step is the `PUT _ALL_` statement, and in release 6.11 an equivalent statement is available for the macro environment. This is a major enhancement that only requires 2 words.

Example: Display all macro variables and their environments.

```
%PUT _ALL_;
```

SAS PROCs

The procedures listed below are ones that we seem to use extensively and in most cases invoke them very simply. Covering all of the options would be beyond the scope of this paper, but I would like to mention a few uses and new options.

PROC FREQ

Is used to display distributions of a variable. We also use FREQ extensively to verify data while testing. An examples might be to produce a table of states values in a mailing list. This table can give the programmer confidence in the values and easy display incorrectly keyed states etc.

A production system I support has dated transactions IE. For the JUNE run all transactions should have occurred in JUNE, but sometimes the wrong month's tape was input.

The simple PROC FREQ of the transaction date as the first report gives the data control person an easy way to verify that the data makes sense.

Example:

```
PROC FREQ DATA=TRANS;  
  TABLE TRANDATE;  
  TITLE 'Table of Transaction dates for this run';  
RUN;
```

PROC SUMMARY/MEANS

Without question SUMMARY/MEANS is one of the most valuable programs to summarize data and reduce data in a production environment.

It is certainly easier to process a smaller summarized file than a large detailed one, and we use SUMMARY/MEANS extensively.

The ORDER = OPTION

A very valuable addition to PROC FREQ, PROC TABULATE, PROC SUMMARY/MEANS, and PROC REPORT has been the ORDER = option. This option gives four choices for the order that the final tables will be classified and printed. In the past, tricky programs to make up dummy values and fool the system were needed. The four ORDER = values are:

DATA	class values are ordered in the order they were read
FORMATTED	orders class values by formatted representation
FREQ	values are ordered by descending frequency count (class values found in greatest numbers of values occurs first)
INTERNAL	the same order PROC SORT would arrange them

PROC PRINTTO

PROC PRINTTO allows the redirection of the SAS LOG or PRINT files to a disk file. This can be useful when a PROC produces the desired values, but perhaps the report needs to be rearranged in some manner.

PROC SORT

The NODUPS and NODUPKEY options have been around for a while and can easily delete duplicate data.

NODUPS will not output a record if the entire observation matches another. This can easily

discard data that got doubled up somehow.

NODUPKEY will only output one record per BY variable and could be useful if you need one record per account for example.

With release 6.07 SORT now records how a SAS dataset is sorted so that if a later SORT is coded, SORT first checks to see if it needs to sort, and if not the unnecessary sort is skipped. This is done automatically with no work for the programmer. This is a very nice addition that saves millions of unneeded sorts worldwide, and the developers should be commended on this feature.

Along with the above, PROC DATASETS can mark a dataset as sorted even though PROC SORT didn't do the original sorting. And example might be a SAS datafile build from a raw file that was already sorted. By specifying the MODIFY SORTEDBY clause, the dataset is marked and again later unnecessary sorted will be skipped.

PROC FORMAT CNTLIN Option

Value labeling formats are certainly an integral part of SAS in earlier releases the formats had to be hard coded, or generated with macros. Quite often the data needed already exists in other files and the CNTLIN option allows FORMAT to get it's information from a SAS dataset with some special variables. In almost all cases building this SAS dataset will be simpler than a corresponding macro. There is also a CNTLOUT option if an external file is needed from the FORMAT data.

Selected SAS Options

Among the many system options I would like to just discuss a few.

ALTLOG and ALTPRINT allow for a duplicate copy of the SAS log and print files to be directed to a file. This can be useful to track online systems or in any case where an archived copy makes sense.

The SYSPARM option allows the invoking command to pass values to SAS at startup time very much like the MVS JCL PARM option. The value passed can contain up to 200 characters and it can be retrieved by the SYSPARM function or the &SYSPARM macro variable. This can be very useful when code is to be hidden or protected, but yet still allow for changes to be specified.

```
MVS // EXEC SAS,OPTIONS='SYSPARM="JANUARY"'
```

```
TSO SAS OPTIONS('SYSPARM='JANUARY'')
CMS SAS (SYSPARM JANUARY
```

```
DATA X;
  IF N = 1 THEN
    DO;
      MONTH=SUBSTR(SYSPARM(),1,8);
      RETAIN MONTH NEWYEAR;
      IF SYSPARM()='JANUARY' THEN
        NEWYEAR='YES';
      ELSE
        NEWYEAR='NO';
    END;
  . . . rest of program
```

MVS Features

The PROCs are useful if you are running SAS under MVS

PROC PDSCOPY copies and manages MVS Partitioned datasets.

PROC TAPELABEL and PROC TAPECOPY for handling tapes.

PROC SOURCE for printing and extracting PDS members to a sequential dataset and for performing mass changes.

Accessing System Control Blocks

One of the most exciting and obscure features is access to several system control blocks through the INFILE and FILE statements. Normally only assembler programs can access these blocks, and it is normally impossible for COBOL and other higher languages.

Using the JFCB

The Job File Control Block is 176 bytes of information stored for each DD card specified in the job step. This very useful block contains extensive information about each dataset including datasetname, device type, whether the dataset is catalogued, whether the dataset is a SYSIN or SYSOUT dataset, label processing options and much more. By accessing this block, the SAS DATA step can determine what dataset name was provided in the JCL. This is especially useful if the same program is run against several different datasets, and whatever dsname is provided through DD cards is to end up in a title or similar field.

A SMF processing program can determine whether it is reading a live VSAM file, a sequential backup disk file, or a tape file. Since many of the indicators in the JFCB are bit settings, the DATA step may need to do bit testing, which is no problem for the DATA step.

A JFCB example

Determine the DSNAME and DSORG from the JFCB.

```
DATA NULL_;
  INFILE IN JFCB=JFCBIN;
  LENGTH TITLDSN $ 44;
  LENGTH DSORG1 $1.;
  IF N = 1 THEN
    DO;
      TITLDSN=SUBSTR(JFCBIN,1,44);
      DSORG1=SUBSTR(JFCBIN,99,1);
      IF DSORG1='1.....'B THEN
        DSORGOUT='PS';
      END;
    INPUT etc. ;
  RETAIN TITLDSN DSORGOUT;
  RUN;
```

/* DONT NEED DATASET */
 /* ASK FOR JFCB */
 /* SET LENGTHS AS */
 /* REQUIRED */
 /* FIRST TIME IN ? */
 /* YES, DO BLOCK */
 /* EXTRACT DSNAME */
 /* AND DSORG BYTE 1 */
 /* BIT TEST AS NEEDED*/
 /* MUST BE SEQUENTIAL*/
 /* END OF BLOCK */
 /* REST OF PROGRAM */
 /* RETAIN */
 /* END OF STEP */

Other Available Control Blocks

Other MVS control blocks available on INFILE and FILE include DEVTYPE containing device type information, DSCB containing the data set control block information for non-vsam disk files, the UCBNAME containing the device address from the UCB, and the VOLUME parameter containing the volume serial of the disk dataset. Another useful INFILE option EOVS is set to 1 each time a concatenated datasets boundary is detected. The above mentioned control blocks can then be reread with the new values if desired.

Summary

While SAS is certainly a large and comprehensive system, it is quite often the little things that can make programming easier if we only know about those features.

SAS is a registered trademark of SAS Institute Inc.

The author will be glad to answer questions and accept suggestions at the following address:

Steven First
 Systems Seminar Consultants
 Suite 206
 1220 Fernrite Drive
 Madison, WI 53716

Voice: (608) 222-7081
 Fax: (608) 222-7310
 Internet: train@sys-seminar.com