Paper **PO-088**

T.I.P.S. (Techniques and Information for Programming in SAS®)

Kathy Harkins, Carolyn Maass, Mary Anne Rutkowski
Merck Research Laboratories, Upper Gwynedd, PA

ABSTRACT:

This paper provides a collection of basic programming tips and techniques that SAS® users can implement in their daily programming. This collection of tips should be useful immediately and will improve the efficiency of the programs. There are several examples organized by the following categories (Keywords: BASE STAT FUNCTIONS):

- Read and write data selectively
- Concise coding techniques
- Effective use of sorting techniques
- Data manipulation
- Macros (tips for the beginner)

READ & WRITE DATA SELECTIVELY:

Example 1:
Use the KEEP= or DROP= on the SET or MERGE statement to keep unneeded variables out of the Program Data Vector (PDV) (the storage area for variables, values, and attributes).

Acceptable:	**More Efficient:**

data small;
 set large;
 keep a b c prod lrgst;
 prod = a*b;
 lrgst = max(a,b,c);
More SAS® statements;
run;
	data small;
	 set large(**keep=a b c**);
	 prod = a*b;
	 lrgst = max(a,b,c);
	More SAS® statements;
	run;

Example 2:
Produce all subsets you require for further processing in one step to minimize the # of times a large dataset is read in.

Acceptable:	**More Efficient:**
data **one**; set large; if a < 10; run; data **two**; set large; if 9 < a < 90; run; (similar for data **three**;)	data **one two three**; set large; if a < 10 then output one; else if 9 < a < 90 then output two; else if a > 89 then output three; run;

Example 3:
Read an existing SAS® dataset and subset it based on values of one (or more) of the variables. Use a where instead of an if….

Acceptable:	**More Efficient:**
data **new**; set old; **If** age > 65 and gndr = 'F'; More SAS® statements..; run;	data **new**; set old (**where=**(age > 65 and gndr = 'F')); More SAS® statements..; run;

CONCISE CODING TECHNIQUES:

Example 1:
Execute only the necessary statements – move a "subsetting" if before statements you only want to execute after the condition is met.

Acceptable:	**More Efficient:**
data elders; set demog; age = date2 – date1; relday = date3 – date2; tot_rslt = sum(rslt1,rslt2,rslt3); More SAS® stmnts.; **if age > 65**;	data elders; set demog; age = date2 – date1; **if age > 65**; relday = date3 – date2; tot_rslt = sum(rslt1,rslt2,rslt3); More SAS® stmnts.;

run;	run;

Example 2:

Execute only the necessary statements – when "if" conditions are mutually exclusive, use if-then-else instead AND move most likely condition to top of logic.

Acceptable:	**More Efficient:**
data rsn_dscn; set pat_stat; **if** upcase(resn_dsc) in ('Reason 1', 'Reason 2') **then** code_dsc = 1; **if** upcase(resn_dsc) in ('Reason 3', 'Reason 4') **then** code_dsc = 2; **if** upcase(resn_dsc) in ('Reason 5', 'Reason 6') **then** code_dsc = 3; run;	data rsn_dscn; set pat_stat; **if** upcase(resn_dsc) in **('Reason 5', 'Reason 6') then** code_dsc = 3; **else if** upcase(resn_dsc) in ('Reason 3', 'Reason 4') **then** code_dsc = 2; **else if** upcase(resn_dsc) in ('Reason 1', 'Reason 2') **then** code_dsc = 1; run;

Example 3:

Take advantage of boolean logic [expressions evaluate to true (0) or false (1)] to assign values instead of using if-then-else logic.

Acceptable:	**More Compact:**

data survey;
set survey;
if age <= 25 then agegr = 1;
else if age <= 40 then agegr = 2;
else agegr = 3;
run;
	data survey;
	set survey;
	agegr =
	(age <= 25) +
	2*((age > 25) and
	(age <=40)) +
	3*(age > 41);
	run;

Example 4:

Use select statements instead of if-then-else when many mutually exclusive conditions exist.

Method 1 (no select expression):	**Method 2(a select expression):**

data new;
set old;
select;
when (temp < 32)
code = 1;
when (temp < 80)
code = 2;
when (temp < 110)
code = 3;
otherwise code = 4;
end;
More SAS® statements …;
run;
	data new;
	set old;
	select (cat);
	when (‘A’) code = 1;
	when (‘B’) code = 2;
	when (‘C’) code = 3;
	otherwise code = 4;
	end;
	More SAS® statements;
	run;

EFFICIENT SORTING:

Example 1:
Use the WHERE= option on the PROC SORT to reduce unnecessary observations.
NOTE: By default, the dataset in the data = option is *replaced* by the sorted version. Therefore, to keep the integrity of your original dataset, get into the habit of using the OUT = option

Acceptable:	**More Efficient:**
proc sort data = adverse_events; By intensity; Run;	**proc sort** data = adverse_events (**where** = (relation = ‘Y’)) **out** = relatedAE; By intensity; Run;

Example 2:
Use the NODUPKEY= option on PROC SORT to eliminate duplicate output observations with the same values for the BY variables.
NOTE: Useful in situations when you have multiple observations for each individual and you only want, for example, the first or last (use DESCENDING on sort) adverse event in your dataset.

Acceptable:	**More Efficient**
proc sort data = adverse_events out = lastAE; by an_num **descending** intensity; Run; **data keeplast**; Set lastAE; by an_num **descending** intensity; If first.intensity; Run;	**proc sort** data = adverse_events **NODUPKEY** **out** = lastAE; By an_num **descending** intensity; Run;

Example 3:
Use the TAGSORT= option on the PROC SORT to reduce amount of temporary disc spaced used.
NOTE: Best performance is gained with this option when the total length of the BY variables is short when compared to the record length.

Acceptable:	**More Efficient:**
proc sort data = adverse_events **out** = NoTagAE; by intensity; run;	**proc sort** data = adverse_events **tagsort** **out** = taggedAE; By intensity; Run;

Example 4:
Use the CLASS statement with your procedure (MEANS or SUMMARY)
This eliminates the need to do a PROC SORT prior to your procedure.

Acceptable:	**More Efficient:**
proc sort data = adverse_events; By intensity; Run; **proc summary** data=adverse_events; var duration; **by intensity;** output new = new MEAN = meandur; Run;	**proc summary** data=adverse_events; **class intensity;** var duration; output new = new MEAN = meandur; Run;

DATA MANIPULATION:

Example 1:
By-Merging of Sorted Datasets
Merging without a BY merges observation by observation.
NOTE: If a variable is found in both datasets, the resultant value for this variable is pulled from the LAST dataset in the list.
By-Merging of Sorted Datasets using IN=
Allows specification of which input dataset contribute to the merge.

Step 1: Sort	**Step 2: Merge**
proc sort data **=** adverse_events**;** By ***an code date***; run; **proc sort** data **=** therapy**;** By ***an code date***; run;	data both; **merge** adverse_events **(in= ina)** therapy **(in=int)** ; by *an code date*; **if ina;** run;

Example 2:
Labels
When printing the contents of a SAS® dataset created with labels, add readability to your output by printing the variable label.

Create your SAS® datase with Labels

```
Proc SQL;
  create table adverse_events (label="Adverse Event") as
  select   prsc_tm  label=     "PRESCRIBED TIME",
           strltrdy    label=     "START_DAY_REL_TO_TRIAL",
           dur_un1  label=     "DURATION UNITS",
           intn_c     label=      "INTENSITY_CODE",
  from ae;
quit;
```

Proc Print - No Labels	**Proc Print - with Labels**
proc print data = adverse_events; title 'WITHOUT Label Option'; **run**;	**proc print** data adverse_events **label**; title 'WITH Label Option'; **run**;

Example 3:
Use of Formats

Create Formats	**Print using Formats**
proc format; value codes **1** = 'High' **2** = 'Medium' **3** = 'Low'; run;	Proc print data=adverse_events **label**; **format** intn_cd codes.; Run; **Output:** WITH Format on Intensity Code PRESCRIBED_ TIME INTENSITY_ CODE Post-dose High Post-dose Medium Post-dose High

Example 4:
Renaming Variables.
Variables are renamed with either the RENAME= data set option or the RENAME statement.

RENAME= Data Set Option:	**RENAME statement:**
data two; set **one(rename=(x1=y z1=u)** **keep=y u total);** **total=y+u;** run;	data two; set one; **rename x1=y z1=u;** total=x1+z1; **keep=y u total**; run;

Example 5:
Retaining Variables.
The RETAIN statement causes a variable to retain its value from the previous iteration of the DATA step.

RETAIN with first. and last:	**RETAIN without a variable list:**

proc sort data=new; by rating; run;
data count; set new; by rating;
retain cnt 0 howmany 0;
if **first.rating** then cnt=1; else cnt+1;
if first.rating then howmany+1;
if **last.rating** then output;
run;
proc print data=countem; run;
Here is the printout of data set countem:
OBS RATING TX CNT HOWMANY
1 apparent 10 3 1
2 discrete 9 6 2
3 none 7 1 3
	data group;
	Input x;
	retain;
	if x=1 then gender='M';
	if x=2 then gender='F';
	cards;
	data lines
	;
	X is retained if value is other than 1 or 2

MACROS (tips for the beginner):

Example 1:
Creating a Macro Variable

%LET :	**CALL SYMPUT :**

%let dsn=NEWDATA;
title "Display of Data Set &DSN";
 	
TITLE "Display of Data Set **NEWDATA";**
	%macro create;
	data temp;
	set newdata end=final;
	if age >=20 then do;
	N+1;
	output;
	end;
	if final then **call symput('number',n);**
	run;
	%mend create;
	Footnote "&number Observations";

	Footnote “26 Observations”;

Example 2:
Concatenating Several datasets using a macro.

Acceptable without macro:	**Alternate Method :**

data x1; x=1; run;
data x2; x=2; run;
data x3; x=3; run;
 	
data final;
  set x1 x2 x3;
run;
	data x1; x=1; run;
	data x2; x=2; run;
	data x3; x=3; run;
	%macro test;
	data final; set
	%do i = 1 %to 3;
	  x&i
	%end;
	 run;
	%mend test;
	%test

Example 3:
Generating a series of DATA steps

Invoke macro CREATE:	**Produces:**

%macro create;
%do i=1 %to 3;
  data month&I;
  infile in&I;
  input product cost date;
  run;
%mend create;
%create;
	data month1;
	  infile in1;
	  input product cost date;
	run;
	
	data month2;
	  infile in2;
	  input product cost date;
	run;
	
	data month3;
	  infile in3;
	  input product cost date;
	run;

Example 4:
Comment out code using a macro or use “HOT” key:

“HOT” Key Method :	**Macro Method :**
Use the “HOT” key **cntl+/** to comment selected code. /*data small.subset;*/ /*set big.dataset; /*if mod(_n_, 50) = 28;*/ /* select every 50th record */ /*run;*/	**%macro skipstep;** data small.subset; set big.dataset; if mod(_n_, 50) = 28; /* select every 50th record*/ run; **%mend skipstep;**

REFERENCES:

1. In the Know…SAS® Tips & Techniques From Around the Globe, Phil Mason
2. SAS® Guide to Macro Processing, SAS Institute, Inc.
3. SAS® User's Guide: Basics, SAS Institute, Inc.

CONTACT INFORMATION:

Kathy Harkins
Merck & Co. Inc.
PO Box 1000
North Wales, PA 19454-1099
267-305-5533
kathy_harkins@merck.com

Carolyn Maass
Merck & Co. Inc.
PO Box 1000
North Wales, PA 19454-1099
267-305-7145
carolyn_maass@merck.com

Mary Anne Rutkowski
Merck & Co. Inc.
PO Box 1000
North Wales, PA 19454-1099
267-305-6925
mary_anne_rutkowski@merck.com