The RTSUG Consultant's Panel Discussion

moderated by Frank DiIorio

Background

One of the aspects about programming in general and SAS programming in particular that has always intrigued me is how people arrive at a solution. That is, if you pose a question to "n" SAS programmers, you'll have at least "n" workable and correct solutions. If you take this a step further and say "how would *experienced* SAS programmers differ in their approaches to the same problem" the results would, I think, be even more interesting.

To that end, I cajoled three local SAS consultants to engage in an electronic panel discussion. My thanks to:

- Ben Cochran, The Bedford Group
- Bob Passmore, Passmore Consulting Services
- Jack Shoemaker, Thotwave Technologies

for taking time from their hectic schedules to participate.

Here's how it worked. I sent an initial, deliberately vague (meaning, "real world") specification for a simple macro. This first go-round was designed to solicit questions from the participants. Their questions, along with my original intent for the macro, led to Part Two, The Revised Spec. This is more or less what, as a programmer, you'd like to see as a guideline for your work – specific, but not micro-managing. This is what guided the actual coding of the macro. Once I had the completed macros in hand, I circulated them to all panelists and solicited their comments.

With the exception of some reformatting for publication purposes, all communications are reproduced as-is. I've added a couple of my own comments at the end.

Original Spec

"Write a macro that counts the number of observations in a SAS data set."

Questions About the Original Spec

"Why do you need the macro? How will it be used? (Can you give some examples?)"

"So many different situations could be behind such a request. Depending on the response, I might question whether a Macro is really necessary (is it overkill for a limited necessity?), or see a need to explore how many different contexts the requested macro may need to be used (test whether the named table exists, ask how they understand the phrase "SAS data set", etc.)

What is to be done with the information obtained?

Is this requirement for use in a specific program? Or shared by several?

How "fool proof" does the macro need to be (dataset name does not exist, or is null, or "nobs" cannot be obtained)?"

The Revised Spec

Write a macro that counts the number of observations in a data set. The macro does not need to be part of a DATA step (therefore it can consist of standalone DATA step / PROC/etc.). The macro is intended to be general-purpose, but would likely be used most frequently for branching within another macro, along the lines of:

```
... calling macro ...
%Counter(data=new.v2)
%if &count. 0 %then %do;
    ... stuff that happens if found and with obs ...
    %end;
    %else %if &count. = 0 %then %do;
          ... stuff that happens if found but empty ...
          %end;
    %else %if &count. < 0 %then %do;
          ... stuff that happens if data set is not found ...
          %end;
```

Inputs:

- Data set name. Required. One or two-level name. If one, assume LIBNAME of WORK.
- Macro variable holding results. User-specified global macro variable. If not specified, use COUNT.

Output:

- Value written to the variable specified in input [2], above.
- Message to the SAS Log indicating: data set name, macro variable name, and macro variable value.

Other:

- If data set is not found, set the value of the macro variable to -1.

"Panelist #1" Ben Cochran

```
%macro obs_tot(dsn);
    %global count;
    %let dsid = %sysfunc(open(&dsn));
    %if &dsid %then
       %do;
          %let count =%sysfunc(attrn(&dsid, NOBS));
          %let rc    =%sysfunc(close(&dsid));
          %put &dsn has &count observation(s).;
       %end;
    %else
       %do;
          %put Open for data set &dsn failed - %sysfunc(sysmsg());
          %let count = -1;
          %put count = &count;
       %end;
%mend obs_tot;
%obs_tot(sasuser.class);
```

Comments

from Jack Shoemaker

"This is a neat and compact solution. The attrn() function is a nice non-data-step way to grab the number of observations in a SAS data set. In fact, Ben's solution is pure macro code. Closing the data set with the close() function is a nice touch. Obviously, Ben has good programming habits. Notwithstanding, I am surprised to not see any comments. The macro unconditionally places the result in the global macro variable COUNT. This is only a problem if COUNT already exists in the calling environment and should not be clobbered. A second macro parameter containing the name of the macro symbol to hold the result is a better solution."

from Bob Passmore

"Simple and elegant... it provides the basic functionality requested, returning a message to the log and placing the number of obs. result in a macro variable "count". It's short, and easy for a client just learning SAS to follow. It does not, however, follow some of the particular specifications given. For instance, no alternative name can be specified for "count" in %obs_tot(...). Providing two different names helps when verifying that two datasets exist before encountering the %if ... %then clause in a macro program.

A nice touch: `%sysfunc(sysmsg())` provides a useful message using one simple _expression!

An obscure technical issue: If &dsn is null, `%sysfunc(open(&dsn))` can report a value > 0 in perhaps rare circumstances. To avoid the possibility of returning COUNT>0 when no dataset name was given, the macro could first test "`%if %length(&dsn)>0...`".

"Panelist #2" Jack Shoemaker

```
%macro Counter(
  DATA=,
  OBSVAR=COUNT
  );
/* ------------------------------------------------
 * RTCount by Jack Shoemaker on 29AUG2003
 * Updated 08SEP2003 and re-named Counter
 *
 * Usage
 *   %Counter(DATA=your.dataset,OBSVAR=YourMacroSymbol)
 *   Only works as stand-alone statement
 *
 * Parameters
 *   DATA - name of SAS data set
 *   OBSVAR - name of macro symbol to store count
 *          - default is COUNT
```

```
 *
 * Brief description
 *   Counts observations in SAS data set
 *   Places value in macro symbol specified by OBSVAR
 *   If data set not found, value is -1
 *
 * Notes (small-print department)
 *   Only works with SAS data sets.
 *   If COUNT already exists in GLOBAL environment,
 *   it gets clobbered.
 * ------------------------------------------------- */
 /* Place desired macro symbol in global environment */
  %global &OBSVAR;
  %local DSID M;
 /* Upcase data set name for LOG display */
  %let DATA = %upcase(&DATA);
  / -------------------------------------------------
  * Determine if data set exists
  * DSID will be 0 if data set does not exist
  * ------------------------------------------------- */
  %let M = I;
  %let DSID = %sysfunc( open( &DATA, &M ) );
  /* If data set not found, then set return value to -1  */
  %if &DSID = 0 %then %do;
    %let &OBSVAR = -1;
  %end;
  /* -------------------------------------------------
  * Otherwise use NOBS= option to obtain obs count
  * Use NOBS= option to count obs
  * NOBS= is a compile-time option, so no need to
  * execute the SET statement.  'if 0' always evaluates
  * to FALSE, so SET statement is never executed
  * ------------------------------------------------- */
  %else %do;
  data _null_;
    if 0 then set &DATA. nobs = nobs;
    call symput( "&OBSVAR", left( nobs ) );
    stop;
    run;
  %end;
 /* Write results to SAS log                          */
  data _null_;
    c = &&&OBSVAR;
    str = "Value of &OBSVAR. is " || trim( left ( put( c, best32. ) ) ) || ".";
    put "COUNTER NOTE:  Counting rows in &DATA..";
    put "               Value will be stored in &OBSVAR..";
    put "              " str;
    stop;
    run;
%mend Counter;
data x;
  x = 1;
  run;
%Counter( DATA=x );
%put COUNT is &COUNT.;
%Counter( DATA=x, OBSVAR=xyz )
%put COUNT is &XYZ.;
%Counter( DATA=y, OBSVAR=def )
%put COUNT is &DEF.;
```

Comments:

from Jack Shoemaker

"I would improve this macro in two ways. First, the header comments should be revised to reflect the revisions. Second, I'd use the attrn() and close() functions as Ben did in his examp le."

from Bob Passmore

This solution demonstrates the use of standards for writing SAS macros:

- It is well documented, and evidently follows a documentation style.

- Comments in the code clearly explain how it is used, including "only works with SAS datasets".
- Macro arguments appear one per line for clarity

It also implements the request that the counter variable name can be optionally defined on invocation (some lines on log removed....):

```
82   %counter(data=test,obsvar=nobs);

COUNTER NOTE:  Counting rows in TEST.
               Value will be stored in nobs.
               Value of nobs is 1.
NOTE: DATA statement used:
      real time           0.00 seconds
      cpu time            0.00 seconds

83   %put &nobs;
```

The strategy to use a non-executing "data _null_" step to obtain the count from the NOBS parameter on the SET statement differs from the "`%sysfunc(attrn(&dsid,NOBS)`" method chosen by the other two panelists. %Counter returns a result>0 in circumstances where the other two determine that the NOBS statistic is not available. Below is a simplified SAS Log at the end of which an executed data step displays the number of records copied from the SAS View:

```
186    %counter(data=SASHELP.VCOLUMN);      /* #2 */

COUNTER NOTE:  Counting rows in SASHELP.VCOLUMN.
               Value will be stored in COUNT.
               Value of COUNT is 2147483647.

187    %obscount(data=SASHELP.VCOLUMN);     /* #3 */
Note: Engine cannot determine observation count for SAS view SASHELP.VCOLUMN.
      Count=-1.
188   %put %obs_tot(sashelp.vcolumn);      /* #1 */
sashelp.vcolumn has -1 observation(s).

189   data vcol;
190        set sashelp.vcolumn (keep=libname memname name);
191        run;
NOTE: There were 859 observations read from the data set SASHELP.VCOLUMN.
NOTE: The data set WORK.VCOL has 859 observations and 3 variables.
```

Determining the observation count of a SAS View is out of the scope of the assignment, so we may excuse all three panelists for failing to prepare for such a request. Indeed, a comment in #2 clearly states this limit. #1 (and #3) demonstrate a different choice: returning an error code for a non-SAS dataset reference rather than issuing a misleading count. But users may not have carefully read the instructions, or, not knowing the difference between types of tables may invoke the utility anyway. Whether a program should "anticipate" out-of-scope usage and make the effort to issue error return codes or warning messages is an interesting topic for discussion! The cost is in the extra design and testing.

To summarize: While not designed to return obs count from SAS views, %Counter does return correct values in some "out-of-scope" cases. However, it does not determine which views provide such metadata to the NOBS parameter, and can return an incorrect count instead of a -1.

Like #1, this macro does not check for a null instance of &data, but assumes that open() will return a 0 value.

%Counter is very servicable utility program for the client's macro library which meets all the stated requirements.

"Panelist #3" Bob Passmore

```
%*    ObsCount.SAS    A macro program that returns a macro variable containing the
                      number of observations in named a SAS dataset/view.  Value is
      -1 if no dataset/view is found or if obs. count cannot be determined (view).

      Syntax:     %obscount(data=data=SAS_DATASET_NAME  {,obsrc=COUNT_VAR_NAME} ):,

                  Default COUNT_VAR_NAME is "count".  If "data" is not given, a
                  short explanation is written to the log.

      Usage:      May be run interactively.  Intended for use within a SAS %macro
                  program.  Program logic can branch depending on value of &count.

      Author:     Bob Passmore, Passmore Consulting Services, Durham, NC  Sep 2003.
                  Prepared at the request of Frank DiIorio for use by a SAS users group.
;
%macro obscount(data=,obsrc=Count);
```

```
%global &obsrc;
%local dsid dsnlen dstype objtype word1 word2 _anobs _nobs _nvars;

%*   Assign value to return code indicating no observation count can be reported.
;
%let &obsrc=-1;

%let dsnlen=%length(&data);
%if &dsnlen=0 %then %do;
    %put Note: No dataset name has been passed to %nrstr(%obscount).;
    %put ;
    %put %nrstr(      Syntax:  %obscount(data=SAS_DATASET_NAME););
    %put %nrstr(               Observation count is returned as &count.);
    %put %nrstr(               If count=-1, no such dataset/view or count unavailable.);
    %put ;
    %goto exit;
    %end;

%*  Standardize DSN as 2-part SAS name.
;
%let word1=%sysfunc(scan(&data,1));
%let word2=%sysfunc(scan(&data,2));
%if .&word2=.
    %then %let dsn=work.&word1;
    %else %let dsn=&word1..&word2;

%*  Determine whether Data or View "&dsn" exists.
;
%let dstype=none;
%if %sysfunc(exist(&dsn,data)) %then %let dstype=data;
%else
%if %sysfunc(exist(&dsn,view)) %then %let dstype=view;
%if &dstype=none %then %do;
    %put Note: Item "&dsn" does not exist as a SAS dataset or SAS view.;
    %goto exit;
    %end;

%*  Determine observation count for "&dsn".
;
%let dsid=%sysfunc(open( &dsn(type=&dstype) ));
%if &dsid>0 %then %do;
    %let _anobs=%sysfunc(attrn(&dsid,anobs));
    %let _nobs=%sysfunc(attrn(&dsid,nobs));
    %let _nvars=%sysfunc(attrn(&dsid,nvars));
    %end;
%let dsid=%sysfunc(close(&dsid));

%*  Report results.
;
%if &dstype=data
    %then %let objtype=SAS dataset;
    %else %let objtype=SAS &dstype;
%if &_anobs=0 %then %do;
    %put Note: Engine cannot determine observation count for &objtype &dsn..;
    %goto exit;
    %end;

%put Note: &objtype &dsn has &_nobs observations and &_nvars variables.;
%let &obsrc=&_nobs;

%exit:
%put %str(      &obsrc=&&&obsrc...);
%mend;
```

Test program

```
%put %obscount;
%put %obscount();

%put %obscount(data=sashelp.company);
%put %obscount(data=sashelp.country);
data test;
run;
%put %obscount(data=test,obsrc=nobs);
%let Count=Empty;
%put %obscount(data=sasmacr);
```

```
%put %obscount(data=sashelp.company);
%put %obscount(data=sashelp.table);
%put %obscount(data=SASHELP.VCOLUMN);

proc access dbms=xls;
  create saslib.children.access;
   path='c:\Client Projects\DMM\Data\children.xls';
   scantype=yes;
   getnames=yes;
   assign=yes;
   list all;
  create saslib.children.view;
   select all;
run;
%put %obscount(data=SASLIB.children);
```

Log for test program

```
295  %put %obscount;
Note: No dataset name has been passed to %obscount.

      Syntax:  %obscount(data=SAS_DATASET_NAME);
               Observation count is returned as &count.
               If count=-1, no such dataset/view or count unavailable.

      Count=-1.

296  %put %obscount();
Note: No dataset name has been passed to %obscount.

      Syntax:  %obscount(data=SAS_DATASET_NAME);
               Observation count is returned as &count.
               If count=-1, no such dataset/view or count unavailable.

      Count=-1.

297
298  %put %obscount(data=sashelp.company);
Note: SAS dataset sashelp.company has 48 observations and 8 variables.
      Count=48.

299  %put %obscount(data=sashelp.country);
Note: Item "sashelp.country" does not exist as a SAS dataset or SAS view.
      Count=-1.

300  data test;
301  run;

NOTE: The data set WORK.TEST has 1 observations and 0 variables.
NOTE: DATA statement used:
      real time           0.01 seconds
      cpu time            0.01 seconds


302  %put %obscount(data=test,obsrc=nobs);
Note: SAS dataset work.test has 1 observations and 0 variables.
      nobs=1.

303  %let Count=Empty;
304  %put %obscount(data=sasmacr);
Note: Item "work.sasmacr" does not exist as a SAS dataset or SAS view.
      Count=-1.

305  %put %obscount(data=sashelp.company);
Note: SAS dataset sashelp.company has 48 observations and 8 variables.
      Count=48.

306  %put %obscount(data=sashelp.table);
Note: SAS dataset sashelp.table has 8 observations and 21 variables.
      Count=8.

307  %put %obscount(data=SASHELP.VCOLUMN);
Note: Engine cannot determine observation count for SAS view SASHELP.VCOLUMN.
      Count=-1.

308
```

```
309  proc access dbms=xls;
310    create saslib.children.access;
311     path='c:\Client Projects\DMM\Data\children.xls';
312     scantype=yes;
313     getnames=yes;
314     assign=yes;
NOTE: ASSIGN names is now ON.
315     list all;
Excel File: C:\CLIENT PROJECTS\DMM\DATA\CHILDREN.XLS Version: 4
Function: CREATE  Descriptors- access: CHILDREN  view:
Item   Column Label                SAS Name Format
 1     GRADE                       GRADE    BEST8.
 2     CHILD NAME                  CHILDNAM $31.
 3     FAMILY                      FAMILY   $18.
 4     ID                          ID       BEST5.
 5     PARENTS                     PARENTS  $32.
 6     PHONE                       PHONE    $13.
 7     BIRTHDAY                    BIRTHDAY MMDDYY8.
 8     LAST UPDATE                 LASTUPDA $9.
316    create saslib.children.view;
NOTE: The access descriptor SASLIB.CHILDREN was written.
317     select all;
318  run;

NOTE: The view descriptor SASLIB.CHILDREN was written.
NOTE: PROCEDURE ACCESS used:
      real time           0.01 seconds
      cpu time            0.01 seconds


319  %put %obscount(data=SASLIB.children);
Note: SAS view SASLIB.children has 124 observations and 8 variables.
      Count=124.
```

Comments:

from Jack Shoemaker

"Bob's solution provides much more robust messaging. I particularly like the usage note if the macro is called with less than the required parameters. I don't think that &DSN needs parsing. That is, neither &WORD1 nor &WORD2 are referenced in the subsequent code."

from Bob Passmore

"This response illustrates some choices that might be made if one decides the program should "anticipate" its actual range of use. It was therefore designed to obtain the NOBS value from any table for which NOBS data are available.

It also follows a modest set of standards. One of the expectations it demonstrates is that a user should be able to display the command syntax by "running" the command (analogous to entering "commandname /?" in a DOS window)."

Moderator's Comment

The solutions that were offered were clean and robust. What surprised me, though, was that none of them used SAS's meta data, held in this case in DICTIONARY.TABLES. What follows is a utility that I've used over the years. It is, perhaps, not as fast as solutions using the SCL functions (OPEN, ATTRN, etc.), but it's simple to code and is a model for more complex utilities using the dictionary tables. Due to the context in which it is usually used, its error-checking is minimal – it assumes that the DATA parameter was coded (it could have defaulted to &SYSLAST).

```
%macro countobs(data=, count=_count_);
       %global &count.;

       %let data = %upcase(&data.);
       %if %index(&data., .) > 0 %then %do;
           %let libname = %scan(&data., 1, .);
           %let memname = %scan(&data., 2, .);
           %end;
           %else %do;
                 %let libname = WORK;
                 %let memname = &data.;
                 %end;

       proc sql noprint;
            select nobs into :&count
            from dictionary.tables
```

```
            where libname="&libname." & memname="&memname." & memtype=”DATA” ;
        quit;

        %if &sqlobs. = 0 %then %let &count. = -1;
        %put COUNTOBS-> Macro variable %upcase(&count.) = %left(&&&count);
%mend;
```