Paper 223-25

Workarounds for SASWare Ballot® Items

Jack Hamilton, First Health, West Sacramento, California USA

ABSTRACT

As part of its effort to insure that SAS Software is useful to its users, SAS® Institute collects suggestions for changes and enhancements and asks users to rank them in importance. This paper presents workarounds for several SASWare Ballot items.

UPDATES

I did not have access to Version 8 when I wrote this paper. After V8 becomes available, I will revise this paper as needed. I will also incorporate comments received at SUGI. The new version will be available on my web page at the address shown below. You will be able to register for notification of future updates.

INTRODUCTION

SASWare Ballot items often ask for a way to do something that SAS Software can already do. Sometimes the existing procedure is not obvious, or is hidden in a Changes and Enhancements document, or requires macro coding. This paper presents a few workarounds for requests on the SASWare Ballot.

The number after a request title is the year in which the item appeared on the SASWare Ballot.

NOT COVERED IN THIS PAPER

Literally dozens of items on past years' ballots have been addressed by the Output Delivery System. One such item is item #1 on the 2000 SASWare Ballot - "provide a system option to control the number of decimal places printed for any output". Such items are not addressed here.

Also not addressed here are items which are directly answered in the Changes and Enhancements documentation.

There are also many other items which could have been addressed given more space.

GENERAL

PROVIDE THE ABILITY TO DISPLAY THE CURRENT DATE AND TIME IN THE SAS LOG AND OUTPUT (NOT JUST THE DATE AND TIME THE SESSION BEGAN) FOR AN INTERACTIVE SAS SESSION (1998)

A partial workaround is to code:

```
OPTIONS NODATE;
TITLE "%SYSFUNC(datetime(), datetime.)";
```

The SYSFUNC macro is documented in *Macro Facility Enhancements to Release 6.09E and Release 6.11.* It allows you to use a data step function in the macro language, and format the result. In this case, the DATETIME function is used to obtain the current date and time as a SAS datetime value; the value is then run through the DATETIME. function.

An advantage to this method is that you can specify any date, time, or datetime format that you want, or combinations of functions. The primary disadvantage is that you cannot place the resulting date into the default location of the system date; it must go into a title or footnote location.

The following code places both the English and Italian dates into the title line:

```
TITLE "%SYSFUNC(date(), date9.)) /
%SYSFUNC(date(), itadfde9.))";
```

The result is

```
20JAN2000 / 20Gen2000
```

PROVIDE THE ABILITY TO DISPLAY THE CURRENT DATE AND TIME IN THE SAS LOG AND OUTPUT (NOT JUST THE DATE AND TIME THE SESSION BEGAN) FOR AN INTERACTIVE SESSION (1998)

The solution above addresses this problem; the datetime function returns the current data and not, not the date and time at the beginning of the session.

Note that you must reissue the TITLE statement before each PROC or data step.

PROVIDE A SYSTEM OPTION TO RESET ALL OPTIONS BACK TO THE DEFAULT SETTING AT INITIALIZATION OF THE SAS SYSTEM (1998)

A general solution is complicated in version 6, but if you want to save only certain settings, and those settings are available through the new GETOPTION function, you can write a pair of macros or data steps to save and restore values.

Here's a data step solution:

```
options ls=132;

/* Save options.  */
data saveopts;
   length optvalue $200.;
   optvalue = getoption("linesize",
                        "keyword");
   output;
   optvalue = getoption("pagesize",
                        "keyword");
   output;
run;

/* Do some stuff here that might change LS
   and PS. */
options ls=64 ps=30;
title 'Small page';
proc print data=sasuser.iris (obs=10);
run;

/* Restore the settings.  */

data _null_;
   set saveopts;
   call execute('options ');
   call execute(optvalue);
   call execute(';');
run;

/* The following step will run with the
   original settings.  */
title 'Big page';
proc print data=sasuser.iris (obs=10);
run;
```

This produces an extra data set and messages in the log. A macro solution might be cleaner. The one shown below is simple; it saves and restores only three specified values:

```
%macro saveopts;

   %global saveopts;
   %let saveopts =
      %sysfunc(getoption(ls, keyword))
      %sysfunc(getoption(ps, keyword))
      %sysfunc(getoption(center, keyword));

%mend saveopts;

%macro restopts;

   options &SAVEOPTS.;

%mend restopts;

/* Set some options. */
options ls=132 ps=60 center;

/* Save them.  */
%saveopts;

/* Change values. */
options ls=64 ps=30 nocenter;

title 'Small page';
proc print data=sasuser.iris (obs=10);
run;

/* Restore original values. */
%restopts;

title 'Original page';
proc print data=sasuser.iris (obs=10);
run;
```

The macros could be much more elaborate. You could, for example, pass the names of the options you want to save as a parameter. You could also specify the name of the option into which values would be saved, which would allow you to switch back and forth between several different sets of options.

In version 8, as I understand it, this problem will be much easier to solve.

ALL PROCEDURES

PROVIDE AN OPTION TO PRODUCE A HEADER LINE WHEN THE INPUT DATASET CONTAINS NO OBSERVATIONS (1999, 1998)

This is one of the most commonly asked questions on the SAS-L email list. This solution is simple and will handle almost all data sets correctly. It will also handle WHERE clauses:

```
%macro prnt0obs
       (data=_last_,
        msg="No items to display.",
        line=5,
        column=5);

 data _null_;
    call execute('data _null_; file
                                print;');
    call execute("put #&line. @&column. "
                 || quote(&msg.) || ';');
    set &data.;
    call execute('run cancel;');
    stop;
 run;
 * Inserted code will be executed here. ;
 run;

%mend prnt0obs;
```

You might use it like this:

```
title 'Iris Data';
proc print data=sasuser.iris
                (where=(sepallen=0));
%prnt0obs( data=sasuser.iris
                (where=(sepallen=0)) );
```

There will be no observations in SASUSER.IRIS fulfilling the WHERE clause, so this code will be generated:

```
data _null_; file print;
put #5 @5 "No items to display.";
```

A page will be printed in the log telling you that no observations were available to print. Note that the set up for the PROC PRINT will be used for on this page (I consider that to be a good feature).

If the WHERE clause selects some observations, this code would be generated:

```
data _null_; file print;
put #5 @5 "No items to display.";
run cancel;
```

The run cancel statement will stop the data step from executing, and the "no items" page will not print.

A more common solution to this type of problem is to use the NOBS= option of the SET statement to obtain the number of observations; if NOBS = 0, then your print a message. There are two problems with that method: it doesn't work correctly with data sets that have deleted observations, or with any kind of view; and it doesn't handle WHERE clauses. Using the observation count from DICTIONARY.TABLES or SASHELP.VTABLE has similar problems.

Another alternative is to use the data set information function ATTRN to get the number of logical observations (NLOBS). This works, but doesn'thandle WHERE clauses. You also have to check whether the data set is a real data set or a view; if it's a view, you still have to iterate though the observations. That approach is a lot more work with probably no gain in execution speed.

DATA STEP

PROVIDE A FUNCTION TO RETURN THE VALUE OF A VARIABLE, GIVEN THE VARIABLE NAME AS A CHARACTER STRING (2000)

If you know whether the variable in question is numeric or character, you can accomplish this indirectly by setting up an array consisting of all variables of its type, then iterating through the array until you find the name you want:

```
76   data _null_;
77
78      set sasuser.iris (obs=3);
79
80      array _numvar _numeric_;
81      length _varname $8.;
```

```
82
83      do over _numvar;
84         call vname(_numvar, _varname);
85         if _varname = 'SPEC_NO' then
86            do;
87            put _numvar=;
88            leave;
89            end;
90      end;
91
92   run;

SPEC_NO=1
SPEC_NO=3
SPEC_NO=2
```

Note that the IF statement in line 85 isn't restricted to a strict equality test; you could also choose to print all variables whose names contain a particular string, or you could chose the variable to print based on the value of some other variable in the observation. If you plan to print more than one variable, you should delete the LEAVE statement.

PROVIDE SUPPORT FOR THE LIKE, CONTAINS, AND BETWEEN-AND OPERATORS FOR THE IF STATEMENT (1998)

CONTAINS can be accomplished with the INDEX function; the following two data steps produce equivalent results:

```
80   data test1;
81      set sasuser.feeder;
82      where pmenu contains 'sasdesk';
83   run;

NOTE: The data set WORK.TEST1 has 5
observations and 8 variables.

84   data test2;
85      set sasuser.feeder;
86      where index(pmenu, 'sasdesk') > 0;
87   run;

NOTE: The data set WORK.TEST2 has 5
observations and 8 variables.
```

BETWEEN-AND can be accomplished with compound operators; the following two data steps produce equivalent results:

```
112  options nostimer nofullstimer;
113  data;
114     set sasuser.iris;
115     where sepallen between 50 and 60;
116  run;

NOTE: The data set WORK.DATA1 has 67
observations and 5 variables.

117  data;
118     set sasuser.iris;
119     if 50 <= sepallen <= 60;
120  run;

NOTE: The data set WORK.DATA2 has 67
observations and 5 variables.
```

LIKE can be accomplished with the new regular expression functions. These functions provide much more functionality than LIKE, but can be quite complicated to use. The following two data steps produce equivalent results:

```
177  data;
178     set sashelp.manage;
179     where keywords like '%male%er%';
180  run;

NOTE: The data set WORK.DATA12 has 5
observations and 7 variables.

181  data;
182     retain rx; drop rx;
183     if _n_ = 1 then
184        rx = rxparse('`: "male" : "er"
:');
185     set sashelp.manage end=eodata;
186     if rxmatch(rx, keywords) > 0;
187     if eodata then
188        call rxfree(rx);
189  run;

NOTE: The data set WORK.DATA13 has 5
observations and 7 variables.
```

Regular expressions are too complicated to discuss here. The documentation is no longer available for version 6, but is the same as in version 7. For an overview, see Mike Rhoads paper from SUGI22, *Some Practical Ways to Use the New SAS Pattern-Matching Functions*.

Unfortunately, the regular expression functions were implemented in a way that makes them unavailable in WHERE clauses.

ALLOW THE IN OPERATOR TO ACCEPT EXPRESSIONS AND VARIABLE NAMES (1998)

A workaround in the data step is to use the SELECT statement instead. Instead of using

```
if x in (a, b) then
   put 'hello';
```

(which doesn't compile), use

```
select (x);
   when (a, b)
      put 'hello';
   otherwise
end;
```

PROC CATALOG

ALLOW THE USE OF THE _ALL_ KEYWORD ON THE DELETE STATEMENT WITH THE APPROPRIATE ENTRY TYPE TO DELETE ALL MEMBERS WITH THE SPECIFIED ENTRY TYPE (1999)

What is wanted here is the ability to code something like:

```
proc catalog;
   delete _all_ / entrytype=format;
run;
```

You can't do that, but you can use the system table DICTIONARY.CATALOG to prepare a list of entries to delete:

```
proc format;
   value a
      1 = 'a'
      2 = 'b';
   value b
      3 = 'x'
      4 = 'y';
run;

%let entries = ;

proc sql noprint;

   select  objname
   into    :entries separated by ' '
   from    dictionary.catalogs
   where   libname = 'WORK'
           and memname = 'FORMATS'
           and memtype = 'CATALOG'
           and objtype = 'FORMAT';

quit;

proc catalog catalog=work.formats;
   delete &ENTRIES. / entrytype=format;
   run;
quit;
```

A data step solution using SASHELP.VCATALG is also possible:

```
data _null_;

   set sashelp.vcatalg
          (where=(libname='WORK'
                  and memname='FORMATS'
                  and memtype='CATALOG'
                  and objtype='FORMAT'))
          end=enddata;

   if _n_ = 1 then
      call execute('proc catalog
catalog=work.formats;');

   call execute('delete ' || objname || '/
et=format;');

   if enddata then
      call execute('run; quit;');

run;
```

In my tests, the data step solution was substantially slower than the SQL solution (over a minute to execute vs. less than a second). Either solution can be embedded into a macro, and the data step solution translates fairly easily into a pure macro solution using the data set functions (it would probably still be slow – I'd use SQL and live with some messages in the log).

PROC SORT

ADD AN OPTION TO OUTPUT OBSERVATIONS DELETED BY THE NODUP OR NODUPKEY OPTION TO AN ALTERNATE DATA SET (1999, 1998)

Since the NODUP option does not work correctly anyway, adding an option to output the deleted observations would not be especially useful anyway (see my SUGI 25 poster *The Problem With NODUPLICATES*). This workaround uses PROC SORT followed by a datastep with FIRST. and LAST.

```
data test;
   input A B $ @@;
cards;
1 A 2 B 2 A 3 C 3 X
4 D 1 A 3 C 3 C 5 E
run;

proc sort data=test;
   by b a;
run;

data nodups
    dups;
   set test;
      by b a;
   if first.a then
      output nodups;
   else
      output dups;
run;
```

PROVIDE THE ABILITY TO SPECIFY A FUNCTION IN THE BY STATEMENT AND SORT BY ITS RESULT (1998)

This capability would be used to sort a data set according to some value which is not in the data set, but which can be calculated from variables in the data set. It's easy to do this with PROC SQL.

This example creates a small data set and then sorts it by the absolute value of variable A:

```
data test;
   input A B $ @@;
cards;
1 A -2 B 2 C 3 D -3 E
run;

proc sql;
   create table test as
      select  *
      from    test
      order   by abs(a);
quit;
```

The results:

```
OBS     A    B
 1      1    A
 2      2    C
 3     -2    B
 4     -3    E
 5      3    D
```

REFERENCES

SAS Institute Inc., *Proceedings of the Twenty-Second Annual SAS Users Group Internation Conference*, Cary, NC: SAS Institute Inc.

TRADEMARK CITATION

CONTACT INFORMATION

Jack Hamilton
First Health
750 Riverpoint Drive
West Sacramento, California 95605
JackHamilton@FirstHealth.com

http://www.qsl.net/kd6ttl/sas/sas.htm

Selected papers are also available at:

http://www.sashelp.com/Articles/ViewPapers.asp