

MACARRAY: A Tool to Store Dataset Names in a Macro "Array"

Frank C. Dilorio, ASG, Inc., Cary, NC

INTRODUCTION

The setting. One of the major strengths of the SAS macro language is its ability to make code compact and to allow conditional execution of partial, complete, or multiple program statements. It is also characterized by its relatively limited vocabulary. Notable among the omissions is array processing. This shortcoming is addressed by a macro described in this paper.

The application. Some applications call for an activity such as printing or descriptive statistics, to be performed on every dataset in a library. A simple, reliable approach to this problem is to use the dictionary table TABLE to generate what is, in effect, an array of macro variables containing the library's dataset names. This paper describes the MACARRAY macro. It has the essential elements of a "real" array: one value containing the number of elements and an arbitrary number of actual "array" elements.

The logic behind the macro's organization is discussed first, followed by an example of its use.

THE MACRO

The complete macro is presented in Appendix A. It is broken into several pieces here for the sake of explanation.

First, we define the macro, giving it a name and identifying one keyword parameter (the LIBNAME of the SAS dataset library of interest).

```
%macro macarray(libname=);
```

A comment describes the output created by the macro and gives an example of its use.

```
* MACARRAY
  Creates macro variables DS1-DSn and DSCOUNT
  - names and count of all datasets pointed to
  by parameter LIBNAME. Also creates DSL1 to
  DSLn - these contain the dataset labels.
```

```
Example: dsmacvar(libname=study1)
  If datasets BASELINE, AE, and SUMMARY
  were in library STUDY1, DSMACVAR would
  assign a value of 3 to DSCOUNT and would
  assign AE, BASELINE, and SUMMARY to macro
  variables DS1, DS2, and DS3,
  respectively.
```

```
Note: The macro assumes that the LIBNAME has
      been allocated. If not, DSCOUNT
      returns a value of 0.
```

```
Author: Frank Dilorio, ASG, Inc., 1996
```

```
*** Copy freely so long as authorship
      is cited! ***
```

The "array" size is made global so other macros will have it available. DSCOUNT is the number of datasets found in the library identified by the user. If no datasets are found, DSCOUNT will equal 0.

```
%global dscount;
```

Run the SQL procedure. We create dataset _NAMES_, reading from the dictionary tables. _NAMES_ holds, among other items, the name and label of the datasets in the library. The SELECT statement counts the number of observations in _NAMES_ and stores it in macro variable DSCOUNT. If _NAMES_ is empty DSCOUNT is 0.

```
proc sql noprint;
  create table _names_ as
    select memtype, libname, memname, memlabel
    from dictionary.tables
    where memtype = 'DATA' &
          libname = "%upcase(&libname.)";
  select nobs into :dscount
    from dictionary.tables
    where memtype = 'DATA' & libname = "WORK" &
          memname = '_NAMES_';
quit;
```

If we have no datasets, branch to the bottom of the macro.

```
%if &dscount. = 0 %then %goto %bottom ;
```

Make the "array" elements global, for the same reason as the GLOBALING of DSCOUNT, above.

```
%do i = 1 %to &dscount.;
  %global ds&i dsl&i;
%end;
```

Write dataset names and labels to macro variables DSx and DSLx, respectively.

```
data _null_;
  set _names_;
  call symput("ds" || left(trim(put(_n_,3))),
    memname);
  call symput("dsl" || left(trim(put(_n_,3))),
    memlabel);
run;
```

Before exiting, delete _NAMES_.

```
proc delete data=_names_;
run;
```

The following is optional. It will simply display the number of datasets, followed by each dataset name and label.

```
%put %left(&dscount.) Datasets: ;
%do i = 1 %to &dscount.;
  %put Dataset &i.: &ds&i has label &dsl&i;
%end;

%bottom: ;
%mend;
```

EXAMPLE

While the macro doesn't produce a true-blue array of macro variables, it *does* give the programmer a convenient means of getting the same functionality as one would have with "real" arrays. Just like traditional arrays, the macro variables created by MACARRAY aren't really helpful until used in an application.

The example below shows how the variables could be used in the "real world." The application is simple — print 50 observations from each dataset in a library. The hard-coded (and error-prone!) approach to this task would require the programmer to specify the name of each dataset. Using MACARRAY, however, all that is required is insertion of the PRINT procedure code in a macro %DO loop (of course, another requirement is that the programmer keep track of the multiple ampersands!).

First, let's set up some test datasets.

```
data ex_1;
  x = 1;
run;

data ex_2(label="Label for EX_2");
  x = 2;
run;
```

Now we define a macro to do the actual printing.

```
%macro try_it;
```

Invoke MACARRAY, pointing it to the WORK library.

```
%macarray(libname=work);
```

At this point all macro variables (DSCOUNT, DS1, DSL1, etc) have been defined. Run them through a DO loop if the count is greater than zero (that is, if we found some datasets). Notice that if this execution of MACARRAY found "n" datasets and an earlier invocation of the macro identified "n+1" datasets we would still use only the correct ("n") datasets because we control the loop by the value of DSCOUNT. So long as DS_n and DSL_n macro references are controlled by referencing DSCOUNT there is no danger of referring to a variable that doesn't exist or is inappropriate.

```
%if &dscount. > 0 %then
  %do i = 1 %to &dscount.;
    proc print data=work.&ds&i(obs=50);
      title
        "Dataset WORK.&ds&i, label &dsl&i";
    run;
  %end;
```

End the macro, then invoke it. The programmer doesn't need to know how many datasets are in the library. All that TRY_IT does is points to the library and prints anything that's available. The amount of code that's needed is the same to print two or 20 datasets.

```
%mend;
%try_it;
```

APPENDIX A: MACRO LISTING

```
%macro macarray(libname=);
```

* MACARRAY

Creates macro variables DS1-DS_n and DSCOUNT - names and count of all datasets pointed to by parameter LIBNAME. Also creates DSL1 to DSL_n - these contain the dataset labels.

Example: dsmacvar(libname=study1)

If datasets BASELINE, AE, and SUMMARY were in library STUDY1, DSMACVAR would assign a value of 3 to DSCOUNT and would assign AE, BASELINE, and SUMMARY to macro variables DS1, DS2, and DS3, respectively.

Note: The macro assumes that the LIBNAME has been allocated. If not, DSCOUNT returns a value of 0.

The macro writes the following lines to the SAS Log.

```
2 Datasets:
Dataset 1: EX_1      has label
Dataset 2: EX_2      has label Label for EX_2
```

The PRINT procedure would be invoked twice. The statements generated during the first pass of the loop are:

```
proc print data=work.ex_1(obs=50);
  title "Dataset WORK.ex_1, label ";
run;
```

The statements generated during the second and final pass of the loop are:

```
proc print data=work.ex_2(obs=50);
  title "Dataset WORK.ex_2, label Label for
EX_2";
run;
```

EXTENSION

If we keep in mind the parallel between the macro "array" and its DATA step cousin, an enhancement to the macro would be to add an "array" name. The invocation of the macro might look like:

```
%macarray(libname=work, arr_name=test);
```

Rather than creating DSCOUNT, DS_n, and DSL_n, MACARRAY would create TESTN, TEST1, TESTL1, TEST2, TESTL2, and so on. This extension would allow the programmer to use multiple "arrays" in an application.

REFERENCES

ATTRDIFF utilizes the SAS dictionary tables. These are explained in SAS Technical Report P-222, "Changes and Enhancements to Base SAS Software, Release 6.07," and Dilorio and Michal, "Data About Data: An Introduction to Dictionary Tables," SESUG '95 Proceedings (also in SUGI 21 Proceedings).

QUESTIONS? COMMENTS?

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

102 Westbury Drive	919.942.2028
Chapel Hill NC 27516-	7216987@mcimail.com
9154	

Author: Frank DiIorio, ASG, Inc., 1996

*** Copy freely so long as authorship is cited! ***
;

%global dscount;

%let libname = %upcase(&libname.);

proc sql noprint;

create table _names_ as

select memtype, libname, memname, memlabel

from dictionary.tables

where memtype = 'DATA' & libname = "&libname.";

select nobs into :dscount

from dictionary.tables

where memtype = 'DATA' & libname = "WORK" & memname = '_NAMES_';

quit;

%if &dscount. = 0 %then %goto %bottom;

%do i = 1 %to &dscount.;

%global ds&i dsl&i;

%end;

data _null_;

set _names_;

call symput("ds" || left(trim(put(_n_,3))), memname);

call symput("dsl" || left(trim(put(_n_,3))), memlabel);

run;

proc delete data=_names_;

run;

%put %left(&dscount.) Datasets: ;

%do i = 1 %to &dscount.;

%put Dataset &i.: &ds&i has label &dsl&i;

%end;

%bottom: ;

%mend;