Paper 6503-2016

Writing Reusable Macros

Philip R Holland, Holland Numerics Limited

ABSTRACT

I get annoyed when macros tread all over my SAS® environment, macro variables and data sets, so how do you write macros that play nicely and then clean up afterwards? This paper describes techniques for writing macros that do just that.

INTRODUCTION

A macro would be reusable if it has the following list of features:

- Subsequent calls of a macro cannot be impacted by a previous call.
- All SAS data sets and SAS objects created during a macro call will need to be documented and then deleted at the end, unless explicitly kept.
- Any SAS options changed within the macro are restored to their previous settings.

Therefore, a reusable macro should not:

- Create SAS data sets and SAS objects with names that are likely to have been created outside of the macro, unless the user is able to specify them.
- Create global macro variables, unless specifically written to do so.
- Update SAS options, unless specifically written to do so.

MANAGING SAS DATA SETS

The simplest naming convention to avoid names that have been used outside of the macro would be to prefix them with two underscores "*__dsname*". However, if you a writing a suite of reusable macros which call each other, this may not be adequate. To prevent conflicts a safer way would be to prefix all names with "*__macroname_dsname*", which will make them more likely to be unique, but will also make them easier to find and delete at the end of the macro call without impacting related macros:

```
%MACRO abc123(in=, invar=, out=, outvar=, keep=N);

  DATA __abc123_new;
    SET &in.;
    &outvar. = UPCASE(&invar.);
  RUN;

  PROC SORT DATA = __abc123_new OUT = &out.;
    BY &outvar.;
  RUN;

  %if %upcase("&keep.") eq "N" %then %do;
    PROC DATASETS LIB = work NOLIST NOWARN;
      DELETE __abc123_: / MEMTYPE = ALL;
    RUN;
    QUIT;
  %end;

%MEND abc123;

%abc123(in=sashelp.class,invar=name,out=class_sort,outvar=name_upper)
```

Note the KEEP= parameter in the macro definition, which provides a way to retain intermediate SAS data sets and SAS objects for debugging purposes. Also the IN= and OUT= parameters specify the input and output SAS data sets, which can then be controlled by the programmer.

While this example only considers SAS data sets, this should also apply to SAS objects, like formats, informats, templates and macros, which can also impact subsequent macro calls.

MANAGING SAS MACRO VARIABLES

In theory macro variables should be easier to control, as SAS uses the concept of "scope" for them. This means that a "global" macro variable is accessible anywhere in a SAS program, but a "local" macro variable can only be accessed within a macro and in a called macro. Macro parameters are automatically defined as local macro variables, but all new macro variables should be defined explicitly as global or local. The following code shows the three types of macro variables: global &macvar1, local &macvar1 and parameter &var, where only &macvar1 exists outside of macro %newmac, but all three macro variables exist inside:

```
%GLOBAL macvar1;
%LET macvar1 = 1;

%MACRO newmac(var=);

  %LOCAL macvar2;
  %LET macrvar2 = 2;

  %PUT macvar1=&macvar1. macvar2=&macvar2. var=&var.;

%MEND newmac;

%newmac(var=Y)

%PUT macvar1=&macvar1.;
```

This should be fine, except that it is perfectly possible to create a global macro variable inside of a macro. If this macro is then compiled, it may not be obvious that a global macro variable has been created, with unforeseen consequences.

So how could a macro be written to create a global macro variable safely? One way would be to specify the name of the new global macro variable in a macro parameter:

```
%MACRO newglobal(global1=, value1=, global2=, value2=);

  %GLOBAL &global1. &global2.;

  %if "&global1." ne "" %then %do;
    %LET &global1. = &value1.;
  %end;

  %if "&global2." ne "" %then %do;
    %LET &global2. = &value2.;
  %end;


%MEND newglobal;

%newglobal(global1=newglob, value1=Y)

%PUT global1=&global1.;
```

MANAGING SAS OPTIONS

The current settings of all SAS options are stored in a SAS dictionary table called dictionary.options (for PROC SQL), or sashelp.voptions (for PROC SQL and Data Steps). The SAS-supplied function GETOPTION() can then be used to extract the current setting. The new option settings can then be specified and used, and then the OPTION statement will allow the previous settings to be restore from the saved macro variables values:

```
OPTIONS PAGESIZE=60 LINESIZE=132 ORIENTATION=landscape;

%MACRO papermac(in=, ps=, ls=, orient=landscape);

  %LOCAL ps_in ls_in orient_in;
  %LET ps_in = %sysfunc(GETOPTION(pagesize,keyword));
  %LET ls_in = %sysfunc(GETOPTION(linesize,keyword));
  %LET orient_in = %sysfunc(GETOPTION(orientation,keyword));

  OPTIONS PAGESIZE=&ps. LINESIZE=&ls. ORIENTATION=&orient.;

  PROC PRINT DATA = &in.;
  RUN;

  OPTIONS &ps_in. &ls_in. &orient_in.;

%MEND papermac;

%papermac(in=sashelp.cars, ps=80, ls=72, orient=portrait)
```

CONCLUSION

There are no complicated programming techniques in this paper, just common sense. Before writing a new SAS macro think about what it will create and use, including SAS data sets, macro variables and options, and make sure that only the ones that the user can specify can “escape”. Writing macros that “play nice” will keep you popular and your macro users productive.

CONTACT INFORMATION

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

Name:	Philip R Holland
Organization:	Holland Numerics Limited
Address:	94 Green Drift
City, State ZIP:	Royston, Hertfordshire, SG8 5BT, United Kingdom
Work Phone:	+44-7714-279085
Fax:	+44-1763-242486
Email:	phil@hollandnumerics.com
Web:	www.hollandnumerics.com
sasCommunity.org:	www.sascommunity.org/wiki/User:Prholland