



Philip R Holland, Holland Numerics Ltd

Writing Reusable Macros

Agenda

- Introduction
- Managing SAS Data Sets
- Managing SAS Macro Variables
- Managing SAS Options
- Conclusions

Introduction

A reusable macro will have the following features:

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

Introduction

A reusable macro should not:

- 4) Create SAS data sets and SAS objects with names that are likely to be created outside the macro, unless the user is able to specify them.
- 5) Create global macro variables, unless specifically written to do so.
- 6) 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 are 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.

Managing SAS Data Sets

- The example below uses SAS data sets, but SAS objects, like formats, informats, templates and macros, should be controlled in the same way.

Managing SAS Data Sets

```
%MACRO abc123(in=, invar=, out=, outvar=, keep=N);
  DATA __abc123_new; **unique name (2)**;
    SET &in.; **user can set out=**;
    &outvar. = UPCASE(&invar.); **user can set outvar=, invar=**;
  RUN;

  PROC SORT DATA = __abc123_new OUT = &out.; **user can set out=**;
    BY &outvar.; **user can set outvar=**;
  RUN;

  %if %upcase("&keep.") eq "N" %then %do; **user can set keep=**;
    PROC DATASETS LIB = work NOLIST NOWARN;
      DELETE __abc123_ / MEMTYPE = ALL; **unique name (2)**;
    RUN;
    QUIT;
  %end;

%MEND abc123;

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

Managing SAS Macro Variables

- SAS uses the concept of “scope” for macro variables, where a “global” variable can be accessed anywhere in a program, but a “local” variable only exists in the macro where it is created and in any called macro.
- Macro parameters are automatically created as “local” variables.
- However, it is good programming practice to define new macro variables explicitly as “global” or “local”.

Managing SAS Macro Variables

```
%GLOBAL macvar1;  
%LET macvar1 = 1; **macvar1: global variable - accessible anywhere**;  
%MACRO newmac(var=); **var: local variable - accessible inside macro**;  
    %LOCAL macvar2; **macvar2: local variable - accessible inside macro**;  
    %LET macvar2 = 2;  
    %PUT macvar1=&macvar1. macvar2=&macvar2. var=&var.;  
%MEND newmac;  
%newmac(var=Y)  
%PUT macvar1=&macvar1.; **macvar1: global variable - accessible anywhere**;
```



Managing SAS Macro Variables

- The danger comes from the creation of “global” variables in macros, which can be hidden when the macro is called, particularly if the macro is compiled.
- How could a macro be written to create a “global” variable safely?
- One way would be to specify the name of the “global” variable to be created in a macro parameter.

Managing SAS Macro Variables

```
%MACRO newglobal(global1=, value1=, global2=, value2=);  
  %GLOBAL &global1. &global2.; **global variables - accessible anywhere**;  
  %if "&global1." ne "" %then %do; **only set value if name is non-missing**;  
    %LET &global1. = &value1.;  
  %end;  
  %if "&global2." ne "" %then %do; **only set value if name is non-missing**;  
    %LET &global2. = &value2.;  
  %end;  
%MEND newglobal;  
  
%newglobal(global1=newglob, value1=Y)  
%PUT global1=&global1.;
```



Managing SAS Options

- The current settings of all SAS options can be accessed using a SAS dictionary table called **dictionary.options** (using PROC SQL) or **sashelp.voption** (using PROC SQL or Data Steps).
- The SAS function GETOPTION() looks in the SAS dictionary to extract the current setting. For example:
GETOPTION(orientation,keyword) could return
“ORIENTATION=PORTRAIT”.

Managing SAS Options

The following example uses the steps below to preserve the SAS options settings:

- 1) The GETOPTION() function calls at the top of the macro are used to save the current settings.
- 2) New SAS options settings are assigned in the macro.
- 3) The saved settings are reapplied at the end of the macro using the OPTIONS statement to restore them to their original values.

Managing SAS Options

```
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)); **save pagesize**;
  %LET ls_in = %sysfunc(GETOPTION(linesize,keyword)); **save linesize**;
  %LET orient_in = %sysfunc(GETOPTION(orientation,keyword)); **save orientation**;

  OPTIONS PAGESIZE=&ps. LINESIZE=&ls. ORIENTATION=&orient.; **change settings**;

  PROC PRINT DATA = &in.;
  RUN;

  OPTIONS &ps_in. &ls_in. &orient_in.; **restore settings**;

%MEND papermac;

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



Conclusions

- **Common sense:** there are no complicated programming techniques in this paper, just common sense.
- **Think:** 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”.
- **Remember:** writing macros that “play nice” will keep you popular and your macro users productive.



Contact Details

Philip R Holland, SAS Consultant

Holland Numerics Ltd

94 Green Drift, Royston, Herts SG8 5BT, UK

tel: +44-7714-279085

fax: +44-1763-242486

email: phil@hollandnumerics.com

web: blog.hollandnumerics.org.uk