About the presenter

Philip R Holland has been working with SAS software for over 30 years, providing SAS technical consultancy and training through his own company, Holland Numerics Ltd, in the financial, retail and pharmaceutical sectors in the UK, Belgium, Holland, Germany and USA since 1992.

He is an enthusiastic software developer, not only using SAS, but also Perl, Java, JavaScript and Visual Basic. His fourth book, “SAS Programming and Data Visualization Techniques”, was published in 2015.

SAS® GLOBAL FORUM 2016

IMAGINE. CREATE. INNOVATE.

Philip R Holland, Holland Numerics Ltd

Writing Reusable Macros

HOLLAND NUMERICS LIMITED

#SASGF

Introduction

A reusable macro <u>will have</u> 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

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.

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)
```

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
email:	phil@hollandnumerics.org.uk
web:	blog.hollandnumerics.org.uk