PhUSE 2008

Paper CS06

Alias Macros, a Flexible Versioning System for Standard SAS Macros

Jean-Michel Bodart, UCB Pharma, Braine-l'Alleud, Belgium
Guido Wendland, UCB Pharma, Monheim, Germany

ABSTRACT

The Alias Macros concept adopted by UCB Global Statistical Programming is a macro versioning system that offers transparency and flexibility of use, together with easy, centralized library administration and maintenance.

Different versions of the same macro can be used concurrently and/or successively within the same project (clinical study). Alias Macros specify a default version for each macro at the project level. Programs generally use the default version, but a different one can be called as needed in specific programs.

When new versions become available, an upgrade of chosen macros can be decided at any time by the project programmers, implemented in minutes, tested on selected programs within the project, and generalized or rolled back very easily, as far as the macro-parameters syntax is backward-compatible.

This system does not require other modules than Base SAS®.

INTRODUCTION

Libraries of Standard SAS macros are generally considered in the Pharmaceutical Industry a desirable way of re-using carefully developed, valuable SAS code within the analysis of clinical trials, in an attempt to increase the quality and effectiveness of the programming.

It's common practice to build macros in order to provide easily adaptable, dynamic code that will perform similar computations and/or display of results on various data sets, with just the specification of a few parameters, thereby saving tedious, careful adaptations of non-dynamic programs.

Companies acquisitions and merging, personnel turnover, re-organization of the work in a more flexible and globalized way contribute to the need to share those macros with a growing number of colleagues and collaborators.

However, without defining and maintaining a central library of macros, assigning them unique version numbers, and controlling their changes, one would rapidly reach a confused situation where macros of different sources are copied across different trial or project folders, adapted in various ways by different people, copied again, ending up with multiple macros of the same name but slightly different parameters and capabilities. So that it sometimes becomes difficult to figure out why suddenly a macro is not able anymore to perform some tasks it was doing very well previously.

Dealing with multiple versions of macros can be a difficult task for the developers, library administrators and end users of those macros. Indeed SAS macros tend to evolve over time, provide improved functionality, have bugs discovered and corrected, focus on new needs and forget about past uses.

Moreover, some macros which call each other in order to perform specific tasks may have to evolve jointly to offer these improvements. Backward-compatibility may be sacrificed. Additionally, despite careful initial testing in a variety of situations, unexpected limitations may still be uncovered afterwards, and compromise the usefulness and acceptability of new sets of macros among users.

For all these reasons, a corporate library of standard macros is needed. Ideally, it should be simple, easy to use, but also flexible, transparent, easily maintained, and allow the users to control which version(s) they are using and when to upgrade. We believe alias macros can meet these requirements.

THE PAIN OF UPDATING MACROS

The introduction of updated macro versions during the course of a project, or even between two successive projects, is not always smooth. The distribution process of the new macros may generate many problems. UCB used to have the latest versions of standard macros copied by administrators to a read-only folder within each trial or project, at the time of the trial or project setup. When new versions became available, they would be refreshed for a specific trial or project, on request of its lead programmer. But in case of unexpected backward-compatibility problem, the impact could be serious. Reverting to previous versions was not readily possible. As the experience of problems accumulated and since the users did not master the changes brought up by the new versions, they became reluctant to apply any update.

In some cases, the situation was even more paradoxical: the new version of a macro allowed some reports to be created much more easily than before; however a whole set of existing programs would require significant rework in order to be able to produce again some previous reports with the new macro version. The implementation did not allow using one version for some reports and another for the remaining ones.
Therefore, when the merger of UCB Pharma with Schwarz Bio Sciences happened, and the need to establish common working practices arose, it was desired to implement a flexible, user-controllable, transparent versioning system of standard macros. This is how we came up with the new concept of Alias macros.

STANDARD MACROS LIBRARY, VERSIONING AND ALIAS MACROS

AUTOCALL MACROS

The easiest way to make a number of centrally maintained macros available to the SAS programming environment is probably by using the SAS Autocall facility.
In this approach, the SAS global option SASAUTOS is updated to include the library folder(s) in the search path for macros. The MAUTOSOURCE global option is required to turn on the Autocall facility. Other strategies for building and using SAS macro libraries are described by Carpenter (2002).
Note: The MRECALL global option is required if a temporary re-definition of an Alias macro is to be used (see below).
A requirement of autocall macros is that the macro name specified in the macro definition (%MACRO statement) must be the same as the filename containing the macro definition (without the ".sas" file extension). In addition it must be a valid sas name (case-insensitive, made of one or more characters from "A" through "Z" or "a" through "z", digits "0" to "9" and/or underscores "_", and not starting with a digit).
It may be useful to have a convention in order to easily identify standard macros that are part of the library, from user-defined, trial- or project- specific custom macros. Historically the UCB convention was to start standard macro names with an underscore ("_").

VERSIONING

Usually version numbers use the x.y notation, where the integer part (x) corresponds to the major version number and the decimal part (y) to the minor version numbers. The first version is generally called version 1.0.
Enhancing the robustness, efficiency, user-friendliness of a macro, or removing some limitations, while keeping the same major functionality unchanged and a large part of the existing code unmodified are considered minor revisions, denoted by an incremented minor version number.
Applying more important changes to the existing code, together with changes in the main functionality, corresponds to major revisions, denoted by a version number incremented to the next integer.

The most transparent and straightforward way to implement macro versioning is probably to include the version number in the macro name itself. But as the decimal point is not valid inside a sas macro name, it must be replaced. One possible option is to replace it by an underscore character (xx_yy). Another one is to assume a format such as xxyy where a fixed number of digits (yy) represents the decimal part or minor version number while a (possibly variable) number of digits (xx) represent the integer part or major version number. In order to make it clearer that this represents a version number, we took the convention to append it to the macro name, following the string "_v" or "v".
So, if we create the first version of a macro that retrieves the list of all variables found in one dataset, we could name it "_varlist_v1_0" or, if avoiding underscores is preferred, "varlistv0100".

This macro library and this versioning system allow users to call any version of any standard macro from the library from any trial or project folder. Users may call the usual (default) version of a macro in some programs, try in other programs a more recent version when it becomes available, or go back to a previous version when needed.

The macro library administration is easily performed: a few administrators have write access to the library folders, where they simply copy the new macros that have been validated and are ready to use for production. They check that the new macro has a new version number so that it does not overwrite an existing file, and that the numbering is sequential. Although all versions could be kept in a single common folder, it was decided to keep only the most recent version of a macro in a Standard\Macro folder of the Library, for easy identification. Earlier versions are moved to the Standards\Retired folder. In order to keep access to the old versions at the project/trial level, the path to autocall macros is defined as follows:

```
OPTION SASAUTOS=("<path to SAS Institute macros>"
                 "<path to Library>\Standards\Macro"
                 "<path to Library>\Standards\Retired"
                 "<path to project- or trial-specific macros>");
```

The first locations in the search path take priority over the following locations. This implies that Standard Macros and their retired versions have precedence over trial/project macros, so that users are not able to override standard

macro definitions in the project/ trial macro folders. If a modified version of a standard macro is needed in a trial/project, then it should be renamed (e.g. replace version number by "mod" to convey the information that a modified version is used).

ALIAS MACROS

Of course, it is also desirable to define a default version for each macro, to be called each time the macro is invoked without specifying a version number.
This functionality is implemented by so called alias macros. An alias macro is just a way to redirect a macro call without version number to a call of its default version (i.e. usually the latest one).

BASIC FORM OF ALIAS MACRO

In its basic form, an alias macro is a macro definition with no arguments and no version number, that the macro processor resolves into a call to the default version of the same macro, which will consider as its own any arguments following the alias macro invocation.

For example, imagine we have two versions (1.0 and 1.1) of the autocall macro _varlist (used to retrieve the list of variables found in some dataset(s)):

_varlist_v1_0.sas (in library folder "Retired") contains the following macro definition:

```
%macro _varlist_v1_0( <argument1>, <argument2> ...);
   [more SAS statements...]
%mend;
```

_varlist_v1_1.sas (in library folder "Current") contains the following macro definition:

```
%macro _varlist_v1_1( <argument1>, <argument2> ...);
   [more updated SAS statements...]
%mend;
```

_varlist.sas (Alias macro, in library folder "Alias", ready to be copied to the Project's folder "Alias") contains the following macro definition (defining the latest version [1.1] as the default):

```
%macro _varlist;
   /* This alias macro defines a default version
      for that macro */
   %* Note the absence of arguments and semi-colon
      in actual macro invocation below;
   %_varlist_v1_1
%mend;
```

The default version of the macro is always callable via the alias macro, e.g.:

```
%put %_varlist ( <argument1>, <argument2> ...);
```

As the macro %_varlist was defined with no arguments, the SAS macro processor will resolve (i.e. replace) the macro-expression '`%_varlist `' to the contents of the macro-definition, so the invocation above becomes:

```
%put /* This alias macro defines a default version
       for that macro */
     %* Note the absence of arguments and semi-colon
       in actual macro invocation below;
     %_varlist_v1_1 ( <argument1>, <argument2> ...);
```

And after removing the comments (this applies to both styles "`/* ... */`" and "`%* .... ;` " but would not work properly in this case for the style "`* ... ;` "):

```
%put %_varlist_v1_1 ( <argument1>, <argument2> ...);
```

Of course it's always possible to explicitly specify a given version in the invocation, e.g.:

```
%put %_varlist_v1_0 ( <argument1>, <argument2> ...);
```

or

```
%put %_varlist_v1_1 ( <argument1>, <argument2> ...);
```

A reason for explicitly specifying the current version in some program would be to stick to that version, even if a new version is expected to become available and become the new default some time later.

ADVANCED FORMS OF ALIAS MACROS

When it's desired to use an old version of a macro that calls other sub-macros, and a specific version of the sub-macro is also required, then a more advanced form of alias macro is needed. The advanced alias macro temporarily replaces the default version of the sub-macros by re-defining the aliases of these sub-macros. When the main macro terminates, the redefined aliases of the sub-macros have to be suppressed.

For example, %_rep2000 is a macro used very frequently at UCB. The current version (3.2) generates reports in Portable Document Format (PDF) from a summarized dataset. A previous version (2.1) was generating ASCII text files, which could be transformed later into a Microsoft-Word document (DOC). Although PDF format is now systematically used for study reports and submissions, the DOC format is still sometimes preferred, e.g. to create a PowerPoint presentation with contents linked to one or more reports in DOC format. PDF documents could not be used for that purpose. However, the %_rep2000 macro needs 3 other macros (%_splits, %_justify, %_parm_db) which have evolved over time without remaining backward-compatible: %_rep2000 version 3.2 uses the most current versions of these macros (respectively, 1.1, 1.1 and 2.1) while %_rep2000 version 2.1 requires older versions (respectively, 1.0, 1.0 and 1.5).

The Alias macro for %_rep2000, %_splits, %_justify and %_parm_db looks like:

```
%macro _rep2000;
   /* This alias macro defines a default version
      for that macro */
   %* Note the absence of arguments and semi-colon
      in actual macro invocation below;
   %_rep2000_v3_2
%mend;
%macro %_splits;
   /* This alias macro defines a default version
      for that macro */
   %* Note the absence of arguments and semi-colon
      in actual macro invocation below;
   %_splits_v1_1
%mend;
%macro _justify;
   /* This alias macro defines a default version
      for that macro */
   %* Note the absence of arguments and semi-colon
      in actual macro invocation below;
   %_justify_v1_1
%mend;
%macro _parm_db;
   /* This alias macro defines a default version
      for that macro */
   %* Note the absence of arguments and semi-colon
      in actual macro invocation below;
   %_parm_db_v2_1
%mend;
```

Because it requires non-default versions for its sub-macros, %_rep2000 version 2.1 cannot be called directly but requires a specific alias macro (identified as alias macro for version 2.1), which looks like:

```
%macro _rep2000_a2_1/parmbuff;
  /*** Option parmbuff will collect all parameters passed within parentheses
       and store them in macro-variable &syspbuff ***/
  %* requires temporarily redefining alias to corresponding versions of submacros
*;
   %macro _splits; %_splits_v1_0 %mend;
   %macro _justify; %_justify_v1_0 %mend;
   %macro _parm_db; %_parm_db_v1_5 %mend;
```

```
    /*** Call %_rep2000_v2_1 with the specified arguments.
         Note the &syspbuff macro-variables which resolves to the parameters
         of original macro-call, within parentheses.
         Macro unquoting is required  ***/
     %unquote(%nrstr(%_rep2000_v2_1)&syspbuff);

     %* revert to original version of alias macros
        by suppressing the modified definitions
        (requires option MRECALL to be in effect, so that,
         when needed, the original versions can be re-compiled) *;
     proc catalog catalog=work.sasmacr force;
        delete _splits _justify _parm_db / entrytype=macro ;
        run;
     quit;
     option mrecall;
  %mend;
```

Note the naming convention (the suffix a2_1) applied to identify this as an alias to a specific, non-default version of the %_rep2000 macro (v2_1).

Also note the final call to the CATALOG procedure, to delete the three compiled submacros. Since the PROC CATALOG can only be executed after completing the %_rep2000 call, the arguments of the %_rep2000 call cannot be appended to the resolved alias macro definition, but must be explicitly captured and passed back to %_rep2000_v2_1. This involves:

- the use of the /PARMBUFF option in the %macro statement, which captures any number of macro arguments into the automatic macro-variable &SYSPBUFF, together with the enclosing parentheses.
- an actual call to %_rep2000_v2_1 with the parameters contained in &SYSPBUFF
- macro-quoting the expression %_rep2000_v2_1 (preventing it to be called prematurely without its arguments). This is done by the macro-quoting function %NRSTR().
- unquoting the combined expression (macro call and arguments) so that they are resolved as a single macro statement. This is done by the macro-function %UNQUOTE().

Finally, note the option MRECALL, needed to ensure the normal version of the corresponding three alias macros will be recompiled the next time they are needed.

IMPLEMENTATION OF NEW VERSIONS OF MACROS

Normally, the implementation of new macros requires updating the alias macros to refer to the new version.

Alias macros referring to the latest versions of each standard macro are created by the library administrators in a specific library folder, and updated as new versions become available.

When a project/ trial analysis starts, the (Lead) Project programmer(s) must copy the current alias macros to the project/ trial alias folder, which is now also defined as part of the SASAUTOS option:

```
   OPTION SASAUTOS=("<path to SAS Institute macros>"
                    "<path to Library>\Standards\Macro"
                    "<path to Library>\Standards\Retired"
                    "<path to project or trial Alias macros>");
                    "<path to project- or trial-specific macros>");
```

This is the responsibility of the (lead) project/trial programmer(s) to update the project or trial alias macros at the time that is most convenient with regard to the project/trial, in order to start using the new versions.

It's also possible to roll back to a previous version, if needed, by editing the corresponding alias macro.

IDENTIFYING MACRO VERSION(S) USED

The most recent version of a standard macro can always be readily retrieved from the Library\Standards\Macro folder (remember older versions are always moved to the Library\Standards\Retired folder).

The default macro version in a specific project/trial can be retrieved from the project/trial Alias macro folder. It's good practice to keep a history of alias macro changes as comments within the alias macro itself.

Programs not using the default version of a macro do explicitly specify the version number in the macro call.

When SAS options MPRINT and/or MLOGIC are active, the full macro name including version number is automatically printed in the SAS log, at zero maintenance cost, whether the default version is used or not.

E.g.:

```
MPRINT(_REP2000_V3_1):   STOP;
MPRINT(_REP2000_V3_1):   END;
MPRINT(_REP2000_V3_1):   END;
MPRINT(_REP2000_V3_1):   RUN;

NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.00 seconds


MPRINT(_PARM_DB_V2_0):   PROC SQL NOPRINT;
MPRINT(_PARM_DB_V2_0):   SELECT name INTO : _ckLisType FROM sashelp.vmacro WHERE name='_LISTYPE';
MPRINT(_PARM_DB_V2_0):   SELECT name INTO : _ckOdsType FROM sashelp.vmacro WHERE name='_ODSTPKPT';
MPRINT(_PARM_DB_V2_0):   SELECT name INTO : _cktitle FROM sashelp.vmacro WHERE name='_REP2KTIT';
```

Note: Project- or trial-specific macros generally don't have a version number.

EMERGENCY FIX OF BUGS IN THE DEFAULT VERSION OF A MACRO

In emergency situations it might be required to temporarily use a modified version of a library macro until a new, corrected version is developed, validated for general use, and added to the library. The library version would be copied inside a specific project, modified and validated for use in that specific project, which is typically much less work than an update and revalidation for general use. A specific convention is proposed for this case, i.e. to name the modified version according to the original version with an additional suffix "mod".

E.g., if this is the case for version 1.1 of macro %_varlist, the modified version would look as follows:

_varlist_v1_1mod.sas (in the Project's macro folder) contains the following macro definition:

```
%macro _varlist_v1_1mod( <argument1>, <argument2> ...);
   /* Note: this modified version of macro _varlist_v1_1
            is created and validated for temporary, restricted
            use within the Project XXXXXX, as a workaround to
            the limitation/bug of version 1.1 described in
            Incident Report YYYYY */
   [more updated SAS statements...]
%mend;
```

If this modified version is used as the new default version in that project, then the corresponding Alias macro would be modified as follows:

_varlist.sas (Alias macro, in the Project's folder "Alias") contains the following macro definition (defining version 1.1 modified as the new default):

```
%macro _varlist;
   /* This alias macro defines a default version
      for that macro */
   %* Note the absence of arguments and semi-colon
      in actual macro invocation below;
   %*_varlist_v1_1

   /* DD-MM-YYY, J-M Bodart:
      Temporarily use modified version as a workaround to
      the limitation/bug of version 1.1 described in
      Incident Report YYYYY */
   %_varlist_v1_1mod
%mend;
```

CONCLUSION

The alias macro concept provides a couple of advantages over traditional macro library approaches, such as the on-demand copy of the latest version of all the standard macros by an administrator to a project- or trial-specific folder with read-only access for regular users:

- Flexibility: It's possible to use multiple versions of a macro inside the same project or trial. Older versions remain available. New versions can be tested in a few programs, without impacting others, before generalizing their use.
- User control: At any time, users can change the default version of one or more macros by just updating the corresponding Alias macros.
- Simplicity of use: alias macros are created and updated in a specific library folder, ready to be copied by the users to the project or trial folder.
- Transparency: The log displays which macro version is used. Old version are easily retrieved (all are stored in the same library folder).
- Easy maintenance: Version control is easily implemented; rigorous checks of macro updates are readily performed. Library administrators assign version numbers sequentially. Only the library folders require access restrictions (read access to regular users, write access to administrators).

Up to now, we have confirmed the feasibility to apply this concept on a variety of macros in a test environment. We should be able to determine in the next future whether these benefits are actually confirmed when alias macros are fully implemented in the production analysis of a real clinical trial or project.

REFERENCES

Carpenter, A. L. (2002). Building and Using Macro Libraries. Paper 12-27 presented at the SAS Users Group International 27, Orlando, FL.

SAS OnlineDoc (http://support.sas.com/onlinedoc/913)

CONTACT INFORMATION

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

Jean-Michel BODART
UCB Pharma SA
Chemin du Foriest
Braine-l'Alleud B-1420
Belgium
Work Phone: +32-2-386 3250
Fax: +32-2-386 3558
Email: jean-michel.bodart@ucb-group.com
Web: www.ucb-group.com