PhUSE 2007

Paper PO01

Anticipating User Issues with Macros

Lawrence Heaton-Wright, Quintiles, Bracknell, Berkshire, UK

ABSTRACT

How can you stop users asking you questions like:

"What macros are available?"
"Why doesn't this macro work?"
"How do I stop this macro doing that?"

There are lots of techniques available to SAS macro developers such as providing a help option, checking that the input values of a macro parameter are correct, supplying useful warning and error messages and creating, maintaining and documenting a code library.

The aim of this poster is to provide some ideas to help macro developers to help their users help themselves.

PROVIDE INFORMATION

The first step in helping your users discover what macros are available and how to use them is to provide them with this information. If the macro invocation does go wrong then you need to provide your users with meaningful LOG messages. If they're still puzzled why the macro hasn't executed as they expect then perhaps they are not invoking the correct version or even invoking the macro itself.

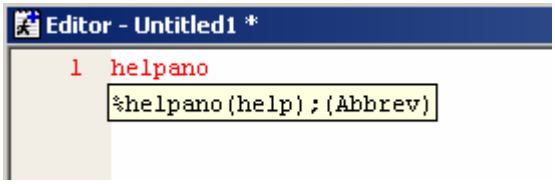
"WHAT MACROS ARE AVAILABLE?"

There are several methods that can be used to provide users with a list or macros that are available for use. However, they all depend on the macro programmer providing the initial documentation.

- Use a SAS macro to list the macros available. SAS already provides a macro like this (HELPANO which documents the annotate macros available)
- Create a database (intranet, Excel, Access, etc.) which contains similar information to a help macro
- Use Keyboard Macro Files (PharmaSUG 2003 CC25) to create shortcuts for macro calls
- Create a laminated printout of a database of available macros (old-fashioned but still worthwhile)

HELP USERS TO HELP THEMSELVES

We can utilise the HELP technique that SAS so neatly includes with the annotate macros. Using a Keyboard Macro File abbreviation we can invoke the HELPANO macro which show a list of the annotate macros available for use.



```
                    ANNOTATE HELP

HELP is currently available for the following macros

                        BAR             CIRCLE
       CNTL2TXT         COMMENT         DCLANNO
       DRAW             DRAW2TXT        FRAME
       LABEL            LINE            MOVE
       PIEXY            POLY
       POLYCONT         POP             PUSH
       RECT             SCALE           SCALET
       SEQUENCE         SLICE           SWAP
       SYSTEM           TXT2CNTL



 Enter %HELPANO(macroname) for details on each macro,
    or %HELPANO(ALL) for details on all macros.
```

All the macro developer has to do is then copy and modify the HELPANO macro so that instead of the SAS-supplied annotate macros, help is provided on the list of company-specific macros available to users.

```
%HelpMac (Help);
```

In the LOG file appears a help series of help messages that you, the macro developer, has defined. This can include all the macro variables and default values (if any) or other useful information about the macro.

DOES THE MACRO EXIST?

This has already been discussed (what macros are available). However, this could be due to the user not having the correct paths in the SASAUTOS or SASMSTORE options. The SASAUTOS option defines the areas (and the order they're searched through by SAS) where macros are stored. The SASMSTORE option specifies the data library where the stored compiled macros catalog is located.

In SAS V9 the MAUTOLOCDISPLAY option was added. This extremely useful option displays the autocall macro source location in the log when the macro is invoked.

```
OPTIONS MAUTOLOCDISPLAY;
%LET var=%LOWCASE(FSTAFSH);
```

In the LOG file we get:

```
MAUTOLOCDISPLAY(LOWCASE):  This macro was compiled from the autocall file C:\Program
                           Files\SAS\SAS 9.1\core\sasmacro\lowcase.sas
```

"WHY DOESN'T THIS MACRO WORK?"

This is probably the most common question.

And you're not allowed to reply: "Because you've done something stupid!"

For some reason this seems to annoy people so you need to provide information in the LOG file to help your users determine why the macro hasn't executed as they expected. This will then free you, the developer, to put your feet up and admire the view out of your office window or, more likely, continue to develop more macros and programs to help your users!

WHAT DOES THE LOG FILE SHOW?

As with any de-bugging/investigations with SAS we start with the LOG file. Presumably, our users will have checked the LOG file for ERRORs, WARNINGs and NOTEs. If they haven't, get them to check the LOG file first! However, assuming that they have checked the LOG and aren't quite sure what message SAS is giving them, then this is the first port of call for the macro developer.

When developing the macro you can help your users by supplying them with precise, meaningful LOG messages. We've all had the experience of trying to decipher SAS LOG file messages. By checking for incorrect input parameters, not enough or too many input parameters or incorrect values for input parameters we can inform the users exactly what they have done erroneously.

ERROR-TRAPPING TO HELP YOUR USERS

As a macro developer we need to be aware that users can, sometimes by accident, have incorrect values for the input macro parameters. When developing a macro there are several techniques that can be deployed to determine whether the input parameters have valid values or not.

WHAT INPUT PARAMETERS HAVE VALUES?

As a developer you can check what input macro parameters have values defined in the macro invocation using the PARMBUFF option (see PhUSE 2005 TS08) which creates the SYSPBUFF automatic macro variable. This macro variable can be scanned for macro variables containing values that the user has entered.

CHECK INPUT PARAMETERS FOR INCORRECT VALUES

If these values are meant to be a restricted list of values, this can be checked for and the macro terminated with a suitable error message.

```
%IF NOT (&ynVar. EQ Y OR & ynVar. EQ N) %THEN %DO;
    %PUT ERROR: YNVAR NOT DEFINED AS Y OR N [&ynVar.];
    %PUT ERROR: Terminating &SysMacroName.;
    %PUT;
    %GOTO EndMacro;
%END;
```

If the macro variable we are checking [YNVAR] does not contain Y or N then the error message is displayed and the GOTO statement forces the macro processor to branch to the label specified here (in this case ENDMACRO). Further down the macro we have:

```
%EndMacro:
    %PUT NOTE: ENDING &SysMacroName. MACRO;
    %PUT;
```

An alternative to the %GOTO functionality is to use a series of %IF %END statements testing a clause to terminate or continue the macro. In V9 the %RETURN statement has been added which causes normal termination of the currently executing macro.

LETTING THE USERS KNOW WHAT'S INCORRECT

Once input parameters have been checked for valid values, we need to let users know this. The easiest method for achieving this is to send this information to the LOG file via %PUT statements. Using these statements enable developers to precisely inform the users exactly what they have entered incorrectly, as shown above in the YNVAR checking example.

HAVE YOU GOT THE CORRECT MACRO PARAMETERS?

This should be easy to detect following a check of the LOG file or the macro invocation. The users can check their "What macros are available" help guide or the SAS log file could hold some clues.

If keyword parameters are used then an incorrect parameter will result in an error message such as:

```
ERROR: The keyword parameter XXXXXX was not defined with the macro.
```

Positional parameters can be trickier. Only if more positional parameters are defined will SAS pick this up as an error message.

```
ERROR: More positional parameters found than defined.
```

MACRO DESIGN TIPS

There are several things to consider when designing macros.

- Have you already got a macro to do this job? Can this be modified?
- Can you re-cycle code from previous projects?
- Can you use utility macros to check variables or automate sections?
- Have you got standard variable naming and usage conventions?
- Remember to design and document your macros first before embarking on coding
- De-clutter the LOG file once the macro is coded and tested

Once you've answered these questions and started the design of your macro you can start thinking about the actual code in detail.

CAN YOU MODIFY AN EXISTING MACRO?

You have received a request from a user to create a macro to execute a certain task. One of the first things you should think about is: Does a macro already exist to do this task? If so, let the user know and get on with your next task, which may or may not involve food, coffee, tea or even in extreme circumstances, actual work.
If not, your next thought should be: Can I modify an existing macro?

MODIFIYING EXISTING MACROS

This is not as simple as you may think. You've got to make sure that any modifications you make to an existing macro do not alter previously validated macros. If you've got existing test data, then you'll re-run your modified macro on this test data to ensure that the macro still works as it did. You then need to test your modification.

USING UTILITY MACROS

We've already checked to see if there's a macro available to do the task requested. However, there may be other tasks which you need to accomplish in the new macro which require the use of utility macros. These are macros which perform small, but vital, tasks which can be used in many different macros. One use of this is to check the input values of a macro variable, or finding the maximum title number which is currently defined. You don't want to have to keep re-writing this code as that just gets tedious and is a waste of your valuable time (see above for food and drink breaks, snoozing, etc.).

STANDARD VARIABLE NAMING AND USAGE CONVENTIONS

There are many conventions in the pharmaceutical industry: CDISC, your own company standards, customer companies, etc. But these generally all apply to SAS data sets and variables. There aren't many when it comes to macro variables. The investment required for a set of standardised macro variables is not too bad when you consider what type of variables you will commonly be using.

Some suggestions are:

- Yes/No flag variables - start with YN (i.e. ynDeBug)
- Datasets - variable contains DSET (i.e. inDSet)
- Input-type variable - start with IN (i.e. inDSet)
- Output-type variables - start with OUT (i.e. outDSet)

These can be scanned for in conjunction with the PARMBUFF option and utility macros invoked to check that these variables contain valid values.

MACRO VARIABLES IN UPPER OR LOWER CASE?

Even though the user can enter macro values in upper-, lower- or mixed-case, as a developer you want to be working in either upper- or lower-case. Macro parameters can be converted using the UPCASE macro function or the SAS-supplied autocall macro LOWCASE (which acts as a SAS macro function).

```
%LET ynvar = %UPCASE(ynvar);
```

This means that you can compare a list values for YNVAR in upper case, rather than trying to cater for both cases, which just makes the coding more complicated.

DE-CLUTTERING THE SAS LOG FILE

There are several macro de-bugging options that are available for the macro developer when testing and developing their macros.

MPRINT, MLOGIC & SYMOLGEN are the most common. Unfortunately these can actually provide too much information for users when it comes to de-bugging the LOG file. It's a good idea to turn these options off for the duration of the macro.

For V9 SAS has provided the additional options MPRINTNEST and MLOGICNEST. These options enable macro nesting information to be displayed in conjunction with the MPRINT and MLOGIC options respectively. Instead of just showing the macro name within the MPRINT (*MACRO*), the MPRINTNEST option shows whether a macro has been called within a macro, e.g. MPRINT (*MACRO.NEWMACRO*).

SOURCE can be turned off (OPTIONS NOSOURCE), thereby reducing the amount of SAS code in the LOG file. However, this should only be used for macros which have sufficient notes and warnings provided by the developer. However, some customers request that all SAS source code be available in the LOG file, and in this case, SOURCE should be left active.

It's always a good idea to have a debug on/off switch as part of your macro call, e.g. YNDEBUG = Y or N, Y to switch on all the macro de-bugging options to enable you, as a developer, to de-bug and modify your macro.

The best method for ensuring that your macro does not disrupt the system set-up once the macro has finished executing is to interrogate the SASHELP.VOPTION data view to select the settings for the macro de-bugging options and send to macro variables at the start of the macro execution. Once the macro has finished executing then reset the options back to their original values which are stored in the macro variables defined earlier.

In SAS V9 have upgraded this data view by including the GROUP variable. This variable allows the user to select options from only certain grouping. In this case GROUP = 'MACRO' will select macro-specific options.

```
PROC SQL;
    SELECT optname
        , setting
        FROM SASHELP.voption
        WHERE group EQ 'MACRO'
    ;
QUIT;
```

REFERENCES

Creating Display Manager Abbreviations and Keyboard Macros for the Enhanced Editor, Arthur L. Carpenter, PharmaSUG 2003 Coder's Corner 25

Automatic Parameter Checking, Greg Silva, PhUSE 2005 TS08

CONTACT INFORMATION

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

Lawrence Heaton-Wright
Quintiles Limited
Station House, Market Street
Bracknell, Berkshire, RG12 1HX
United Kingdom
Work Phone: +44 1344 708320
Fax: +44 1344 708106
Email: lawrence.heaton-wright@quintiles.com
Web: www.quintiles.com