Paper 015-2011

# The Validator: A Macro to Validate Parameters

Steven A. Wilson, Gilead Sciences, Inc., Foster City, CA

## ABSTRACT

When developing reusable macro tools, one important aspect of coding such macros is how to validate the parameter values passed to the macro. While all developers should strive to include some validation of passed parameter values, the type of validation, messages returned and exception handling should be performed in a consistent fashion for all macros in your autocall library.

This paper discusses the development of a macro that is a utility for validating parameter values passed into other macros. This allows a consistent set of parameter validations to be performed with consistent results within your macro toolbox. Techniques for using this utility macro as a stand-alone tool as well as implementation as a stored-compiled macro are also discussed.

## INTRODUCTION

There have been many papers presented at SGF over the years that discuss how to develop a set of macros as an autocall macro library[1]. This allows any number of often-used code segments to be executed reliably by all users in a group. This is a widely recognized means of enhancing programmer productivity as well as producing reliable results.

Such autocall macro tools should be developed to a higher standard than an ad-hoc macro. This includes utilizing robust and efficient coding techniques, proper documentation and validation, exception handling and consistent behaviours.

This paper assumes knowledge of the SAS macro language and will focus instead on a single issue of consideration when developing a macro autocall toolbox – the validation of passed parameter values.

As a macro developer, you will often see the same type of objects being passed into you macros as parameters. Some examples are:

| | |
|---|---|
| - SAS data set name | %macro a ( data= ) ; |
| - SAS variable name | %macro b ( data=, var= ) ; |
| - integer with a specific range | %macro c ( month=, day= ) ; |
| - file location | %macro d ( infile= ) ; |
| - Y/N indicator | %macro e ( test=N ) ; |
| etc. | |

The proper validation of these passed parameter values is important, but as much as half the code in your macro may simply be validating the parameters passed into the macro.

Is it necessary to write a separate validation check for every parameter in every autocall macro that passes a SAS Data Set name? This paper argues 'No', and proposes a single utility macro that is useful for validating parameters passed to other macros. Taking this often-repeated validation code out of your macro has a number of advantages:

1) The validation code is written only once, to a high level of efficiency. For example, avoiding use of the SASHELP dictionary tables, which are easy to use for validation, but can be grossly expensive in terms of efficiency.

2) Performing validation of parameter values in newly developed autocall macros becomes easy. One simple macro call can fully validate a passed parameter.

3) Once the validation code is removed from your autocall macros, the remaining code is only the specific code needed to perform the macro's task. This makes your autocall macros easier to code, review and maintain.

Let's call our macro the `%validator`.

## DESIGN CONSIDERATIONS

Recall that the goal of the %validator macro is to make our validation checking, messages and exception handling consistent within all the macros in our autocall toolbox. As with any programming task, the design phase must address the specific requirements that have been requested of your application. For a utility of this nature, there are some important considerations and some automatic SAS System macro variables that can be used.

### WILL THIS MACRO BE USED ON MULTIPLE PLATFORMS? &SYSSCP

For example, if your macros are used on both Windows and Unix, this utility would need to have different behaviours. This automatic macro variable will tell you which operating system your program is running under. For example, the delimiter used on different operating systems may differ:

```
%if &sysscp = WIN %then
  %let dlm = \ ;
%else %if &sysscp = UNIX %then
  %let dlm = / ;
```

### WILL THIS MACRO BE USED UNDER DIFFERENT VERSIONS OF SAS? &SYSVER

Ever so often, code will need to be different depending on the version of SAS that it is executing under. Use this macro variable to detect the SAS version. For example, SAS 9.2 introduces a number of new functions that you may wish to incorporate in your code.

The CMISS function is one new function that can be used to count the number of missing values for a variable in a Data Set. Prior to 9.2, no such function existed, requiring you to compute the value manually:

```
data countmissing;
  set temp;
  array vars(8) $ disp1-disp8;
  %if &sysver = 9.2 %then
  %do ;
    /* Using CMISS, only one statement needed rather than 5: */
    missing = cmiss(of vars[*]);
  %end ;
  %else
  %do ;
    missing=0;
    do i = 1 to 8;
      if vars(i)=' ' then missing+1;
    end;
    drop i;
  %end ;
run;
```

### WILL THIS MACRO BE USED IN DIFFERENT ENVIRONMENTS? &SYSENV

Exception handling may be very different in a batch job from what is done in an interactive session.

```
%if &sysenv=BACK %then
%do ;
   ENDSAS ;   /* End SAS session immediately */
%end ;
%else %if &sysenv=FORE %then
%do ;
   %abort ;   /* Stop submitted code immediately */
%end ;
```

### HOW WILL EXCEPTIONS BE DETECTED? &SYSERR

SAS maintains an automatic macro variable that can be used to determine if there has been a SAS error in the submitted program. The values of this macro variable are a bit peculiar, with 1, 2 and 4 indicating non-error conditions.

```
%if &syserr=3 or &syserr gt 4 %then
   %put Error detected in the submitted program. ;
```

We will also define global macro variables in %validator to flag errors and contain messages. We follow a coding convention of using an underscore as the first character in globally defined macro variables in our utilities.

## %VALIDATOR MESSAGES

Another design issue for our %validator is to develop a standard methodology for detecting and reporting errors. Remember that our goal is to provide helpful messages that are consistent in appearance. Firstly, we will use a global macro variable flag and message, as well as the `&syserr` automatic macro variable:

```
%global  _valflag    /* Error or Warning flag  */
         _valmsg     /* Error or Warning message text.  */ ;

%if &syserr=3 or &syserr gt 4 %then
%do ;
   %let _valflag=E;  /* Indicate error condition  */
   %let _valmsg = Helpful error message. ;
   %put ERROR:***-------------------------------------***;
   %put ERROR: &_valmsg ;
   %put ERROR:***-------------------------------------***;
   %goto err_exit ;  /* End processing when an Error  */
%end ;

%nobs ( &data )  /* Autocall macro returns &_anyobs */
%if &_anyobs=Y %then
%do ;
   /* Only provide message when a Warning */
   %let _valflag=W ;  /* Indicate warning condition  */
   %let _valmsg = Data set contains 0 observations. ;
   %put WARNING: &_valmsg ;
%end ;
```

Second, we utilize our standard flag and link:

```
/*-------------------------------------------------*/
/*  Messages, program halting and clean-up.
/*-------------------------------------------------*/
%err_exit:
 %* When error encountered, stop execution of batch jobs. ;
 %if &sysenv=BACK %then
 %do ;
    ENDSAS ;   /* End SAS session immediately */
 %end ;

 %if &sysenv=FORE %then
 %do ;
    %abort ;   /* Stop submitted code immediately */
 %end ;
```

In order to provide better messages to the user, we can make these further enhancements to our design:

1) The message written into the SAS log should also indicate the parameter that has failed validation. We can implement a PARM_NAME parameter in our %validator for use in messages. This will contain the name of the parameter being validated.

2) A standard set of code should be used for writing the messages. Thus, another macro will be developed that will be used only for writing messages into the SAS log.

## SAS SYSTEM MESSAGES

In addition to the messages that %validator is writing into the SAS log, the SAS System itself may be writing various notes into the log depending on the settings of various SAS System options. These options include:

MPRINT MLOGIC SYMBOLGEN SOURCE NOTES

You may wish to turn all these off during the execution of %validator since the validation may generate a lot of text in the SAS log that would be confusing and irrelevant to the end user. But of course, after %validator has completed, you need to re-set these options to their original state.

Fortunately, this can be easily accomplished by storing the settings of these SAS System Options at the beginning of %validator, and then resetting these options at the end of %validator:

```
%let _symbolgen = %sysfunc(getoption(symbolgen)) ;
%let _mlogic    = %sysfunc(getoption(mlogic)) ;
%let _notes     = %sysfunc(getoption(notes)) ;
%if &_symbolgen = NOSYMBOLGEN %then %let _symbolgen=;
%if &_mlogic    = NOMLOGIC    %then %let _mlogic=;
%if &_notes     = NONOTES     %then %let _notes=;
%if %length(&_symbolgen&_mlogic&_notes) gt 0 %then
%do ;
   options
   %if %length(&_symbolgen) gt 0 %then nosymbolgen ;
   %if %length(&_mlogic)    gt 0 %then nomlogic    ;
   %if %length(&_notes)     gt 0 %then nonotes     ;
     ;
%end ;

…your %validator macro code…

/* Clean up: Re-set SAS system options.  */
%if %length(&_symbolgen&_mlogic&_notes) gt 0 %then
%do ;
      options &_notes &_mlogic &_symbolgen ;
%end ;
```

## OUTLINE OF A %VALIDATOR MACRO

Now we have the design issues resolved to allow us to create the skeleton of our %validator macro. We still do not have any code that actually performs any validation yet, but we do have the structure of how the validations will be processed:

```
%macro validator ( type=, parm=, parm_name=  ) ;
   /*-----------------------------------------------------*/
   /* Utility macro to perform validations.
   /*   TYPE      = Type of validation to perform  (DATA, VARIABLE, REQUIRED, etc.)
   /*   PARM      = Parameter value to be validated
   /*   PARM_NAME = Optional name of parameter
   /*-----------------------------------------------------*/

   %let _valflag = ;    /* Flag set for Error or Warning condition      */
   %let _valmsg  = ;    /* Contains message to be written to SAS log    */
   %let _optset  = ;    /* Flag whether SAS System Options need re-set  */

   /*-----------------------------------------------------*/
   /* Examine for current SAS error condition.
   /*-----------------------------------------------------*/
   %if &syserr=3 or &syserr gt 4 %then
   %do ;
      %let _valflag=E;
      %if &_sysenv = BACK %then
        %let _valmsg = Ending SAS because of an error in the submitted program. ;
      %else
        %let _valmsg = Macro canceled because of error in the submitted code. ;
      %goto err_exit ;
   %end ;
```

```
   /*-----------------------------------------------------*/
   /* Turn off SYMBOLGEN (or other options)               */
   /*-----------------------------------------------------*/
   %let _symbolgen = %sysfunc(getoption(symbolgen)) ;
   %if &_symbolgen = NOSYMBOLGEN %then %let _symbolgen=;
   %if %length(&_symbolgen) gt 0 %then
   %do ;
      %let _optset=1 ;  /* Flag that SAS System Options need to be re-set  */
      options
      %if %length(&_symbolgen) gt 0 %then nosymbolgen ;
          ;
   %end ;

   /*-----------------------------------------------------*/
   /* Perform validations here. This will be the bulk     */
   /* of the code in this %validator macro.               */
   /*-----------------------------------------------------*/
   /* Validate TYPE=DATA    SAS Data Set                  */
   %if &type=DATA %then
   %do ;
      %if %sysfunc(exist(&parm,data)) = 0 %then
      %do ;
         %let _valflag=E;
         %let _valmsg = SAS Data Set does not exist: &parm ;
         %goto err_exit ;     /* Stop processing */
      %end ;

      %nobs ( &data )  /* Autocall macro returns &_anyobs */
      %if &_anyobs=Y %then
         %let _valflag=W;
         %let _valmsg = Warning message. ;
         %logmsg ;           /* Write message to log and continue  */
      %end ;
   %end ;


   /*----- LOTS of additional validation code -----*/

   /*-----------------------------------------------------------------*/
   /*  Messages, program halting and clean-up.                        */
   /*-----------------------------------------------------------------*/
  %err_exit:
   %if %length(&_valflag) = F %then
   %do ;
      %log_msg    /* Write the error message into the SAS log  */
      %if &sysenv=BACK %then
      %do ;
         ENDSAS ;    /* Immediately end batch jobs when an error  */
      %end ;
   %end ;

   /* Re-set the SAS System options that control macro output in the log.  */
   %if %length(&_symbolgen) gt 0 and &_optset=1 %then
   %do ;
         options &_symbolgen;
   %end ;

   %if &_valflag = F and &sysenv=FORE %then
   %do ;
      %abort ;    /* Immediately stop interactively submitted code when an error  */
   %end ;
%mend validator ;
```

A previously mentioned, we will also develop a separate macro that is used only to write consistently formatted messages into the SAS log:

```
%macro log_msg ;
   /*-------------------------------------------------*/
   /* Utility macro to write message to the SAS log
   /*-------------------------------------------------*/
   %if _valflag=E %then %let str1 = ERROR;
   %else %let str1 = WARNING;
   %put &str1:***-------------------------------------***;
   %put &str1: &_valmsg ;
   %put &str1:***-------------------------------------***;
%mend log_msg ;
```

Now we can develop new macros that utilize our %validator macro. This allows us to concentrate on the task that macro is to perform rather than writing a lot of code to validate that the autocall macro has received all the necessary parameters

```
%macro new_macro ( data= ) ;
   /*-------------------------------------------------*/
   /*  Validate the parameters
   /*-------------------------------------------------*/
   %validator ( type=DATA , parm=&data, parm_name=DATA )

   /*-------------------------------------------------*/
   /* Remaining code in your macro is now specific to
   /* the task at hand and does not require code for
   /* validating passed parameters or writing messages.
   /*-------------------------------------------------*/
   %put Are you still awake ? ;
%mend new_macro ;
```

## VALIDATIONS TO BE PERFORMED

A review of your macro toolbox will show that there are many common parameter types that are in use by your autocall macros. These parameter values should have validations performed, including but not limited to, the following:

- Values
  - Non-null value is required
  - Numeric integer
  - Integer in a range of values
  - Single word
  - Value must be in a set of acceptable values
  - Yes/No indicator
  - Valid date in a specific range
- SAS Libref
  - Must be a valid libref
  - Must be an assigned libref
  - User must have write-access to the library

- SAS data set
  - May be one-part or two-part name
  - May contain data set options, including the WHERE= data set option
  - Valid SAS Data Set name
  - Data Set must exist
  - Data Set must contain data
  - Write access to Data Set
- SAS Variable
  - One or more valid SAS variable name
  - Variable must exist in a specified SAS Data Set
  - Must be a Numeric/Character variable
  - Must contain a specific value or only a set of acceptable values
  - Format and Informat associated with the variable must be on the SAS format search path
  - Variable contains data that does not have a format label
- SAS Format
  - Valid format/informat name
  - Format/informat exists on the current format search path
- Directory Name
  - Existing directory
  - Write access to the directory
- File Name
  - Existing external file
  - Write access to the file

Consider in your own environment how many macros you have with common parameter types. For example, how many macros in your autocall toolbox have a parameter used to pass a SAS Data Set name? If you are validating that passed parameter value – and you should always be validating parameter values in an autocall macro – you are doing the same validation in each of your autocall macros. Gathering this often-repeated code into one location is a perfect situation for implementing a macro.

The next sections show some example code to perform validations. We will consider validating values separately from validating other types of objects.

## VALIDATING VALUES

When validating values, we can be either validating the value passed in the parameter, or we can be asked to validate the values in a SAS Data Set. Let's first consider validating parameter values.

Parameter values can be classified into various categories, as detailed in the validations to be performed in the previous section. Here are a set of common validations of parameter values:

1) Indicator parameter value must be Y or N

Many types of parameters are Y/N values, where the parameter is used to indicate one of two conditions. Macro parameters that pass indicators should not be case-sensitive and should not be overly restrictive. One simple statement allows greater flexibility when using such a parameter:

```
/* Accept values of Y,N,YES,NO,Yes,No,y,n,yes,no */
%let _yn = %upcase(%substr(&parm,1,1)) ;
```

This could then easily be followed with a quick validation of the passed value:

```
/* Abort when not an acceptable value. */
%if &_yn ne Y and &_yn ne N %then
%do ;
   %let _valflag=E;
   %let _valmsg = Parameter PARM is invalid: &yn ;
   %goto err_exit ;
%end ;
```

2) Parameter value must be in a set of acceptable values or range.

This is a validation that is a more general case of the condition above. A special parameter ACCEPTABLE= has been added to %validator to accommodate this type of validation.

```
/* Verify the parameter value is an acceptable value  */
%let i = 1 ;
%let _valflag = F;
%do %while(%scan(&acceptable,&i,%str( )) ne %str() and &_valflag=F ) ;
   /* Examine specific acceptable values  */
   %if %scan(&acceptable,&i,%str( )) = &parm %then
     %let _valflag = ;
   %else %if %index(%scan(&acceptable,&i,%str( )),%str(-)) %then
   %do ;
      /* Examine a range when the acceptable values contains a dash  */
      %if %scan(%scan(&acceptable,&i,%str( )),1,%str(-)) le &parm le
          %scan(%scan(&acceptable,&i,%str( )),2 %str(-)) %then
           %let _valflag = ;
   %end ;
   %else %let i = %eval(&i+1) ;
%end ;
%if &_valflag = F %then
%do ;
   %let _valmsg = PARM value of &parm is not an acceptable value: &acceptable  ;
   %goto err_exit ;
%end ;
```

3) Parameter value is required.

This type of validation is so common that we may with to include a REQUIRED= parameter in our %validator macro to indicate that this validation should be performed in addition to any other validation specified in the macro call.

```
/* Abort when a required parameter is not supplied. */
%if &required=Y %then
%do ;
   %if %length(&parm) = 0 %then
   %do ;
      %let _valflag=E;
      %let _valmsg = Value required for parameter PARM ;
      %goto err_exit ;
   %end ;
%end ;
```

4) Parameter value must be an integer.

```
/* Verify that the parameter value is an integer. */
%if %verify(&parm,0123456789) %then
%do ;
   %let _valflag=E;
   %let _valmsg = Value in PARM is not an integer. ;
   %goto err_exit ;
%end ;
```

5) Parameter value must be a single word with no special characters.

```
/* Verify the parameter value is only one word  */
%if %length(%scan(&parm,2,%str( ))) ne 0 %then
%do ;
   %let _valflag = F;
   %let _valmsg = PARM must be a single word. ;
   %goto err_exit ;
%end ;

/* Look for special characters.  */
%else %if %verify(%upcase(&parm),
          ABCDEFGHIJKLMNOPQRSTUVWXYZ.1234567890) gt 0 %then
%do ;
   %let _valflag = F;
   %let _valmsg = PARM must be alphanumeric. ;
   %goto err_exit ;
%end ;
```

6a) Parameter value (or values) must be found in a SAS Data Set.
6b) All values in a SAS Data Set must be among a set of acceptable values.

These validations are closely related and can be performed using the same set of code with the only difference being the insertion of the NOT operator. While the code is a bit more complex, remember, we code this validation once and can use our utility to perform the validation easily.

```
%if &type=VALUE %then %let _not=;  /* PARM value must be in data set */
%else %let _not= NOT ;  /* All data set values must be among PARM values  */

%let _dsid=%sysfunc(open(&data));  /* DATA parameter from %validator call */
%if &_dsid=0 %then
%do ;
   %let _valflag=E;
   %let _valmsg = &data cannot be opened. ;
   %goto err_exit ;
%end;
%let _varnum=%sysfunc(varnum(&_dsid,&var)); /* VAR parameter in %validator */
%if &_varnum=0 %then
%do ;
   %let _valflag=E;
   %let _valmsg = The variable &var does not exist in data set &data.. ;
   %goto err_exit ;
%end;
%let _vartype=%sysfunc(vartype(&_dsid,&_varnum));
%let _dsid = %sysfunc(close(&_dsid)) ;

/* Construct a WHERE statement to subset &data.  */
/* For numeric variables, allow a range of acceptable values, eg. 1-99  */
%if &_vartype=N and %index(&parm,%str(-)) gt 0 %then
  %let _where = WHERE &_not (%scan(&parm,1,%str(-)) le &var le
                %scan(&parm,1,%str(-))) ;
%else %let _where = WHERE &var &_not in (&parm) ;
%let _ckfound = 0 ;
%let _ckval = ;

/* A SAS View is used because the passed data set may have a WHERE=   */
/* data set option. The Data Step fails when there is both a WHERE=   */
/* data set option and a WHERE statement.                             */
PROC SQL ;
  CREATE VIEW _TEMP AS
   SELECT *
   FROM &data ;
QUIT;
```

```
DATA  _NULL_ ;
  SET  _TEMP ;
  &_where ;  /* Insert the WHERE clause that was constructed above.  */
  /* Observation in &data meets the subsetting criteria.        */
  IF _N_=1 THEN DO;
     /* When looking for unacceptable values, obtain      */
     /* the invalid value for presenting in the message.  */
     CALL SYMPUT('_CKFOUND', '1') ;
     %if &_vartype=C %then
       CALL SYMPUT('_CKVAL', &var) ;
     %else
       CALL SYMPUT('_CKVAL', PUT(&var,BEST.)) ; ;
  END ;
  STOP ;
RUN ;

/* Delete the SQL View */
PROC SQL ;
  DROP VIEW _TEMP ;
QUIT ;

%if &_ckfound = 1 /* Record found */ and "&_not"=%str("NOT") %then %do ;
   %let _valflag=E;
   %let _valmsg = &var in &data contains an non-permissable value: &_ckval ;
   %goto err_exit ;
%end;
%else %if &_ckfound = 0 %then %do;   /* No records in the subset of &data  */
   %if %str(&_not)=%str() %then
   %do ;
      %let _valflag=E;
      %if %scan(&parm,2) ne or
          (&_vartype=N and %index(&parm,%str(-)) gt 0) %then
        %let _valmsg = None of the values are found in &var in &data: &parm ;
      %else
        %let _valmsg = Value &parm is not found in &var in data set &data.. ;
      %goto err_exit ;
   %end;
%end;
```

## VALIDATING SAS OBJECTS

When validating SAS objects (i.e. data sets, formats, librefs, etc.), be aware of whether you need to validate whether the object exists or if it is the name of an object to be created by the autocall macro.

1) Parameter value must be a valid SAS name.

There is a SASNAME function in SAS, but no corresponding %sasname function, so we can build our own:

```
/* Abort when not a valid SAS name. */
/* PARM must be alphanumeric, begin with a letter or  */
/* an underscore and be 32 characters or less.        */
%if %verify(%upcase(&parm),
               ABCDEFGHIJKLMNOPQRSTUVWXYZ_1234567890)
 or
   %verify(%upcase(%substr(&parm,1,1)),
               ABCDEFGHIJKLMNOPQRSTUVWXYZ_)
 or
   %length(&parm) gt 32  %then
%do ;
   %let _valflag=E;
   %let _valmsg = &parm is not a valid SAS name. ;
   %goto err_exit ;
%end ;
```

2) Validating a valid format name or informat name is a bit trickier, but essentially the same type of code as above. For these cases, I will refer you to the comprehensive paper "OBJECT_EXIST: A Macro to Check if a Specified Object Exists" by Jim Johnson, as presented at PharmaSUG 2010[2].

3) Validate that the parameter value is the name of a SAS object that currently exists. This is actually an easy task, as there are many SAS functions that can be used to do these types of validations.

Mr Johnson's paper mentioned above has lots of sample code for validating various SAS objects, including code for validating SAS Data Sets, variables in a data set, defined formats or informats, LIBREFs, FILEREFs, macros and macro variables. I refer you to this excellent paper rather than repeat the contents here.

Depending on how extensive and flexible you want %validator to be, you can implement a number of checks on a parameter. For example, when validating a parameter that contains a SAS Data Set name you may wish to examine various aspects such as
- is the library is allocated
- is the passed value a valid SAS data set name
- if whether the data set is empty
- allow the passed data set to have data set options, including the WHERE= option
- can data set can be opened with a member-level lock. A failure of this would indicate the data set is corrupt or in use by another user or process:

```
/* Determine if data set is locked by another user or process. Warning only.  */
%let _dsid = %sysfunc(open(&_lib..&_data(cntllev=mem),i)) ;
%if &_dsid = 0 and %upcase(&_lib) ne WORK %then
%do ;
   %let _valflag=W;
   %let _valmsg = Data set &parm is locked by another user. %sysfunc(sysmsg()) ;
   %logmsg
%end ;
%else %let _dsid = %sysfunc(close(&_dsid)) ;
```

### VALIDATING NON-SAS OBJECTS

In some instances, the validation you need to perform validation of non-SAS objects, such a whether a directory name is valid or whether an external file exists. There is also the ability to verify that the executing program has write access to the directory or external file. Again, I will refer you to the comprehensive paper "OBJECT_EXIST: A Macro to Check if a Specified Object Exists" by Jim Johnson, as presented at PharmaSUG 2010.

## IMPLEMENTING AS A STAND-ALONE MACRO

As we have developed our validation macro, we have noted the need to identify the parameter that has caused the failure or warning. But if we wish to allow this %validator macro to be used as a stand-alone macro, a parameter name may not be relevant. Accomplishing this requires only a small change to how messages are created by %validator:

```
/* Abort when not an acceptable value. */
%if &_yn ne Y and &_yn ne N %then
%do ;
   %let _valflag=E;
   %let _valmsg = Value must be Y or N. ;
   %if %length(&parm_name) ne 0 %then
     %let _valmsg = &_valmsg See parameter &parm_name..;
   %goto err_exit ;
%end ;
```

## IMPLEMENTING AS A STORED COMPILED MACRO

By the time we have completely developed our %validator, we have written perhaps a huge amount of macro code. Since %validator will also be called by nearly all of the macros in our autocall toolbox, it may make sense to save our %validator as a stored-compiled macro.

A stored-compiled macro is a macro that has already been compiled and is ready for execution. This facility is most useful when there is a large amount of macro code that will be often used.

This is accomplished by adding an option onto the %macro statement. Notice also that we have added the additional parameters to %validator that have been discussed in this paper.

```
%macro validator (
    type      =,    /* Type of validation to be performed. May be among:     */
                    /*   DATA, VARIABLE, INTEGER, WORD, SASNAME, REQUIRED, … */
    parm      =,    /* Value to be validated                                 */
    parm_name =,    /* Optional parameter name                               */
    acceptable=,    /* Optional acceptable values for the &parm              */
    data      =,    /* Data set to examine when validating a variable        */
    var       =,    /* Variable to examine when validating a value           */
    required  =N    /* Is &parm required to be non-null?                     */
                )
                 / store mstore=libref  ;    /* Store the compile code here */

   …lots of macro code…

%mend validator ;
```

Note that our %macro statement now has an option: / store mstore=*libref* . This instructs the macro facility to store a copy of the compiled macro in the catalog *libref*.sasmacr. Once saved, you must tell SAS that you wish to use Stored Compiled Macros by setting two SAS System options:

```
options mstored sasmstore=libref ;
```

When using the Stored Compiled Macro Facility, it is important to remember that the stored compiled macros will take precedence over the autocall search path, but not over macros defined in the current session.

## CONCLUSION

There have been many numbers of papers presented at SGF over the years that discuss how to develop a set of macros as an autocall toolbox. This paper has presented a technique that can be easily implemented to enhance your autocall macros to make them more robust and efficient while also increasing the productivity of your users by providing useful and specific messages when an error occurs.

## REFERENCES

[1]Wilson, Steven (1994), "Developing a SAS System Autocall Macro Library as an Effective Toolkit", Proceedings of the 19th Annual SAS Users Group International Conference (Applications Development: Best Contributed Paper)

[2]Johnson, Jim (2010), "OBJECT_EXIST: A Macro to Check if a Specified Object Exists", Proceedings of the PharmaSUG 2010 Conference Proceedings (Tutorials Paper TU01)
See www.pharmasug.org/cd/papers/TU/TU01.pdf

## ACKNOWLEDGMENTS

The author would like to express his sincere gratitude to all the SAS macro experts that he has been fortunate to work with during his career as a SAS applications developer. These include Dr. James Schwenke, Dr. Tom Hoffman, Amy Lin, Roger Staum, John Brega, Linda Collins and Dr. Martin Rosenberg. Thanks to all of you for your support and encouragement over the years. You all have helped me enjoy a wonderful career using SAS.

## CONTACT INFORMATION

Steve Wilson is a Sr. Manager of Statistical Programming in the Applications Development group at Gilead Sciences, Inc. Steve has been developing applications in health sciences industry using the SAS Macro Facility for over 25 years. During this time he has given numerous presentations at various SAS user group conferences and was Program Chair of the 1997 Western Users of SAS Software conference in Universal City, CA.

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

| | |
|---|---|
| Name: | Steven A. Wilson |
| Enterprise: | Gilead Sciences, Inc. |
| Address: | 333 Lakeside Drive |
| City, State ZIP: | Foster City, CA 94404 |

Work Phone: (650) 522-4569
E-mail: steve.wilson@gilead.com
Web: www.gilead.com