Automatic Macro Parameter Checking

Greg Silva – Associate Director, Statistical Programming and Operations
Biogen Idec Limited, Thames House, Foundation Park, Maidenhead, SL6 3UD

Introduction

As small pieces of SAS code evolve into large and complex SAS macros, the developer is tasked with providing the required functionality, as well as protecting the users from themselves. No matter how efficient or 'cool' a macro is, if it produces the incorrect results, it's not very useful. The developer has control of what is written in the macro, but is at the mercy of the user when it comes to parameters.

Background

When it comes down to it, writing macros that use parameters can have a lot of issues. When developing macros for any system wide use the following rule should always be applied:

Parameters need to be checked.

Macro parameters should be checked before any "real" processing begins. If you decide that everyone knows that the parameter "patients" means "provide the macro with the number of patients to analyze," then you need to be sure that it is well documented. Even then, users may enter incorrect values.

The obvious answer is to check each parameter in the macro before starting the "real" processing. This allows the user to get instant feedback if there is a problem, as well as insuring that there will be no results generated based on incorrect parameter values.

One of the easiest ways to educate developers and users to the meaning of a variable is to have some standard for the variable names. With the advent of version 7 of SAS, the macro variable name lengths have been extended to 32 characters.

In C, Visual Basic and other languages a standard has emerged for naming variables. It is known as Hungarian Notation. It simply adds a prefix to a variable name that identifies what" type" of variable it is. Some possibilities include:

is = YES/NO
num = Numeric

To finish the automatic parameter checking methodology, the SAS system provides an option and an automatic macro variable (PARMBUFF and &SYSPBUFF, respectively). This combination allows for the capture of macro parameters passed to a macro.

The code that follows shows how to use this information.

The Macro Code

```
%macro check_parameters(lstParameters, chrMacroName);
   %let errorz = 0;
   %let _done  = 0;
   %let _count = 1;
```

The macro reads in a string through the lstParameters parameter. A '$$=$$' is added to show the end of the list of parameters. This is a simple way to know when you are done parsing through a string (or macro variable). The two characters removed from the length are the parentheses that enclose the list of parameters.

```
    %let length      = %eval(%sysfunc(length(&lstParameters)) -2) ;
    %let parameters  = %sysfunc(substr(&lstParameters, 2, &length))%str(, $$=$$) ;
```

The string is parsed through to get the name-value combinations for parameter = value.

```
    %do %until(&_done) ;
       %let parm_name  = %scan(%nrquote(&parameters), &_count, %nrquote(,)) ;
       %let parm_value = %scan(&parm_name, 1, %str(=)) ;

       %if (&parm_value = %nrquote($$)) %then %let _done = 1 ;
```

This section of the code checks the parameter prefix to see if the parameter is of a certain type. Based on the prefix, the parameter is checked by the appropriate type checking macro.

```
       %else %do ;
          %if       %substr(&parm_ value, 1, 2) = is  %then %yesno_parameter(&parm_value) ;
          %else %if %substr(&parm_ value, 1, 3) = chr %then %char_parameter(&parm_value) ;
       %end ;
    %** Increment the parameter count. ;
    %let _count = %eval(&_count + 1) ;
    %end ;
```

Each parameter checking macro will return an error message about that "type" of macro parameter. At the end of all of the checking, a single standard message is used inform you that there is a problem with one or more parameters. At this point you know all of the parameters that have invalid values, as well as the macro that used them.

```
   %if (&errorz = 1) %then %do ;
   %put ----- There was 1 error found in parameters defined for &chrMacroName..
              The application will stop. ;
   %end ;
   %else %if (&errorz > 1) %then %do ;
      %put ----- There were &errorz errors found in parameters defined for &chrMacroName..
                 The application will stop. ;
   %end ;
%mend;
```

Your error checking macro could be as simple as this.

```
%macro yesno_parameter(chrValue) ;
   %if %upcase(&&&chrValue) ^= YES and %upcase(&&&chrValue) ^= NO %then %do ;
      %put ERROR: Invalid Value [%str(&&&chrValue) ] for parameter &chrValue in macro
                                                                  &chrMacroName   ;
      %let errorz = %eval (&errorz + 1) ;
   %end ;
%mend ;
```

How To Use It

As you write your macro, simply add the parmbuff parameter to end of the macro's parameter list.

```
%macro combine(chrProtocolName          =Test,
               isDummyRun               =no,
               chrSubjectIDRange        =,
               numTreatments            =,
               allocation_of_treatments =,
               isStratified             =no,
               numList                  =1) / parmbuff;
```

The only addition code needed to check all parameters with standard prefixes is this simple macro.

```
%*** All parameter checking here... ;
   %check_parameters(&syspbuff, Combine)
                  .
                  .
                  .
%mend ;
```

Running This…

The following code assumes that you have checks for only the boolean (YES/NO) variable type.

```
%Random (chrlName        = Testing,
         isDummyRun      = no,
         isStratified    = 0 ,
         isnumList       = )
```

Produces This…

```
ERROR: Invalid Value [0] for parameter
       isStratified in macro RANDOM.

ERROR: Invalid Value [ ] for parameter
       isNumList in macro RANDOM.

----- There were 2 errors found in
      parameters defined for RANDOM.
      The application will stop.
```

Conclusion

By developing and using standard naming conventions on macro parameters, you have a better understanding of the system through the types of data expected in each macro call.

Incorporating standard components of the SAS system, including PARMBUFF and &SYSPBUFF, will allow you to easily add standard checks on parameters to all new macros that you are writing.

From a developer's perspective, this means less code to write for parameter checking and less code to validate. The biggest advantage for everyone is a better understanding of what goes into a macro, so that there is less chance of erroneous values being generated by the macro.

Contact Information

Greg Silva

Biogen Idec

Greg.silva@biogenidec.com