Bulletproofing and Knowledge Encapsulation in Statistical Macros

John K. Troxell, Merck & Co., Inc., Rahway, NJ

ABSTRACT

Most SAS® programmers agree that SAS macros are a wonderful tool for automating repetitive tasks. More controversial is bulletproofing (making a macro robust to incorrect parameters or data). Less commonly considered is the potential that macros offer for delivery of statistical expertise. Statistical knowledge can be built into macros with advanced rules for validating macro parameter values, implementation of good statistical practice, implementation of otherwise unavailable statistical methods, or delivery of data-driven advice and interpretation in English. Pros and cons of bulletproofing and knowledge encapsulation are discussed, and some specific techniques are presented. Many of the techniques are useful in non-statistical macros as well. Examples are drawn from macros for stepwise regression analysis and analysis of covariance.

INTRODUCTION

This paper discusses two ways to add value to macros beyond automation. It is not the intent of the author to recommend their use in all circumstances. Rather, the intent is to discuss the possibilities in the hope that the reader will make an informed decision.

The unifying theme of the paper is that, at least in some cases, it is desirable to incorporate some knowledge into the design of the macro, so that the macro is something more valuable than merely a labor-saving device. How and when, if ever, to do this is the subject of the rest of the paper.

There is a spectrum of techniques and approaches any one or more of which might be considered. Bulletproofing is described first, followed by a broader collection of ideas in the section on knowledge encapsulation.

BULLETPROOFING

This section starts with a definition of bulletproofing, continues with a discussion of some advantages and disadvantages of bulletproofing, and finishes with a series of examples.

Bulletproofing typically involves checking parameter values to see if they make sense. It also may involve confirming assumptions about the session environment and the input data. When problems are found, informative error messages can be printed to the log and the macro should clean up and exit gracefully.

WHY BULLETPROOF?

A macro might malfunction or not behave as intended if given incorrect input. The same might occur if assumptions about the environment in which it the macro is run are not satisfied.

If the macro malfunctions, it might produce error and warning messages and leave the SAS session in an undesirable state. Or, potentially far worse, the malfunction might not be noticed because no error messages are generated, and yet the results, say some statistical analyses, might be incorrect.

Although in general it is probably impossible for a programmer to guarantee that a macro will never malfunction in any circumstance, it most certainly *is* possible to prevent *some* malfunctions through the use of bulletproofing.

Bulletproofing can also speed debugging of calling programs. It can aid learning of the correct calling syntax - it is like a built-in teacher if messages are helpful and informative. Bulletproofing is built into the macro, so it is always there even when the documentation is not available or not read.

DRAWBACKS TO BULLETPROOFING

Bulletproofing requires additional programming effort. It increases length and complexity of code. One has to be careful not to introduce errors in the bulletproofing code. Finally, one has to be careful only to catch true errors, and not prevent legitimate use of the macro.

BULLETPROOFING EXAMPLE 1: SAS VERSION

Some statistical PROCs behave differently under different versions of SAS. One might wish to prevent use of a macro under SAS versions other than those in which the macro was tested. This is one way to do this:

```
%if not (    "&sysver" eq "6.12"
          or "&sysver" eq "8.00"
          or "&sysver" eq "8.1"  ) %then %do;
    %put;
    %put *** MACANOVA &er&ror: Running under SAS
version &sysver;
    %put *** MACANOVA has not been validated for
this release;
    %put;
    %goto exit;
%end;
```

&er was defined as ER and &ror was defined as ROR; the source code does not contain the word ERROR but the log will if it should. %goto is ugly but works; label %exit: is near the end of the macro. &sysver is not consistent in the number of characters to the right of the decimal point. Quotes are probably not necessary here except to ensure character comparison; the quotes are included.

A useful variation on this technique is to branch conditionally in the macro based on the SAS version. For example, a macro might have to parse PROC GLM output in SAS 6.12 to access certain statistics, whereas in SAS 8, all statistics are available in datasets through the use of the ODS feature.

BULLETPROOFING EXAMPLE 2: PARAMETER VALUES

A typical use of bulletproofing is to examine macro parameter values:

```
%if %length(%sysfunc(compress(&sstype,' '))) eq
0 %then %do;
    %put;
    %put *** MACANOVA: SSTYPE parameter is
missing, defaulting to 3;
    %put;
    %let sstype = 3;
%end;
```

%sysfunc can call almost any data step function, and other useful functions as well. %sysfunc(compress()) is used rather than the autocall macro %compress because it will work even if the user has not installed the SAS autocall library. Blanks are compressed out in case the parameter is a blank, eg. if sstype=%str( ).

```
%if &sstype ne 3 and &sstype ne 2 %then %do;
    %put;
    %put *** MACANOVA &er&ror: SSTYPE parameter
is not equal to 2 or 3;
    %put;
    %let badflag = 1;
%end;
```

```
%if    %sysevalf(&alphabtw le 0)
    or %sysevalf(&alphabtw ge 1) %then %do;
    %put;
    %put *** MACANOVA &er&ror: ALPHABTW
parameter out of range 0 < ALPHABTW < 1;
    %put;
    %let badflag = 1;
%end;

%if %sysevalf(&equi_l ge &equi_u) %then %do;
    %put;
    %put *** MACANOVA &er&ror: EQUI_L parameter
value &equi_l is not less than EQUI_U parameter
value &equi_u;
    %put;
    %let badflag = 1;
%end;
```

%sysevalf ensures floating point rather than character comparison; %eval treats numbers as integers. &badflag will be checked later on.

BULLETPROOFING EXAMPLE 3: INPUT DATA
In this example, checks are done to make sure that a specified input dataset exists and that it contains a specified variable:

```
%if %sysfunc(exist(&data)) ne 1 %then %do;
    %put;
    %put *** MACANOVA &er&ror: DATA parameter
invalid: dataset &data does not exist;
    %put;
    %let badflag = 1;
%end;
```

&data was a poor choice for a macro parameter or variable name, because data is a reserved word. One can also check for the pre-existence of temporary WORK datasets that are created by the macro for internal use, and exit if they already exist, in order to prevent inadvertent overwriting of user data. Such datasets should be probably be cleaned up (deleted) before the macro exits.

```
data _in;
     set &data;
     run;

%let dsid=%sysfunc(open(_in));
%let
varnum=%sysfunc(varnum(&dsid,%upcase(&trt)));
%if &varnum eq 0 %then %do;
    %put;
    %put *** MACANOVA &er&ror: TRT variable &trt
does not exist in dataset &data;
    %put;
    %let badflag = 1;
%end;
%let rc=%sysfunc(close(&dsid));
```

Don't forget to close( ) what you open( ). One can also determine the number of observations, variable type, length and format, and other properties of the data using this kind of code. The code in this example and in the other examples works under SAS version 6.12 as well as SAS 8.

BULLETPROOFING EXAMPLE 4: REQUIRED MACRO
This example checks whether a required macro is available to the SAS session:

```
proc sql noprint;
     select distinct objname
     into :macnames separated by ' '
     from dictionary.catalogs
     where memname in ('SASMACR')
       and objtype in ('MACRO')
       and objname in ('LEVENE');

%if "&macnames" ne "LEVENE" %then %do;
    %put;
    %put *** MACANOVA &er&ror: Required
homogeneity of variance testing macro LEVENE;
    %put *** is not available.;
    %put;
    %let cleanout=1;
    %goto exit;
%end;
```

This code only detects macros that have been defined or %INCLUDEd or AUTOCALLed during the session. It does not detect macros not yet invoked but in an AUTOCALL library. "separated by ' ' " is not necessary here because there is only one macro being checked for, but is useful when the list of macros is longer.

BULLETPROOFING EXAMPLE 5: REQUIRED FILEREF
This code checks whether a fileref is valid:

```
%if %sysfunc(fileref(&macroref)) eq 0 %then %do;
    %if &debug eq Y %then %put MACROREF fileref
&macroref validated;
%end;
%if %sysfunc(fileref(&macroref)) lt 0 %then %do;
    %put;
    %put *** MACANOVA &war&ning: MACROREF
fileref &macroref is defined but invalid;
    %put;
%end;
%if %sysfunc(fileref(&macroref)) gt 0 %then %do;
    %put;
    %put *** MACANOVA &war&ning: MACROREF
fileref &macroref is not defined;
    %put;
%end;
```

&war was defined as WAR and &ning was defined as NING; the source code does not contain the word WARNING but the log will if it should.

Many other kinds of bulletproofing checks can be made; these examples are provided to convey the general idea and a few techniques the author has found useful.

KNOWLEDGE ENCAPSULATION

One can't write any program or macro without knowing what it is supposed to accomplish. But in this section "knowledge" can mean something beyond the knowledge that it takes to write an ordinary program. The focus of this section is on ways that subject matter expertise of the programmer or statistician can be somehow incorporated into (encapsulated within) the macro itself, to the benefit of future users.

ENCAPSULATION EXAMPLE 1: ANOVA TABLE/CONTRASTS
As discussed earlier, one way to incorporate knowledge is to foresee problems and disarm them with appropriate bulletproofing.

Bulletproofing can get quite statistical in nature. For example, many users of PROC GLM might not realize that the SS2 option on the MODEL statement (for Type II sums of squares) affects the ANOVA table, but not the output of the LSMEANS, CONTRAST and ESTIMATE statements. This can be important in certain circumstances. The following code, when used to perform an analysis of data from a trial with two levels of the treatment variable, can produce an inconsistency between the p-value in the ANOVA table and the p-value in the ESTIMATE statement:

```
proc glm;
     class center trt;
     model y = trt center trt*center / ss2;
     estimate 'trtest' 1 -1;
```

In a macro that performs fixed effects analysis of covariance using PROC GLM, and which permits either Type II or Type III sums of squares, it is possible to anticipate this problem. One can parse the specified model statement to see if there is an interaction with the treatment variable, check whether the specified sum of squares type is II, and check whether contrasts are requested. If all three are true, one can alert the user about the inconsistency and prevent the macro from generating the contrasts.

ASSUMPTION CHECKING AND INTERPRETATION

In a typical Phase III clinical trial, two key assumptions of analysis of variance are homogeneity of variance and normality of residuals. These assumptions are commonly tested. The user of an analysis of covariance macro in this setting, probably a statistician or statistical programmer, is well aware of the need to check these assumptions and the methods of doing so. The assumption of independence of residuals is not usually violated and is examined less often. And the assumption that the independent (predictor) variables are truly independent is to a great extent guaranteed by the fact that the experiments are designed well. Therefore, other than perhaps providing the functionality of performing the tests of the normality and homogeneity assumptions, there is no particular need for the analysis of covariance macro to provide interpretation of results.

By contrast, in industrial settings, stepwise multiple regression or multiple regression is often used by non-statisticians to analyze the results of historical plant operating data for the purpose of building a descriptive model. A macro to perform stepwise regression for these users, who are typically engineers, can add tremendous value by checking assumptions, interpreting results and providing advice in common language, eg. English.

In this industrial setting, two key assumptions are independence of predictor variables and independence of residuals. Both are violated frequently. And there are many other things that a statistician would look at in the iterative process of empirical model-building that a non-statistician would probably be unaware of. In fact, there are so many things that can be looked at that even statisticians don't bother doing all of them because it requires extra programming work and time to review results. The programmer can add value here in many ways.

ENCAPSULATION EXAMPLE 2: COLLINEARITY

As discussed above, a big problem in unplanned historical plant operating data is patterns of correlation among groups of potential predictor variables. When the non-independence is severe, it can inflate the variances of the regression parameter estimates, hindering model selection and leading to unreliable regression model coefficients.

PROC REG can produce the collinearity diagnostics of Belsley, Kuh and Welsch (1980). These can be used to identify groups of predictor variables that are too closely involved in patterns of correlation. Unfortunately, the typical engineer is not aware of the need for, the existence of, the way to obtain, and the way to interpret these diagnostics (which are presented in the form of a matrix of numbers).

Prior to SAS 7, the collinearity diagnostics in PROC REG were available only in the listing file, although it is possible to reproduce them using PROC PRINCOMP and a subsequent data step. In SAS 8 they are available in a dataset through the ODS feature.

It is relatively straightforward to write code that can implement criteria similar to those proposed by Belsley, Kuh and Welsch for interpreting the diagnostics and present the results in understandable English messages to the listing and/or log, to the effect of "**Warning: Candidate predictor variables A, B, D and G are too closely involved together in a linear relationship, so that they do not provide independent information. This can lead to poor model selection and/or unreliable and unrealistically large model coefficients.** " The attention of the engineer can thus be drawn to the problem, and the engineer can often think of an appropriate remedy such as subsetting or transforming the variables based on the engineering realities of the process being modeled; or the engineer might ask a statistician, who might recommend other analysis techniques such as partial least squares.

ENCAPSULATION EXAMPLE 3: AUTOCORRELATION

Another problem in regression analysis of historical data from a production process, especially in heavily instrumented continuous process industries, is autocorrelation. Process control computers capture data at high frequency for the purposes of control. If such data or even hourly or daily summaries of such data are analyzed with regression analysis, statistical tests may appear to indicate that predictors belong in the model when in fact they don't. When the data sampling rate is too high, repeated measurements of essentially the same process state will be treated by regression as if they were "true repeats" (independently reproduced observations).

The engineer may not always be aware of the damaging effects of autocorrelation on the reliability of statistical tests in regression, nor may the engineer be aware of how to check for autocorrelation or what to do about it.

It is relatively easy to program checks for residual autocorrelation and generate appropriate message when it is found. One such message might explain the problem, and suggest that the engineer subset the data and take only every nth data point. Or it might recommend the use of time series analysis.

OTHER ENCAPSULATION EXAMPLES: REGRESSION

In the process modeling situation, one might not be so concerned that the distribution of residuals is strictly normal, since the statistical tests may be robust enough to perform adequately for such purposes even in the presence of moderate deviations from normality. For this reason, one might prefer instead to detect significantly skewed residual distributions and generate an appropriate message, perhaps suggesting the user look at a residual plot and possibly consider a transformation of the response variable.

Influential data and unusually large residuals can be flagged. Curvature and heteroskedasticity can be detected. Lack-of-fit can be assessed when there are repeats. Appropriate messages and suggestions can then be generated.

ASSUMPTION CHECKING: FURTHER CONSIDERATIONS

One caveat about assumption checking is that the usual statistical tests employed in checking the assumptions are themselves subject to assumptions. For this reason, in some circumstances, nonparametric and robust methods for checking assumptions might be preferred, even though they might be less powerful.

To avoid overburdening the user with false alarms, one might decide that a high degree of confidence is necessary before alerting the user to a problem, perhaps on the order of 99% or greater. But this is dependent on the situation and requires the judgement of the developers.

Messages may be effective if they appear as footnotes or subtitles on plots and listings. It is particularly effective to display a message about a potential problem on a plot that ought to show it visually. For example, if an assumption check concludes that the distribution of residuals is significantly skewed, it is very effective to display the message on a residual plot or histogram so that the user can confirm the situation visually. Messages can also be collected together in a convenient summary page or file.

ENCAPSULATION OF GOOD PRACTICE

This idea is easy to overlook. But macros can be used as vehicles for leveraging the knowledge of senior statisticians and programmers. That is, the best statistical and programming practice can be encapsulated into a macro that will be used by less experienced staff and those who are pressed for time.

DELIVERY OF NEW STATISTICAL METHODS

Macros can also be used to package new statistical methods. In this case, the macro in a sense becomes a PROC in which the knowledge of the researcher and developer is encapsulated.

DISADVANTAGES OF KNOWLEDGE ENCAPSULATION

Advantages of the different kinds of knowledge encapsulation in statistical macros have been discussed. Drawbacks may include potentially increased complexity of code (violation of the KISS principle), increased development effort, and the danger that inappropriate messages or actions could result if the macros are run in unforeseen circumstances. This last potential drawback is also of course one of the reasons for and is mitigated by appropriate bulletproofing. Finally, the wording of messages has to be done carefully in order to avoid misunderstandings.

CONCLUSION

This paper has attempted to present some ideas and techniques that might be useful in some circumstances. It would probably be incorrect to make a blanket statement for or against bulletproofing or knowledge encapsulation. It is up to the reader to decide whether or not they are appropriate in any given situation.

In the author's opinion, the goal should not be an expert system, but rather a not completely stupid macro. Considerable value can be added by small steps.

REFERENCES

Belsley, David A., Edwin Kuh and Roy E. Welsch. 1980. Regression Diagnostics. John Wiley & Sons, New York.

ACKNOWLEDGMENTS

The author would like to thank Devan Mehrotra and Lynn Wei of Merck Research Laboratories for their assistance with aspects of the analysis of covariance.

CONTACT INFORMATION

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

John K. Troxell
Merck & Co., Inc.
126 Lincoln Ave.
RY 34-A320
Rahway, NJ 07065-0900
732-594-0475
john_troxell@merck.com