

# Cover Your Assumptions with Custom %str(W)ARNING Messages

Shane Rosanbalm, Rho, Inc.

## ABSTRACT

In clinical trial work we often have to write programs based on very little (and mostly clean) data. But an experienced programmer's lizard brain is constantly warning them that their clean log is an illusion, that there is likely dirty data lurking just down the road. And sadly, the lizard brain is usually right. What's a programmer to do?

In this paper we explore using custom WARNING messages in both the data step (with PUT) and the macro language (with %PUT). This programming technique allows us to write simple programs for the data we have while at the same time protecting ourselves from the dirty data we fear.

## INTRODUCTION

Clean logs are a near mandate in the world of clinical trials programming. Log files with ERRORS, WARNINGS, uninitialized variables, etc. are strictly verboten. There are different ways to go about achieving clean logs.

- Enterprise Guide and Studio users have access to the Log Summary report that catches all ERROR and WARNING messages.
- Organizations using PC SAS® often achieve clean logs through the use of log check programs. These log check programs scour the .LOG files in a given directory and produce a summary report of every sign of trouble that is found.
- You could just read your log. ☺

But a clean log is no guarantee that the program results are correct. There are many ways for a program to produce problematic results while still producing a clean log. In this paper we will explore the technique of custom WARNING messages as a means of protecting ourselves from problematic results.

## FIRST EXAMPLE

A simple example of code that would produce a clean log but may still have problematic results is the task of converting a numeric GENDER variable to a character SEX variable.

```
data dm;
  set raw.demo;
  if gender = 1 then sex = 'M';
  else if gender = 2 then sex = 'F';
run;
```

This code only accounts for the GENDER values 1 and 2. But, other values do show up in the data from time to time: *missing*, 3, 99, etc. The question is how do we *easily* protect ourselves from such dastardly data? With custom WARNING messages, of course!

```
data dm;
  set raw.demo;
  if gender = 1 then sex = 'M';
  else if gender = 2 then sex = 'F';
  else putlog 'W' 'ARNING: unaccounted for value of ' gender=;
run;
```

With this second ELSE statement in place, if any values of GENDER other than 1 or 2 shows up in the data, this data step will generate a WARNING in the log. Something like:

```
WARNING: unaccounted for value of gender=99
```

And as long as you are checking your logs (and you should be checking your logs), the dastardly data value can be quickly identified and dealt with.

## WHAT'S WITH THE 'W' 'ARNING'?

Suppose we were to write the second ELSE as follows:

```
else putlog 'WARNING: unaccounted for value of ' gender=;
```

Writing the ELSE statement this way would guarantee that the word WARNING always appears in our log. This is not what we want. We only wish to have the word WARNING appear when there is actually something *henke* going on in our data. So, we break the word up into chunks which are only ever put back together when there is data worthy of being warned about.

## SOMETHING SIMILAR IN MACRO LAND

Because macros just write text, the following naive macro equivalent will not work.

```
%put 'W' 'ARNING: something bad happened';
```

When this line executes what will appear in the log is:

```
'W' 'ARNING: something bad happened'
```

The macro processor does not see the quote symbols (') as anything special and faithfully reproduced them in the log. So, how do we achieve the 'W' 'ARNING' effect in macro land? There are multiple solutions, but my personal favorite is:

```
%put %str(W)ARNING: something bad happened;
```

When this line executes the %str() goes away and what will appear in the log is:

```
WARNING: something bad happened
```

## KEYBOARD ABBREVIATIONS

Typing out the first few keystrokes of code to generate a custom WARNING messages can be a bit tedious (quotes, percents, and parentheses), so I encourage you to turn these code fragments into SAS abbreviations (Ctrl+Shift+A). Maybe you create the following:

- putwarn
  - putlog 'W' 'ARNING: unexpected value for ' ;
- macputwarn
  - %put %str(W)ARNING: ;
- ohbother
  - %put %str(W)ARNING: stopped in the middle of ;

Having these abbreviations at the ready will cause you to put more custom WARNING messages in your programs, making you a safer and faster programmer.

## APPLICATIONS

Nothing that I have presented thus far is groundbreaking. Papers by Mengelbier (SGF 2008) and Rosenbloom & Lafler (SGF 2013) have covered custom WARNING messages before. That said, I think the use of custom WARNING messages is an underutilized technique in need of further publicity. I am hopeful that the following collection of applications will inspire other programmers to adopt this technique.

### Unexpected Values

Suppose you are mapping numeric variable GENDER to character variable SEX. You have the numeric values 1 and 2 in the current cut of data, but you worry that other values (*missing*, 3, 99) might show up in the next data cut.

```
data sdtm.dm;
  set clinical.demonstr;
  if gender = 1 then sex = 'M';
  else if gender = 2 then sex = 'F';
  else putlog 'W' 'ARNING: unaccounted for value of ' gender=;
run;
```

**Edit Checks**

Suppose you have temperature data in degrees Fahrenheit. You might put a range check in your code to capture highly unlikely values.

```
data _null_;
  set clinical.vsmstr;
  if not (95 < temp_f < 102) then
    putlog 'W' 'ARNING: suspicious value for ' temp_f=;
run;
```

**SAS Version Check**

Maybe you've got some code that will only work in 9.4.

```
data _null_;
  if &sysver < 9.4 then
    putlog 'W' 'ARNING: minimum SAS version for this program is 9.4.';
run;
```

**Macro Parameters Non-missing**

Maybe your macro will crash if the parameter POS is missing.

```
%if %nrbquote(&pos) eq %str() %then
  %put %str(W)ARNING: POS is a required parameter;
```

**Macro Parameter Has Valid Values**

Maybe your macro will crash if an invalid value is specified for the parameters POS.

```
%if ^(%upcase(&pos) in (Y N YES NO)) %then
  %put %str(E)RROR: POS [&pos.] must be YES, NO;
```

**Stopped in the Middle of Something**

Phone calls, meeting reminders, instant messages, etc.; the steady stream of interruptions is never ending! When interruptions occur, preserve your sanity by leaving bread crumbs in your program.

```
%put %str(W)ARNING: stopped in the middle of <your text here>;
```

**Temporary Code**

Sometimes you write code that's only meant to be in effect for a short time, after which you want to remove the code.

```
data _null_;
  if today() >= '21oct2016'd then
    putlog 'W' 'ARNING: replace dummy treatment assignments with real ones.';
run;

data _null_;
  if today() > '21oct2016'd then
    putlog 'W' 'ARNING: analysis cutoff date is no longer valid.';
run;
```

**Duplicate Records Check**

Sometimes we assume that a list of BY variables is going to uniquely identify records in our dataset. Cover that assumption with a custom WARNING.

```
DATA test;
  set ts1;
  by usubjid visit;
  if not (first.visit and last.visit) then
    putlog "WARN" "ING: Multiple TS1.VISIT records for " usubjid= visit=;
RUN;
```

### Defense against Hardcoded Values

In GPLOT you frequently have to hardcode the ORDER= option on the AXISn statement. If the data range grows in a subsequent data cut you could be clipping data points inadvertently.

```
axis1 order=(20 to 40 by 5);

proc sql noprint;
  select  min(aval), max(aval)
  into    :minaval, :maxaval
  from    plotdata
  ;
quit;

data _null_;
  if &minaval < 20 or 40 < &maxaval then
    putlog 'W' 'ARNING: revisit your ORDER= option';
run;
```

### Length Protection

Assigning lengths to character variables involves making assumptions about future values. Protect yourself by checking the lengths of character variables which you know have the potential to contain long strings.

```
data sdtm.cm;
  set clinical.cmmstr;
  if length(cmindc) > 200 then
    putlog 'W' 'ARNING: CMINDC > 200 char: ' usubjid= cmindc=;
run;
```

### Empty Dataset Check

The first data cut for a clinical trial often involves an empty dataset or two. Rather than try to use the empty dataset in your program, it might be easier to set up a custom WARNING to notify you once the dataset finally gets some records in it.

```
%macro dataempty(data);

  %local dsid numobs rc;
  %let dsid = %sysfunc(open(&data.));
  %if &dsid %then %do;
    %let numobs = %sysfunc(attrn(&dsid.,nobs));
    %let rc = %sysfunc(close(&dsid.));
    %if &numobs > 0 %then
      %put %str(W)ARNING: dataset [&data] is not empty;
    %end;
  %mend dataempty;

  %dataempty(clinical.deaths);
```

### USUBJID is Not in Expected Dataset

In the following example we expect all USUBJID values to be present in dataset DM. If we find a USUBJID value that is not in DM, we throw a WARNING.

```
DATA LB_1;
  merge lab (in=inLB) sdtm.dm (in=inDM);
  by usubjid;
  if inLB;
  if not (inDM) then
    putlog 'WAR' 'NING: Subject not in SDTM.DM datasets ' usubjid=;
RUN;
```

## Records in Paired Datasets are Consistent

Sometimes there are two input datasets that are supposed to have the same number or type of records in them (i.e., if a particular USUBJID is in dataset A, that same USUBJID should also be in dataset B). This example shows how to check for this type of consistency between two datasets.

```
data work.ORQues ;
  merge work.ORQ (in=_inORQ) work.ORQUES_Raw (in=_inORQUES) ;
  by USUBJID QSSPID QSREFID ;
  inORQUES = _inORQUES ;

  if (not _inORQ) and first.QSREFID then
    putlog 'WAR' 'NING: Record in ORQUES does not match ORQ ' /
      'WAR' 'NING- ' USUBJID= QSSPID= QSREFID= / ;

  if (QUCPYNL eq 'NO') and (inORQUES) and first.QSREFID then
    putlog 'WAR' 'NING: Record with done=NO in ORQ matches records in ORQUES ' /
      'WAR' 'NING- ' USUBJID= QSSPID= QSREFID= COMPDF= QUSC= / ;

  if (QUCPYNL ne '') or (inORQUES) then output ;
run ;
```

Note the use of the slash (/) in this PUT statement. The slash acts as a hard return in the log, putting the various text and variable values on separate lines. This can help with readability.

Additionally note the use of the dash ('WAR' 'NING- ') in each PUT statement. This causes the resulting text to be indented, again helping with readability.

## Simplify Your Macro Code

Writing macros robust enough to handle **any** type of data is often quite difficult. In many cases it makes more sense to write custom WARNINGS to make you aware of cases that don't fit your assumptions about what the data *should* look like. In the following macro, which converts YYYYMMDD dates to ISO8601 format, only the easy cases are handled by the macro, with all "interesting" data resulting in a custom WARNING.

```
*----- convert character YYMMDD8 dates to ISO8601 dates -----;

%macro char2iso(dateVar);

  if &dateVar. ne '' then do ;

    %*--- write WARNING if any non-digit characters ---;
    if compress(&dateVar., '', 'kd') ne &dateVar. then
      putlog "WAR" "NING: Character date in &dateVar. contains non-digit "
        "characters " &dateVar.= ;

    %*--- full date case ---;
    else if length(&dateVar.) eq 8 then
      &outVar. = put(input(&dateVar., YYMMDD8.), IS8601DA10.) ;

    %*--- write WARNING if unexpected length ---;
    else if length(&dateVar.) not in (4, 6) then
      putlog "WAR" "NING: Character date in &dateVar. has unexpected length "
        &dateVar.= ;

    %*--- year (length 4) and year/month (length 6) cases ---;
    else &outVar. = catX('-', subStrN(&dateVar.,1,4), subStrN(&dateVar.,5,2)) ;

  end ;

%mend char2iso;
```

### Duplicate Records Check, Macro Version

The duplicate records check is common enough that you will probably want to make a macro out of it. Here is one possible implementation.

```
%macro dupcheck(data=,var=);

    %let csvar = %sysfunc(translate(&var,%str(,),%str( )));

    proc sql noprint;
        select    &csvar
        from      &data
        group by  &csvar
        having    count(*) > 1
        ;
    quit;

    %if &sqllobs > 0 %then
        %put %str(W)ARNING: obs in [&data] are not uniquely identified by [&var].;

%mend dupcheck;

%dupcheck
    (data=sdtm.ae
    ,var=usubjid aeseq
    );
```

### Well-ordered Records Check

Sometimes you will have partially-missing or ambiguous data that makes it difficult to put records in chronological order. Consider using a custom WARNING to help identify situations where sorting one way (by AESTDTC) does not give the same result as sorting another way (by AESEQ).

```
data _null_;
    set events;
    by usubjid aestdtc;
    retain lastseq;
    if first.usubjid then call missing(lastseq);
    if not (aeseq >= lastseq) then
        putlog 'W' 'ARNING: records are out of order: ' / usubjid= / aeseq= / ;
    lastseq = aeseq;
run;
```

Note the use of the / character in this put statement. This character acts as a hard return in the log, putting the various text and variable values on separate lines. This can help with readability.

### A GITHUB REPOSITORY

Prior to writing this paper I wanted to share custom WARNING messages with my coworkers. To do so I created a repository on GitHub, the free online code-sharing site. The collection of examples can be found at:

<https://github.com/srosanba/sas-custom-warnings/wiki>

The wiki content is basically the same as what is shown above, but I find it easier to read as a wiki as compared to a paper. Your mileage may vary.

### CONCLUSION

Using custom WARNING messages, especially when implemented with SAS keyboard abbreviations (or AutoHotKey hotstrings), can make you a much safer and faster programmer. Start using custom WARNING messages today!

### REFERENCES

[Simple %str\(ER\)ROR Checking in Macros](#), Magnus Mengelbier, SGF 2008

[Best Practices: PUT More Errors and Warnings in My Log, Please!](#), Mary F. O. Rosenbloom and Kirk Paul Lafler, SFG 2013

## ACKNOWLEDGEMENTS

I used crowd-sourcing to obtain the majority of the applications presented in this paper. Thanks to the following RhoSUG members for suggesting applications of custom WARNING messages for use in this paper:

- Brian Armstrong
- Spencer Childress
- Frank Dilorio
- John Ingersoll
- Alice Miller
- Amy Moseby
- Paul Nguyen
- Adela Piña
- Dave Scocca

## CONTACT INFORMATION

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

|                  |                              |
|------------------|------------------------------|
| Name:            | Shane Rosanbalm              |
| Enterprise:      | Rho, Inc.                    |
| Address:         | 6330 Quadrangle Drive        |
| City, State ZIP: | Chapel Hill, NC, 27517       |
| E-mail:          | shane_rosanbalm@rhoworld.com |

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.