Paper 352-2013

On a First-Name Basis with SAS®: Creating Personalized Error Messages Using SAS 9.2

Andrew Clapson, Statistics Canada; Valerie Hastings, Statistics Canada

ABSTRACT

In the interests of creating a user-friendly SAS system, it's often a good idea to include code that checks for common errors, notifies the user and suggests possible solutions. Apart from simply delivering this information to the user, the use of customized message windows is also a good opportunity to express congratulations upon a successful run or even deliver a light-hearted finger wagging in the case of unexpected errors.

Using SAS 9.2, this paper will detail the steps necessary to include basic error messaging functionality in your SAS programs, covering both error notification and confirmation of successful program execution. In addition, through the use of system macro variables these feedback messages can surprise the user by 'knowing' their name and addressing them directly.

INTRODUCTION

In this paper we will explain how to implement basic error messaging functionality simply by including a SAS macro upon execution of your SAS program(s). While there are an endless variety of error conditions that one can set, we use the automatic &SYSCC macro variable for our example, and trigger a positive or negative message window accordingly.

Only functions and commands from Base SAS are required; our code uses the CALL SYMPUT routine and the %WINDOW statement. The user may wish to %INCLUDE this macro during or after program execution.

ASSOCIATING NAMES WITH USER IDS

In order to get up close and personal with your users, you'll first need to know their names. This first step is unfortunately a somewhat manual one; because we're going to use the automatic macro variable &SYSUSER, we need to establish a reference set of all user IDs and their respective first/last name mappings. In large organizations, this can usually be obtained from your system admin team. Alternatively, if only a few individuals will be using the system, you may know their names offhand and can include them in a workbook or dataset.

For this example, our users are some rather well-known characters:

```
data UserList;
input userID $ firstName $ lastName $;
datalines;
picajea Jean-Luc Picard
rikewil William Riker
troidea Deanna Troi
crusbev Beverly Crusher
;
run;
```

This above is quite obviously only an example. If you are dealing with a large user base, a more useful method would be to maintain an external mapping file and simply update it as needed. However, regardless of your chosen record-keeping method, you will now have a user ID variable that can be compared to the &SYSUSER macro variable and linked with a first and/or last name.

DELIVERING CONGRATULATIONS…AND APOLOGIES

Next, we need to define the messages that we want to include. In our example, we only distinguish between two broad conditions: no errors found vs. at least one error found, both of which refer to the state of the &SYSCC automatic macro variable. As well, for the sake of brevity we will include only a few different positive and negative messages.

Regarding message content, how 'snarky' you feel like being with the negative messages is of course up to you, but it's worth keeping in mind that a user may be frustrated if they receive an especially obnoxious error message, as entertaining as it may seem to the programmer.

```
data PosMessages;
infile datalines truncover;
input posMessage $100.;
datalines;
Great work, &firstName.! Everything went fine.
I detected no errors in the last program, &firstName.. Well done!
Nope...no errors detected in this run, &firstName..
;
run;

data NegMessages;
infile datalines truncover;
input negMessage $100.;
datalines;
I detected some errors, &firstName. - be sure to verify your results.
Sorry to report, &firstName., but something went wrong there.
I felt a light dusting of errors there, &firstName....
;
run;
```

Note the use of the macro variable &firstName in the message datasets; this variable will be defined (and resolved) later in the program. Also, you can include as many different messages in either category as you want – we will treat each message type separately, and select a message from the total number available in each category.

DELIVERING THE NEWS

Now that we have defined our user names and the range of messages we want to display for each case, we need to define a macro that will do the following:

1. Read in the list of names, choose our user out of them and associate their user ID with their first name.
2. From the various available messages, randomly select a negative and positive message to display for the user, depending on the presence of any errors.
3. And then, depending on the state of the &SYSCC macro variable after program execution, trigger either the positive or the negative window. Every time the user runs the program they will have a chance of receiving a different message.
4. And finally, if errors are found, the user can then search the log and start debugging the program…or get in contact with the developer or system team, as the case may be.

Now for the body of the program - note that this macro assumes that you have already defined the required user names and positive/negative message datasets, as in the code above.

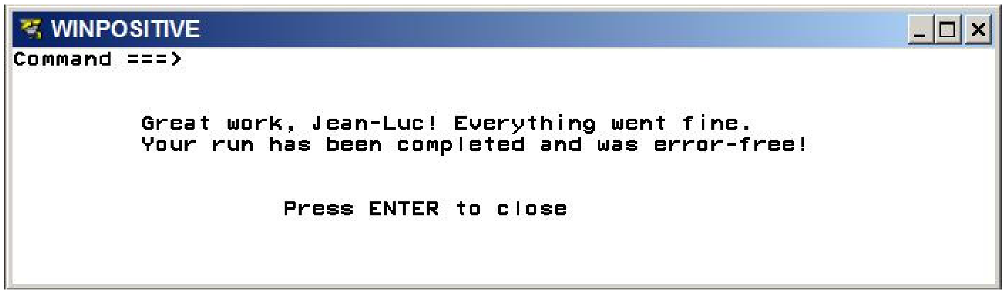
```
%macro personalMessage;
data _null_;
set PosMessages end = eof;
       if eof then do;
              call symputx("nObsPos", _n_);
       end;
run;

data _null_;
set NegMessages end = eof;
       if eof then do;
              call symputx("nObsNeg", _n_);
       end;
run;
```

```
data _null_;
set UserList end = eof;
       if (upcase(userid) = upcase("&sysuserid.")) then call symputx('firstName',
       firstName);
       if eof then do;
          if (symget('firstName') = "") then call symputx('firstName', 'my friend');
       end; /* This sets a default name in case the userID is unknown */
run;

%let randPos = %sysfunc(ceil(&nObsPos.*%sysfunc(ranuni(0))));
data _null_;
set PosMessages;
       if (_n_ = &randPos.) then do;
              call symput('posMessage', posMessage);
       end;
run;

%let randNeg = %sysfunc(ceil(&nObsNeg.*%sysfunc(ranuni(0))));
data _null_;
set NegMessages;
       if (_n_ = &randNeg.) then do;
              call symput('negMessage', negMessage);
       end;
run;

%WINDOW winNegative columns = 80 rows = 15
       #3 @10 "&negMessage."
       #4 @10 "Warning: There was at least one error in your run."
       #7 @20 "Press ENTER to close";
%WINDOW winPositive columns = 70 rows = 15
       #3 @10 "&posMessage."
       #4 @10 "Your run has been completed and was error-free!"
       #7 @20 "Press ENTER to close";

%if (&SYSCC. > 4) %then %do;
   %DISPLAY winNegative;
%end;
%else %do;
   %DISPLAY winPositive;
%end;
%mend personalMessage;
%personalMessage;
```

The first two _null_ datasteps simply count the number of positive and negative statements available and define a macro variable for each amount. The third datastep reads the userID and defines a macro variable holding the user's first name (note that in case of an unknown user, a default case is defined). The last two _null_ datasteps randomly choose an observation from the positive and negative message datasets, respectively. Finally, the two windows (positive/negative) are defined, each using the respective personal message that was selected in the previous steps, as well as some other more standard text.

If you're not familiar with the %WINDOW statement, you simply define the window dimensions and then the messages to display at different coordinates in the window. It's a good idea to include a prompt for the user to close the window: a user can close the window either by clicking on the 'X' in the upper right-hand corner or by pressing the enter key.

In the above example, a &SYSCC value greater than four means there was at least one error in the program, and the negative window will be triggered. A &SYSCC value of less than four means there were no errors detected (though there may still have been warnings) and the positive window will be displayed.



```
Command ===>

        Great work, Jean-Luc! Everything went fine.
        Your run has been completed and was error-free!

                  Press ENTER to close
```

Display 1. An example pop-up window presented to a user

CONCLUSION

With the inclusion of a simple, flexible macro, you can incorporate message windows into your SAS programs that can be expanded to handle a variety of cases. It can be adapted to suit your needs and can display countless different messages based on different conditions. Adding usernames to the windows can be a fun way to get personal with the user while conveying important information at the same time.

REFERENCES

McLean, Greg. 2006. "Grid Computing with SAS – A Developer's Perspective." *SAS Global 2006 Conference*. Ottawa, Ontario: SAS SGF

Mann, Alan. 2004. "%WINDOW: SAS' Diamond In the Rough". Martinsburg, WV. http://analytics.ncsu.edu/sesug/2004/CC14-Mann.pdf

ACKNOWLEDGMENTS

The authors would like to thank the staff of the Statistics Canada SAS Technology Centre, as well as Art Tabachneck, all of whom reviewed this paper and contributed helpful suggestions.

CONTACT INFORMATION

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

Andrew Clapson
Statistics Canada
andrew.clapson@statcan.gc.ca

Valerie Hastings
Statistics Canada
valerie.hastings@statcan.gc.ca