Paper 3106-2015

The Best DATA Step Debugging Feature

Steven First, Systems Seminar Consultants, Inc.

ABSTRACT

Many languages including the SAS data step have extensive debuggers that can be used to detect logic errors in programs. Another easy way to detect logic errors is to simply display messages and variable content at strategic places. The PUTLOG statement will be discussed and examples given that make this probably the easiest and most flexible way to get your data step logic correct.

INTRODUCTION

When your SAS program logic is not doing what you want, there are a couple of ways to trace what is happening. If you are running interactively, the SAS interactive data step debugger is a very comprehensive and somewhat involved method to step through your program. Another much simpler way is to display selected values at strategic spots in your program.

PUTLOG DEBUGGING EXAMPLE

Taking the following example that has State, County, and City data. We want to select rows for Middleton, in Dane county, Wisconsin, but the program below selects no data, why?

```
/*******************************************************/
/* select Middleton, Dane County, Wisconsin            */
/*******************************************************/
data census;                          /* build sas ds       */
 input state $ county $ city $;       /* read row of data   */
 if state='WI';                       /* Filter WI rows     */
 if county='DANE';                    /* Filter DANE county */
 if city='MIDDLETON';                 /* Filter City        */
datalines;                            /* inline data        */
WI DANE     MONONA
WI WAUKESHA WAUKESHA
WI DANE     MIDDLETON
MN HENNEPIN PLYMOUTH
;
RUN;

Partial Log:
NOTE: The data set WORK.CENSUS has 0 observations and 3 variables.
```

The SAS PUTLOG statement can display constant and variable values in the SAS log as the data step executes. By using PUTLOG before or after decision points in your data step, you can see the values as they go by.

The following example uses PUTLOG after the INPUT statement to insure that the values are read correctly, another after each subsetting IF to display what the values are if the program passes through each statement. The PUTLOG first displays an ‘eyecatcher‘ constant along with the special variable _N_ and selected variables. It could have also just displayed the _ALL_ variable to see every value in the step.

```
/*******************************************************/
/* select Middleton, Dane County, Wisconsin            */
/*******************************************************/
data census;                          /* build sas ds       */
 input state $ county $ city $;       /* read row of data   */
 putlog "**** after INPUT         "  _n_= state= county= city=;
```

```
 if state='WI';                    /* Filter WI rows     */
 putlog "**** after STATE check   "  _n_= state= county= city=;
 if county='DANE';                 /* Filter DANE county */
 putlog "**** after COUNTY  check "  _n_= state= county= city=;
 if city='MIDDLETON';              /* Filter City        */
 putlog "**** after all checks    "  _n_= state= county= city=;
datalines;                         /* inline data        */
WI DANE     MONONA
WI WAUKESHA WAUKESHA
WI DANE     MIDDLETON
MN HENNEPIN PLYMOUTH
;
RUN;

159  data census;
160   input state $ county $ city $;
161    putlog "**** after INPUT         "  _n_= state= county= city=;
162   if state='WI';
163    putlog "**** after STATE check   "  _n_= state= county= city=;
164   if county='DANE';
165    putlog "**** after COUNTY  check "  _n_= state= county= city=;
166   if city='MIDDLETON';
167    putlog "**** after all checks    "  _n_= state= county= city=;
168  datalines;
**** after INPUT         _N_=1 state=WI county=DANE city=MONONA
**** after STATE check   _N_=1 state=WI county=DANE city=MONONA
**** after COUNTY  check _N_=1 state=WI county=DANE city=MONONA
**** after INPUT         _N_=2 state=WI county=WAUKESHA city=WAUKESHA
**** after STATE check   _N_=2 state=WI county=WAUKESHA city=WAUKESHA
**** after INPUT         _N_=3 state=WI county=DANE city=MIDDLETO
**** after STATE check   _N_=3 state=WI county=DANE city=MIDDLETO
**** after COUNTY  check _N_=3 state=WI county=DANE city=MIDDLETO
**** after INPUT         _N_=4 state=MN county=HENNEPIN city=PLYMOUTH
NOTE: The data set WORK.CENSUS has 0 observations and 3 variables.
```

WHAT IS THE PROBLEM?

On the third iteration, we see that the INPUT statement never read the city value correctly and its value ('MIDDLETO') was truncated. We also see that the program passed the State check, and the County check, and it did not pass the City check (no display), because of the truncated value. Therefore we have to alter the INPUT statement to read the City correctly.

If the file is very large you could get lots of displaying in the log. Because you can alter the logic in the data step, you might put a record counter check to not display all records. You can also stop the data step early and _N_ is a convenient counter that you can easily interrogate.

```
/*************************************************************/
/* select Middleton, Dane County, Wisconsin                  */
/*************************************************************/
data census;                       /* build sas ds           */
 input state $ county $ city $;    /* read row of data       */
if  _N_  > 50 then stop;           /* limit displayed records */
if    3   le _N_
&  _n_ lt 4 ;
 putlog "**** after INPUT         "  _n_= state= county= city=;
 if state='WI';                    /* Filter WI rows         */
 putlog "**** after STATE check   "  _n_= state= county= city=;
 if county='DANE';                 /* Filter DANE county     */
```

```
 putlog "**** after COUNTY  check "  _n_= state= county= city=;
 if city='MIDDLETON';              /* Filter City             */
 putlog "**** after all checks    "  _n_= state= county= city=;
datalines;                         /* inline data             */
 WI DANE     MONONA
 WI WAUKESHA WAUKESHA
 WI DANE     MIDDLETON
 MN HENNEPIN PLYMOUTH
;
RUN;
```

CONCLUSION

In conclusion, PUTLOG is an easily remembered and very flexible way to display your data and debug your program.

CONTACT INFORMATION

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

Steven First
Systems Seminar Consultants, Inc.
608-278-9964
sfirst@sys-seminar.com
www.sys-seminar.com