Paper 68

The SAS® Debugging Primer

Frank C. DiIorio
Advanced Integrated Manufacturing Solutions, Co.
Chapel Hill NC

We've all been there. Indeed, sometimes it seems we live there. We take a functioning program, make some changes, rerun it and … are thrust into the Land of Bad Code. What happened? Why did the seemingly minor change wreak such havoc? Most important – how do you fix the program?

This paper discusses a variety of SAS debugging techniques. It

- addresses background, behavioral issues related to program development
- identifies some common errors and possible remedies
- describes in necessarily broad terms the tools SAS provides to make debugging a relatively painless process.

The text and examples are limited to Base SAS. Consider, however, that even though other products such as SAS.AF and SAS/IML have their own programming environments and syntax, the program process is in many ways identical. Much of what is discussed here can be fruitfully applied elsewhere in the SAS System.

Finally, the paper implicitly emphasizes the importance of good coding style. Indeed, if the code is written well there will be much less opportunity for using debugging techniques. Write clearly, avoid tricks, consider all possible logic paths, and you will see most of your programming time being dedicated to development and not debugging. For an excellent book dealing with programming style see Steve McConnell's *Code Complete*.

BEHAVIORAL ISSUES (YOU)

Your attitude is as important as the tools and techniques you will be using when fixing a program. It's worthwhile to remember Elisabeth Kubler-Ross' Stages of Grief:

Denial	"This can't be happening. It ran just fine yesterday."
Anger	"I'm a useless $%*&@ idiot!" or, deflecting the blame, "Stupid compiler!"
Bargaining	"If I can just get the report out the door for FedEx I'll rewrite the program tonight and make it clearer."
Depression	"This is too complicated – why didn't I let [competitive co-worker] take this when I had the chance. I'm doomed, I'm doomed, I'm doomed!"
Acceptance	"OK, it's broken, but that doesn't mean I can't fix it."

Let's assume that phases one through four have already been dealt with and that we are starting with acceptance. Here are some behavioral tips to consider. Some of them are SAS-related, others are not. They are not necessarily presented in the order in which they should be used.

DEVELOP (AND FIX) INCREMENTALLY

Even simple programs are best developed in small steps. This is especially true when you are writing logically complex or long DATA steps or when you are using new PROCs.

The reason is simple. Assume you need to make some modifications to a program and that these require creation of a new variable and modification of a regression model. After making the changes and rerunning the program, the regression output makes no sense: the number of observations has dropped drastically and the estimates are biased.

Is the problem in the DATA step or the specification of the regression model? You probably can't tell; the results are confounded. The better approach (and one that will make you feel more comfortable about what you are producing) is to attack the problem in two pieces. First, modify the DATA step and follow it with a frequency distribution, print, or other method that will convince you the new variable takes on the values you expect.

Next, comment out the first step's diagnostics or add a CANCEL option to the RUN statement and modify the regression code. *To do two things at once is to do neither (Publilius Syrus "Moral Sayings").*

DON'T PANIC

Many errors can be propagated by a single, simple mistake early in the program. Likewise, a badly-coded macro can spew hundreds of lines of hard to read error messages. Bear in mind that a single, simple "tweak" often fixes dozens of downstream problems. Consider the following program:

```
/* Written by Joe Novice, June 3 1998
   Summarize current month's values and
   append to year-to-date
%let curr=jun;

data this_mth;
/* moved to c: in jan '98 */
infile " c:\fin\curr\&curr..dat" ;
retain month "&curr.";
input acct 1-6 suffix 7-9 $3. tranno 5.
      trantype $2. amt 8.2;
run;

proc print data=this_mth(obs=100);
title "first 100 obs. from &curr.";
id branch;
run;
```

There is a single, simple mistake here - the omission of the closing comment (*/) in the third line. The header comment is not terminated until the comment after the DATA statement. This means the macro variable CURR will not be defined and the DATA statement will not be executed (both are contained within the unexpectedly long comment). SAS rightfully gets hysterical, as seen in the Log:

```
1    /* Written by Joe Novice, June 3 1998
2       Summarize current month's values and
3       append to year-to-date
WARNING: Apparent symbolic reference CURR not
         resolved.
4    %let curr=jun;
5
6    data this_mth;
7    /* moved to c: in jan '98 */
8    infile "c:\fin\curr\&curr..dat";
     ------
     180

ERROR 180-322: Statement is not valid or it
      is used out of proper order.

WARNING: Apparent symbolic reference CURR not
         resolved.
9    retain month "&curr.";
     ------
     180

ERROR 180-322: Statement is not valid or it
      is used out of proper order.

10   input acct 1-6 suffix 7-9 $3. tranno 5.
     -----
     180
11         trantype $2. amt 8.2;
```

```
ERROR 180-322: Statement is not valid or it
      is used out of proper order.

12   run;
13
14   proc print data=this_mth(obs=100);
ERROR: File WORK.THIS_MTH.DATA does not
       exist.
15   title "first 100 obs. from &curr.";
WARNING: Apparent symbolic reference CURR not
         resolved.
16   id branch;
17   run;

NOTE: The SAS System stopped processing this
      step because of errors.
NOTE: The PROCEDURE PRINT used 0.11 seconds.
```

All the errors were generated for want of a "*/". And they were quickly resolved with the insertion of a "*/". Sometimes it isn't as bad as it seems.

BE ADAPTABLE – CONSIDER ALTERNATIVES

And sometimes it *is* as bad as it seems. Bear in mind that one of SAS's strengths is that it is a robust programming environment. If you program yourself into a corner using a familiar set of tools, chances are that you can save the day by learning new tools and techniques (e.g., use IML or SQL for dataset-level operations when traditional DATA step coding becomes too ponderous. Data-driven formats can replace resource-hungry MERGE or UPDATE steps).

A caveat: when switching tools, be sure you are still solving the original problem. Ask yourself: does the dataset I just created with SQL have the correct unit of observation and variables? Is it identical to the one I was (unsuccessfully) trying to create with a DATA step?

Deliberately becoming a bit naïve about the tools' functionality opens you to the possibility of a completely new solution to the problem, *even if the problem "feels" familiar*. Everything – even the somewhat bizarre – should be fair game. *Unhappy the general who comes on the field of battle with a system. Napoleon I*.

LEARN, RE-LEARN AND RE-VISIT

Regardless of your level of expertise, don't assume you know everything about a PROC or DATA step feature. The SAS System is constantly evolving. New features, many of which were requested by end-users, invariably simplify the life of the programmer. You won't learn about these features unless you review the literature.

Also, people tend to learn a feature for a particular application and then not revisit it later on. Even if it *doesn't* change over time, you'd be missing out on functionality. For example, using PROC REPORT simply for its paneling capability may blind you over time to the PROC's ability to do summarization within groups, create temporary variables, and create datasets. The "Resources" section, below, identifies numerous on-line, print, and human resources.

PUNT (TEMPORARILY), OR AT LEAST "DIAL IT DOWN"

If you are stuck and you have the luxury of time, use it to your advantage. Leave the problem program and work on something else for even a short while. It's remarkable how often even experienced programmers get "wrapped around the axle," getting stuck in one problem attack path. Coming back to the program with a fresh attitude clears the debris from the old path and makes new, alternate paths more appealing.

In a similar vein, if you don't have the luxury of temporarily punting, at least run the ball a bit more deliberately. The code-compile-run-assess output cycle of interactive development environments encourages rapid resubmission of programs. Very often, this simply means an at-loose-ends programmer tries one solution after another without really examining the underlying problem. Slow down, examine the Log and output files, and spend some time away from the keyboard.

LEARN FROM THE EXPERIENCE

Once you debug the program and fix the problem (and you will!) reflect on what the program looks like and how you got there. How did the problem-solving process unfold? Did you rely on experience, intuition, documentation, or other resources? Which were most effective? What "red herrings" or false leads did you encounter? If you have time, identify other programs with similar coding/data/stylistic issues.

Make mental notes regarding approach and comments in the program about why the code looks the way it does. Even if you think you'll never forget the road to the solution, you probably will. Documentation will save you a good deal of head scratching later on. Remember, *problems are opportunities in work clothes (Henry Kaiser)*

BEHAVIORAL ISSUES (SAS)

SAS has some characteristics that directly affect both your reaction to an failed program and how you proceed to fix the program. Consider the following:

IT CAN OVERREACT

As we saw in the "Don't Panic" section, above, one error can spawn many.

IT CAN GET SQUIRRELLY

Sometimes, especially when you have had a long-running interactive SAS session, the system can begin to behave erratically. Performance will degrade, widgets will become invisible, and other, transient problems will develop even though you are making small and legitimate changes to your program. In these situations it is usually best to save your work, exit and restart SAS. The behavior is sometimes due to memory fragmentation. Regardless of the source, it is beyond your control other than to exit and reenter.

IT *USUALLY* TELLS YOU WHAT'S WRONG

Sometimes it will even fix the problem for you. But don't rely on its corrections – if you entered NOCOLUMN in a PROC FREQ TABLES statement, SAS will say it assumed you meant NOCOL and continue executing. Fix the spelling! Warnings and notes about assumptions are distracting, and you can't be sure that the correction will be made in future releases of the SAS System.

SAS *CANNOT* IDENTIFY LOGIC ERRORS

If you tell SAS to do the wrong thing in a syntactically valid way, it is up to you to detect the error and make the changes. For example, if you read an identification variable with a $5. format instead of a $6. and there are at least 6 bytes on each record SAS will *not* bring the inadvertently truncated id to your attention.

Continuing the previous example - SAS doesn't always know what's wrong with your data (that is, it can detect a bad data type but it doesn't know if coding should have been 0, 1 instead of 1, 2 for male/female). Consider another example: you read a 10-record dataset but are missing a semicolon at the end of the INPUT statement. The next statement, LIST, was taken as part of the INPUT statement and LIST was read as a variable name.

```
data test;
infile 'test.dat' obs=10;
input a b c
list;
```

You have 5 records instead of 10 in the dataset. The first field of record two was needed to assign a value to variable LIST. Two input records rather than one were needed for each observation. A variation of this scenario occurs when you get seemingly correct output but the Log indicates something was amiss. Be sure to reconcile the Log and output before declaring a program "correct".

TOOLS

There's no shortage of programming statements and options to aid the debugging process. In this section, we review system options, the SAS Log, macro variables, macros, DATA step statements, the DATA step debugger, procedure-related options, and dictionary tables and views. Given the breadth of the coverage, it is

necessarily cursory.

OPTIONS

System options DETAILS, DKRICOND, DKROCOND, DSNFERR, ECHOAUTO, ERRORABEND, ERRORCHECK, ERRORS, FMTERR, INITSTMT, MSGLEVEL, NOTES, REPLACE, RSASUSER, S, S2, SOURCE, SOURCE2, SYNTAXCHECK, VNFERR, WORK These options control the amount of diagnostic output you receive, and the source statements that are displayed in the Log. They also tell the SAS Supervisor how to react to abnormal conditions – that is, whether a particular event (such as an unknown format) should be considered an error.

Macro-related options CMDMAC, IMPLMAC, MACRO, MAUTOSOURCE, MERROR, MLOGIC, MPRINT, MRECALL, MSTORED, RESERVEDB1, SERROR, SYMBOLGEN. These options allow extended searches for macro references and control the amount of diagnostic information related to macro variables and macro references.

THE SAS LOG

The Log performs many useful activities. Don't assume that your program ran correctly if it produced "reasonable" looking output. Examine the Log for errors, warnings and notes. Here are some of its many useful features. The Log:

- **Echoes your code.** Be sure all the code you *think* should be executed actually *was* submitted for execution.
- **Notes character-numeric conversions.** If you must do this, at least be sure it's being done in the places you anticipated.
- **Identifies creation of missing values.** The Log contains the location (line, column number) of each calculation that resulted in a missing value. Some of these may be acceptable, but watch for an excessive number of locations or a location that generates as many missing values as there are observations in the dataset. The latter is a warning that there may be something systematically wrong with the calculation.
- **Identifies uninitialized variables.** This may be due to: a misspelled variable name; incomplete removal of a group of statements (thus "orphaning" a variable reference); omission of a semicolon, thus identifying a SAS keyword as a variable name (see "SAS *cannot* identify logic errors," above).
- **Describes the size of output SAS datasets.** The datasets should meet your a *priori* expectations about number of observations and number of variables.
- **Describes the location and size (number of lines) of external data sources.** As with output SAS datasets, above, you should have some idea of how large the data are.
- **Contains text/diagnostics** written by PUT and %PUT statements.
- **Contains macro-related** source and symbol diagnostics.

MACRO VARIABLES

Macro variables can control the amount of diagnostic output. They can be used for substitution in RUN statements (the CANCEL option) and can toggle execution of PUT and other statements in DATA steps. Refer to the following example.

```
%let rundiag = ;  /* blank=run prints,
                     cancel=don't run prints */
%let runput = *;  /* *=comment out PUT stmts,
                     blank=execute PUT stmts */

<<< lots of intervening code >>>

proc print data=dataset1;
title "Print of non-critical failures";
run &rundiag.;

data test;
set master.round3;
&runput. put "This is input value of var."
             "  DIAG " diag;
calc = diag * (1 - ratio);
&runput. if calc = . then
            put "CALC was missing. "
                diag= +3 ratio=;
```

Automatic macro variables. The %PUT statement supports an automatic variable, _ALL_. This dumps all current macro variable names and values to the Log. The scope of the displayed variables may be limited by adding the GLOBAL, LOCAL, or AUTOMATIC option to the statement. For example, %PUT _ALL_ GLOBAL displays only globally available user-defined macro variables.

MACROS

You can keep debugging code in production programs without consuming too many additional (i.e., debugging) resources or cluttering the visual look and feel of the program.

```
* Before ;
Data temp;
Set inut.master;
<code, code, code!>
run;

proc print data=temp(where=(id < 100));
title "ID's less than 100";
run;

proc freq data=temp;
tables race salgroup / missing;
title "From TEMP";
run;


* After ;
%macro runit(debug=y);
%let debug = %upcase(&debug.);
Data temp;
Set inut.master;
<code, code, code!>
run;

%if &debug. = Y %then %do;
    proc print data=temp(where=(id <
100));
    title "ID's less than 100";
    run;

    proc freq data=temp;
    tables race salgroup / missing;
    title "From TEMP";
    run;
    %end;
%mend;
%runit(debug=n);
```

DATA STEP STATEMENTS

The DATA step is, arguably, where the most things can go wrong. Fortunately, you have a host of statements and features to help you:

- **PUT statement.** Use PUT statements to display problematic variables, signal that a portion of code is executing, and any other significant DATA step event. The display should be meaningful! Take a small amount of time to annotate the output. Rather than
  ```
  put salary;
  ```
  enter
  ```
  put 'Salary has changed!' salary comma8.2;
  ```
- **_INFILE_ and _ALL_ automatic variables. These** display the last record accessed by an INPUT statement and all variables currently in the Program Data Vector (PDV).
- **LIST statement.** Writes to the Log the last record accessed by an INPUT statement. Unlike the _INFILE_ automatic

variable, it provides a column ruler. This helps identify errant INPUT statement column specifications.

- **_ERROR_ automatic variable.** This automatic variable toggles a dump of all variables on (1) or off (0). Assign it a value of 0 if you want to suppress the dump associated with such conditions as zero divides, invalid INPUT values, and the like.
- **IN dataset option.** Use it to identify which dataset(s) a particular observation came from during a MERGE, SET, or UPDATE operation.
- **? and ?? format modifiers.** Use these to suppress the Log messages generated when SAS encounters invalid raw data. Coding input var 4. ?; will suppress the NOTE about invalid data and only print a dump of the offending line. Coding input var 4. ??; will suppress both the NOTE and the dump. If you know a particular variable is problematic and don't want the dumps and NOTEs cluttering the Log, use these options.

DATA STEP DEBUGGER

Version 6.11 and higher of the SAS System support the DATA step debugger. This is an interactive tool used in the Display Manager Environment. It lets you step through a program line by line, examine or change variable values, and other activities that give you insight into DATA step execution. It is a programming environment unto itself and is beyond the scope of this paper. Refer to SAS Institute's "SAS Software: Changes and Enhancements, Release 6.11" for a complete description of the debugger's syntax and usage.

PROCEDURES

DATA step debugging does not have to take place entirely within the confines of the DATA step. Several procedures can be used to help you get a sense of what's happening in the data. Here are a few:

- **FREQ** Use it to get n-way distributions of categorical variables or for FORMATted values of interval, ratio-scale variables. The distributions can verify that, say, the categories of a value calculated in a DATA step are, in fact, in the expected range or take on legitimate values.
- **CHART** It serves somewhat the same purpose as FREQ, but with a more visually oriented display.
- **PRINT** Use PRINT to display all or part (rows/columns) of the dataset. Use the WHERE statement or dataset option to control what is printed. You could, for example, use WHERE to display observations with a variable value outside a specific range.
- **REPORT** This procedure serves roughly the same purpose as FREQ, but is more tree-friendly, since it can easily "panel" across the page (the PANELS=n option in the PROC statement). Consider using it if you are printing hundreds of observations but only a few, narrowly formatted variables.
- **CONTENTS** ensure that the name, order, data type and other characteristics of the data are what you expect.

DICTIONARY VIEWS

These are SQL tables and associated views that are automatically generated by SAS during system startup and are maintained throughout the session by the SAS Supervisor. They contain much of the information found in the CONTENTS and CATALOG output datasets, as well as non-graphics option settings and macro variables. The tables can be accessed through the SQL procedure. The views can be accessed through any SAS procedure that is able to process views in a read-only manner (i.e., don't use PROC DELETE). For a more thorough discussion of Dictionary Tables and Views, see Frank DiIorio (SUGI Proceedings 1996), SAS Institute Tech Report P-222.

To get a feel for their contents, execute SAS interactively, enter DIR SASHELP on the Command line, restrict the TYPE to VIEW. You should see 16 views beginning with the letter "V" (VCOLUMN, etc.). Enter a B next to these to view the contents, or S to list the variables in the view.

As an example of the tables and views use in debugging, consider this scenario. You are merging several datasets, each of which contains a character variable DGRP. A print of the variable shows that it is truncated in some observations. Examine the variable length in the dictionary tables and views by doing one of the following:

```
proc sql;
select memname, name, length
from dictionary.columns
where name = 'DGRP';
quit;

from the command line:
fsv sashelp.vcolumn
where name = 'DGRP'
show id memname
=length

proc print data=sashelp.vcolumn;
where name = 'DGRP';
run;
```

The tables will reveal which datasets have the variable in question. The dataset with the offending, short length DGRP can be readily identified.

COMMENTS

While not a debugging tool *per se*, comments are tightly woven into the fabric of a good, maintainable program. They explain the program's purpose, describe the typical inputs and outputs, identify portions of the program that may be unusual (non-standard methods of handling a calculation, etc.). In the context of fixing a program, they can describe alternative approaches, those that were tried and did not work. They can be, in effect, an audit trail.

RESOURCES

GURUS

Unless you are working in a vacuum, it's almost inevitable that you will encounter someone who seems to know everything about SAS and can communicate this knowledge in a clear and non-patronizing manner. It is very likely that the way they achieved this near-mystical status was by making, and recovering from, every imaginable error. They made beginner's mistakes, intermediate mistakes, and are very likely now at the level of incredibly esoteric mistakes. But … the key point is that they learned from their mistakes and were able to plunge ahead into even greater levels of program complexity and scope. They are the rulers of Arcania and should be sought out for their wisdom, *but not as a first resort*.

MANUALS

Online help and hard-copy manuals and books. SAS Institute's publications are the definitive reference for syntax. Some Institute publications and most of their Books By Users series approach programming from more real-world standpoints - light on syntax, perhaps, but rich on practical "how to" in a wide variety of disciplines.

PROCEEDINGS

There are numerous annual SAS User Group conferences. Each of these produces a Proceedings, which is collection of presentations at the conference. Some groups produce a CD-ROM in addition to the more traditional bound volumes. Conference Proceedings are a good blend of tutorial and "hands on" information. If you are looking for an alternate explanation of how to define a column percentage in PROC TABULATE or want to see if someone has already solved your immediate problem (possibly by a different approach), this is a good place to look. Proceedings are included with the cost of registration or may be ordered directly from SAS Institute or a regional users group. See the Institute's Web Site (next topic) for ordering information.

THE WEB

SAS Institute's World Wide Web site contains marketing information, office locations, training schedules, downloads for bug fixes and product upgrades, and much more. The Uniform Resource locator (URL) is `www.sas.com`. Other, non-Institute

sites are growing in popularity and user acceptance. Two such sites are SAS for the Masses (http://faith.hypno.net/sasmass/) and Qualex Consulting Services (http://www.qlx.com). New sites and services are regularly announced on the SAS-L list server (next item).

SAS-L

This is a list server with nearly 2,000 subscribers. It is the definitive on-line resource, and it is free. Post questions to `sas-l@uga.cc.uga.edu`. Make sure the subject field gives a clear idea of the nature of your problem. In the body of the message, state the problem as succinctly as possible, describe solutions you've already tried, and post any appropriate Log or output. To subscribe to SAS-L, send an email to

```
listserv@uga.cc.uga.edu
```

No subject is needed. The body of the message should be

```
sub sas-l Your Name
```

No quotes are needed around your name. Just be sure to separate the sub, sas-l, and name fields with at least one blank. You will receive a confirmation message containing instructions on using other useful list server commands. SAS-L sends 25 messages in a typical weekday. To receive all the messages in one long email use the SET DIGEST command once you've subscribed. SET is fully described in the confirmation email mentioned earlier.

ONLINE HELP: THE SAS SAMPLE LIBRARY

The SAS System comes with useful sample programs. The samples are for Base SAS procedures (including SQL) and for any other installed products, such as SAS/GRAPH, and SAS/FSP. To access the sample library in Windows 6.12, start Display Manager and then click on Help, Sample Library. The process is basically the same in other releases and platforms.

OTHER ONLINE FACILITIES

In addition to the Sample Library, clicking on Help in any Display Manager session gives you access to a wealth of online documentation. Explore this drop-down menu for example-driven, operating system, bug-related, new feature, and other documents.

SAS INSTITUTE TECHNICAL SUPPORT

SAS Institute's Technical Support staff may be reached at (919) 677-8008. Yes, it's a toll call, but the support is free. Be prepared to provide your site number to the support staff, and have on hand as many Logs, output listings, and notes as possible. If your problem cannot be handled by the person initially handling your call you will be issued a "tracking number" and the problem will be referred to senior staff. Most problems requiring tracking numbers are resolved within 24 hours.

IN-HOUSE TECHNICAL SUPPORT

Most companies and universities have Help Desks or similarly-named locations that can be called or visited for help. The advantage to using these rather than SAS Institute technical support is that they have site-specific knowledge about network addressing, job submission and retrieval, and other issues not readily apparent to the more product-oriented people at the Institute.

YOU!

In the long run, you are your best resource. Nothing compares to developing a body of knowledge built on application and industry-specific experience. As you build more complex programs, break them, and repair them the speed and reliability of the repairs increases. Most important, the *number* of errors will decrease as you write savvier, more bullet-proof code.

COMMENTS? QUESTIONS?

Your input is always welcome. Contact the author at:

102 Westbury Drive
Chapel Hill NC 27516-9154
919.942.2028
fcd@aimsco.com