



TUNNELLING THROUGH INHERITED CODE

We all have, or will, inherit programs that take a while to figure out. They may work without changes, but eventually you need to figure out what is really happening so you can modify the program.

Some of the things to start with to make inherited code more readable:

1. Use meaningful data set names in all steps, to help remember which data set is being created or used.
2. Use RUN statements to explicitly see the end of SAS steps.
3. When looking at large/complex programs, start to demystify by writing a rough data flow, or program flowchart, to help track where data comes from, and where it goes.
4. Examine each step from the top down, noting data sources and transformations.
5. Turn on all source OPTIONS such as SOURCE, SOURCE2 (included programs), MPRINT, and SYMBOLGEN (for macros). These show more information in the SAS log, useful for tracing logic.
6. Ask questions. Even if the original programmer is no longer around someone may know about the code.

To illustrate the process of how you could work through and make changes to a program written by someone else, we will use an example. The original source code for our example program is shown in Figure 1. The output is in Figure 2, and the SAS log is in Figure 3.

We have been asked to make the following changes:

- add a new department
- add a new employee
- show the employee id on the report

Figure 1 - Original Source Code

```

/* original code */
options nosource ls=80;

%macro macro1;
data z (keep=deptnum full);
length full $16;
set y;
full=fname !! lname;
format deptnum $deptfmt.;
label full='Employee name';
label deptnum='Department name';
label empid='employee id';
%Mend;

filename deptnam
'c:\temp\deptname.txt' ;
filename fmpgfm
'c:\temp\deptfmt.sas';

Data x;
infile deptnam end=eof pad;
input @1 deptno $3. @5 deptname
$15.;
file fmpgfm;
if _n_ = 1 then
put "proc format; value
$deptfmt";
put "" deptno " = " deptname
"";
if eof then put "; run;"; run;

%include fmpgfm;

data y;
input deptnum $ empid $ fname $
lname $;
datalines;
101 1001 steve last
102 1002 fred derf
103 1003 mike ekim
104 1004 mary gold
;

run;
%macro1;
proc print label;
title 'original report';
run;

```

Figure 2 - Output

original report			
OBS	Employee name		Department name
1	steve	last	Purchasing
2	fred	derf	Payroll
3	mike	ekim	Personnel
4	mary	gold	Shipping

Continued on page 3.....

IN THIS ISSUE

Tunnelling Through Inherited Code: Gerry Frey gives an overview and example of how to decipher an inherited program.....Page 1

SAS® Numeric Variable Lengths: Steve First shares how you can save time and space.....Page 2

Creating a Comma-Delimited Flat File: David Beam shows how the SSC Flat File Macro can help make your job easier.....Page 4

SAS® Help Desk I/O: Rosalind Gusinow answers a question about how to detect when you are at the end of an input file.....Page 5

Technical Credit:.....Page 7

Advantages of Face-to-Face Training: Sue Faust takes a look at how training programs are changing and points out some things to consider.....Page 8

Public Class Schedules:.....Page 8



SYSTEMS SEMINAR CONSULTANTS, INC.
2997 Yarmouth Greenway Drive
Madison, WI 53711
Website: www.sys-seminar.com
Email: train@sys-seminar.com
Phone: (608) 278-9964
Fax: (608) 278-0065

A GREAT MIX

WELL-ROUNDED TRAINERS AND CONSULTANTS



Part of what makes my job so interesting is that we at SSC don't just do training, or just do consulting, or just work on a help desk, but rather everyone here does all three. While each of these tasks has its good points, we think that doing all of them makes for very well-rounded IT professionals.

Training allows us to go to a variety of locations and work with large

numbers of people from the best companies. Preparing training material and answering students' questions give instructors the opportunity to learn obscure details that may not be encountered in project work for many years. I can also honestly say that even after many years of instruction, I always learn something from the students in every class. Either they pose a question that I need to research, or they share some insight on an interesting feature, or maybe they give a different viewpoint that I had not considered.

Consulting projects force us to write systems that are not like the storybook examples training sometimes provides. Usually it is not glamorous, but something that a client needs and wants. Our projects range from small quick ad-hoc projects to very large implementations. These projects give real-world experience that instructors can then share with students in our training classes.

Help desk questions run the gamut from very simple to very complex. There is nothing more frustrating for a SAS user than to need help and not know who to ask. Offering a one-stop place to help clients help themselves is one of the most satisfying things that we do.

Please keep us in mind if you have needs in any of these areas, we would love to hear from you.

Sincerely,

Steve First
President



SAS[®] NUMERIC VARIABLE LENGTHS

SAVE TIME AND SPACE BY USING LENGTH

While SAS stores character data virtually unchanged from the way it was input, numeric data is usually converted and stored much differently. All numeric values in SAS are stored in floating point format, which is also known as real binary. The default length of a numeric variable is 8 bytes, and because of the way floating point data is stored, this length can actual store integers with up to 16 significant digits. This internal length really has no relationship to the length and number of digits read into or written out of SAS.

An Example

Read in the following values:

```
data test;
  infile dd1;
  input @1 name $9.
        @10 rate 2.
        @20 hours 3. ;
datalines;
STEVE 10 22
BILL 20 451
;
run;
proc contents;run;
```

Partial Output:

#	Variable	Type	Len	Pos
3	hours	Num	8	8
1	name	Char	9	16
2	rate	Num	8	0

The PROC CONTENTS output shows, as we would expect, that the character field NAME is stored as nine bytes which is what was read. But even though RATE and HOURS are read as two and three digit values, respectively, they are both stored as a full 8 bytes.

Advantages and Disadvantages

Advantages of storing numbers this way are that all numerics are stored the same way regardless of input or output formats, and very large (up to 16 digits) values can be stored before any overflow conditions would occur.

Disadvantages are that it is a very expensive operation to convert from external formats to the floating point notation, and in some cases much more disk storage is used than is actually required. Having bigger records than actually needed can increase run times and data transfer times significantly.

The SAS LENGTH or ATTRIB statements can be used to explicitly define the default length for all numerics, or to set each variable's length individually. It is important to code this statement before any other usage of the variable, as the compiler sets variable length from the first statement defining a variable that it finds in the source code.

Setting the length attribute is solely to save storage space, as all SAS



SYSTEMS SEMINAR CONSULTANTS, INC.

Copyright © 2000 Systems Seminar Consultants, Inc. Madison, WI
All rights reserved. Printed in USA. The Missing Semicolon is a trademark of
Systems Seminar Consultants, Inc., SAS is a registered trademark of SAS Institute Inc., and
SyncSort is a registered trademark of SyncSort, Inc.
in the USA and other countries.

PROCs and the DATA steps use full 8-byte values for storage and calculations.

The table below shows the extent to which we can safely reduce numeric lengths without losing precision on the different SAS platforms.

Representation of Numeric Variables by the SAS® System			
	Length in bytes	Significant Digits Retained	Largest Integer Represented Exactly
MVS	2	2	256
	3	4	65,536
	4	7	16,777,316
	5	9	4,294,967,296
	6	12	1,099,511,627,776
	7	14	281,474,946,710,656
	8	16	72,057,594,037,927,900
	Windows/ UNIX	3	3
	4	6	2,097,152
	5	8	536,870,912
	6	11	137,438,953,472
	7	13	35,184,372,088,832
	8	15	9,007,199,254,740,990

Altering our example program to explicitly set the lengths will save us 10 bytes or approximately 40% of our file.

```
data test;
  length rate hours 3.;
  infile dd1;
  input @1 name $9.
        @10 rate 2.
        @20 hours 3. ;
datalines;
STEVE 10 22
BILL 20 451
;
run;
proc contents;run;
```

Partial Output:

#	Variable	Type	Len	Pos
3	hours	Num	3	12
1	name	Char	9	0
2	rate	Num	3	9

It is critical that we do not reduce numerics too much or an overflow will occur, and since SAS doesn't report this, this condition can be very difficult to detect and debug. Generally, we should only reduce integers, not values that have decimal digits.

In summary, setting lengths can provide significant savings with very little effort. For more detailed information please refer to the SAS documentation.



QUICK TIP

You can use the RUN CANCEL; statement to have a step compiled but not executed. This is great for checking SYNTAX!

```
data test;
  infile 'c:\temp\test.dat';
  input name $ age;
  years=age*12;
RUN CANCEL; /* step compiles but does not execute */
```



TUNNELLING THROUGH INHERITED CODE

CONTINUED FROM PAGE 1

Figure 3 - SAS Log

```
2 /* original code */
3 options nosource ls=80;
NOTE: The infile DEPTNAM is:
FILENAME=c:\temp\deptname.txt,
RECFM=V,LRECL=256
NOTE: The file FMTPGM is:
FILENAME=c:\temp\deptfmt.sas,
RECFM=V,LRECL=256
NOTE: 4 records were read from the infile DEPTNAM.
The minimum record length was 11.
The maximum record length was 14.
NOTE: 6 records were written to the file FMTPGM.
The minimum record length was 6.
The maximum record length was 27.
NOTE: The data set WORK.X has 4 observations and 2
variables.
NOTE: The DATA statement used 0.28 seconds.
NOTE: %INCLUDE (level 1) file FMTPGM is file
c:\temp\deptfmt.sas.
29 +proc format;
30 +'101 ' = 'Purchasing '
31 +'102 ' = 'Payroll '
32 +'103 ' = 'Personnel '
33 +'104 ' = 'Shipping '
34 +;
NOTE: Format $DEPTFMT has been output.
34 + run;
NOTE: The PROCEDURE FORMAT used 0.33 seconds.
NOTE: %INCLUDE (level 1) ending.
NOTE: The data set WORK.Y has 4 observations and 4
variables.
NOTE: The DATA statement used 0.17 seconds.
NOTE: The data set WORK.Z has 4 observations and 2
variables.
NOTE: The DATA statement used 0.17 seconds.
NOTE: The PROCEDURE PRINT used 0.16 seconds.
```

We deduce that to add a department, we must add it to the 'deptname.txt' file. So we add dept 105 (Receiving) to that file:

```
101 Purchasing
102 Payroll
103 Personnel
104 Shipping
105 Receiving
```

Continued on page 6.....

CREATING A COMMA-DELIMITED FLAT FILE

A GREAT SAS® MACRO

Systems Seminar Consultants, Inc. developed an easy to use SAS macro that can create a comma-delimited flat file from any SAS data set. This flat file can then be easily imported into a variety of software packages, including Excel, Lotus, and WordPerfect Mail Merge.

Our macro has recently been upgraded to work with version 8 of The SAS System, allowing data set names and column names more than 8 characters long.

How does the macro work? Suppose you have a SAS data set called "CUSTOMER" with the following data:

Obs	NAME	CITY	STATE	SALEDOLLARS	SELLDATE
1	Jones, Tom	Las Vegas	NV	3456.85	02/18/2000
2	Lake, Veronica	Los Angeles	CA	.	02/13/1999
3	Dailey, Richard	Chicago	IL	245.10	02/06/2000

Assume the SELLDATE variable has a format assigned of MMDDYY10. Using the macro, the following CSV (comma separated values) flat file can be created with one program statement:

```
"NAME", "CITY", "STATE", "SALEDOLLARS", "SELLDATE"  
"Jones, Tom", "Las Vegas", "NV", 3456.85, 02/18/2000  
"Lake, Veronica", "Los Angeles", "CA", ., 02/13/1999  
"Dailey, Richard", "Chicago", "IL", 245.1, 02/06/2000
```

This is the macro statement used to create the above flat file:
%SSCFLAT (MSASDS=CUSTOMER, MFLATOUT=ABCDE.WHATEVER.FLAT);

In the above example, there are two parameters to the macro call:
MSASDS= name of the SAS data file to read in
MFLATOUT= full name of the flat file being created

Parameters and Default Settings

The macro has several other parameters that may be used to control the operation, if you so choose. A list of the important parameters and the default settings follows.

Parameter

MSASDS=

Usage

SAS Data set name (Required!!)

One of the next two is required.

MFLATOUT=

File name of output file

MPREFIX=

Output file name Prefix

Optional parameters (these are default settings):

MHEADER=YES

Show variable names on first line

MLABEL=NO

Use variable LABELS on first line

MLIST=YES

Show records being created in SAS log

MMISSING="."

Missing numeric value character

For example, to create a flat file without the names of the variables in the first line, and missing numeric values showing a zero, use the following:

```
%SSCFLAT (MSASDS=CUSTOMER, MHEADER=NO, MMISSING=0,  
MPREFIX=C:\TEMPSAS\);
```

The resulting flat file gets a default name of "your data set name.dat", with the prefix of "C:\TEMPSAS".

Example: C:\TEMPSAS\CUSTOMER.DAT:

```
"Jones, Tom", "Las Vegas", "NV", 3456.85, 02/18/2000  
"Lake, Veronica", "Los Angeles", "CA", 0, 02/13/1999  
"Dailey, Richard", "Chicago", "IL", 245.1, 02/06/2000
```

Source Code

The source code for this macro is available by contacting Systems Seminar Consultants. If the macro is not installed at your site as an automatic SAS macro, first include the macro program in your logic as follows:

Example for PC/SAS:

```
%INCLUDE 'c:\path\SSCFLAT.SAS';
```

Example for MVS:

```
%INCLUDE 'AA.BB.CC (SSCFLAT)';
```

The SSCFLAT macro has been tested under the MVS, Unix, and Windows versions of SAS. We find using the SSCFLAT macro to be one of the easiest methods for getting data out of SAS and into another application.

Version 8 of Base SAS does have a new feature to create CSV files from a data step, but you still must code the PUT statement. The SSCFLAT macro eliminates the "hard" work of changing the code for each new data set.

QUICK TIP

A simple way to block a section of SAS code from being processed is to make it look like it is a piece of macro code but never invoke it:

```
%MACRO DUMMY; /* Starts a MACRO definition */  
DATA WHATEVER;  
.....  
RUN;  
PROC PRINT DATA=WHATEVER;  
.....  
%MEND DUMMY; /* End of block to ignore */
```



SAS® HELP DESK I/O

SOLUTIONS FROM OUR HELP DESK SERVICE

Q: "I have a program that creates some subtotals. I wish to display those counters in the log when I am done processing my non-SAS file in my data step. I am using the END= option on my INFILE statement. Sometimes the counters are displayed and sometimes they are not. Why doesn't my end of file processing work every time?"

A: There are two SAS options that allow you to detect when you are at the end of an input file and thus perform some special processing. The most commonly used method is the END= option, which allows you to define a variable whose value is zero (0) until your INPUT statement reads the last record. When the *Input statement reads the last record*, the value of the variable is set to one (1). Then somewhere within your data step, you have an IF statement that checks the value of this variable, and if it is one (1), your end of file processing is performed.

Using the END= Option

The problem you have encountered occurs because you have some processing within your data step that bypasses(deletes) some records. These IF statements interrupt the flow of your program, and the IF statement that checks for the end of the file may never be executed. For example, if the last record in your file happens to be one that is bypassed, the flow of the program goes back up to the top of the data step where the step stops. The processing never drops through to your end of file check. (See Example 1.)

The following program creates the flat file used for testing in all of the examples.

```
FILENAME RAWIN 'C:\TEMP\ENDEOF.SAS';
DATA NULL;
  FILE RAWIN;
  PUT '1';
  PUT '2';
  PUT '3';
  PUT '1';
  PUT '4';
  STOP;
RUN;
```

Example 1: END= Option with Subsetting IF

```
DATA ENDSUB;
  INFILE RAWIN END=EOF;
  INPUT TYPE;
  IF TYPE NOT IN (1, 2) THEN DELETE;
  TOTAL+1;

  ** NEVER EXECUTED BECAUSE THE LAST RECORD IS BYPASSED
  DUE TO IF..THEN..DELETE; **;
  IF EOF THEN DO;
    PUT 'END OF DATA STEP WITH EOF= OPTION ' TOTAL=;
  END;
RUN;

PROC PRINT DATA=ENDSUB;
  TITLE 'ENDSUB';
RUN;
```

One way to fix this problem is to add another IF statement, prior to each of your IF...Delete statements, that checks for end of file as well

QUICK TIP

Numeric variables default to 8 bytes in SAS. If you have a numeric date or time value, these can be safely stored in a 4 byte numeric variable (5 bytes for WINDOWS/UNIX), such as:

```
LENGTH MYDATE MYTIME 4;
```

```
INPUT @1 MYDATE MMDDYY10.
      @15 MYDATE TIME8.;
```

run;

as the deletion criteria. (See Example 2.) This could become cumbersome and inefficient if you have a number of conditions that result in bypassing records.

Example 2: END= Option with Subsetting IF - IF Statement Fix

```
DATA ENDSUB;
  INFILE RAWIN END=EOF;
  INPUT TYPE;
  IF EOF AND TYPE NOT IN (1, 2) THEN DO;
    PUT 'END OF DATA STEP WITH EOF OPTION, IF
STATEMENT FIX ' TOTAL=;
  END;
  IF TYPE NOT IN (1, 2) THEN DELETE;
  TOTAL+1;

  ** NEVER EXECUTED BECAUSE THE LAST RECORD IS BYPASSED
  DUE TO IF..DELETE; **;
  IF EOF THEN DO;
    PUT 'END OF DATA STEP WITH EOF= OPTION ' TOTAL=;
  END;
RUN;

PROC PRINT DATA=ENDSUB;
  TITLE 'ENDSUB';
RUN;
```

Using the EOF= Option

The second method of detecting an end of file condition is the EOF= option. (See Example 3.) It specifies the label of a section of your program that your program will branch to when the end of file condition is detected. This method detects the end of file when it tries to *read past the end of the file* (i.e., when your INPUT statement tries to read a record and there are no more records left in the file).

Example 3: EOF= Option

```
DATA ENDSUB;
  INFILE RAWIN EOF=DONE;
  INPUT TYPE;
  IF TYPE NOT IN (1,2) THEN DELETE;
  TOTAL+1;
  RETURN;
  DONE:
  * NOTE THE COLON AFTER THE LABEL *;
  PUT 'END OF DATA STEP WITH EOF= OPTION '
    TOTAL=;
  * STOP PREVENTS 1 EXTRA IMPLIED OUTPUT *;
  STOP;
  RETURN;
RUN;
```

Only the third example will put the end of data message in the log. Note the difference in timing between the END= and the EOF= options.

run;

TUNNELLING THROUGH INHERITED CODE

CONTINUED FROM PAGE 3

But how do we add employee id to the report? The original SAS log is very hard to decipher. We need to see what is really happening. To accomplish this, we turn on the SOURCE, SOURCE2, MPRINT and SYMBOLGEN options and then run the program again. The resulting SAS log, listed below, is more complete and much more informative.

```
2      /* original code */
3      options source source2 mprint symbolgen ls=80;
4
5      %macro macro1;
6      data z (keep=deptnum full);
7      length full $16;
8      set y;
9      full=fname !! lname;
10     format deptnum $deptfmt.;
11     label full='Employee name';
12     label deptnum='Department name';
13     label empid='employee id';
14     %mend;
15
16     filename deptnam 'c:\temp\deptname.txt' ;
17     filename ftmpgm 'c:\temp\deptfmt.sas';
18
19     Data x;
20     infile deptnam end=eof pad;
21     input @1 deptno $3. @5 deptname $15.;
22     file ftmpgm;
23     if n = 1 then
24         put "proc format; value $deptfmt";
25         put "   " deptno "   " = "   " deptname "   ";
26         if eof then put "   "; run;"; run;
NOTE: The infile DEPTNAM is:
      FILENAME=c:\temp\deptname.txt,
      RECFM=V,LRECL=256
NOTE: The file FMTPGM is:
      FILENAME=c:\temp\deptfmt.sas,
      RECFM=V,LRECL=256
NOTE: 4 records were read from the infile DEPTNAM.
      The minimum record length was 11.
      The maximum record length was 14.
NOTE: 6 records were written to the file FMTPGM.
      The minimum record length was 6.
      The maximum record length was 27.
NOTE: The data set WORK.X has 4 observations and 2
variables.
NOTE: The DATA statement used 0.27 seconds.
27     %include ftmpgm;
NOTE: %INCLUDE (level 1) file FMTPGM is file
c:\temp\deptfmt.sas.
29     +proc format;
29     +       value $deptfmt
30     +'101 ' = 'Purchasing '
31     +'102 ' = 'Payroll '
32     +'103 ' = 'Personnel '
33     +'104 ' = 'Shipping '
34     +;
NOTE: Format $DEPTFMT has been output.
34     + run;
NOTE: The PROCEDURE FORMAT used 0.27 seconds.
NOTE: %INCLUDE (level 1) ending.
```

```
35
36     data y;
37     input deptnum $ empid $ fname $ lname $;
38     datalines;
NOTE: The data set WORK.Y has 4 observations and 4
variables.
NOTE: The DATA statement used 0.17 seconds.
43     ;
44
45     run;
46     %macro1;
MPRINT(MACRO1): DATA Z (KEEP=DEPTNUM FULL);
MPRINT(MACRO1): LENGTH FULL $16;
MPRINT(MACRO1): ;
MPRINT(MACRO1): SET Y;
MPRINT(MACRO1): FULL=FNAME !! LNAME;
MPRINT(MACRO1): FORMAT DEPTNUM $DEPTFMT.;
MPRINT(MACRO1): LABEL FULL= 'Employee name';
MPRINT(MACRO1): ;
MPRINT(MACRO1): LABEL DEPTNUM= 'Department name';
MPRINT(MACRO1): ;
MPRINT(MACRO1): LABEL EMPID= 'employee id';
MPRINT(MACRO1): ;
NOTE: The data set WORK.Z has 4 observations and 2
variables.
NOTE: The DATA statement used 0.17 seconds.
47     proc print label;
48     title 'original report';
49     run;
NOTE: The PROCEDURE PRINT used 0.11 seconds.
```

By looking through the more complete log, several things become apparent. The data is manipulated in the macro and three data sets are being created (x, y and z).

We can now add comments and straighten up the code. We are reading the 'deptname' flat file and creating a format from it. (Note: you could do this by using the CNTLIN feature with PROC FORMAT.) The revised source code is in Figure 4, with the final output in Figure 5.

Figure 4 - Revised Source Code

```
/* revised code */
options source2
      mprint symbolgen linesize=80;

/* start macro1, this reads dataset empname */
/* and creates the fullnam dataset for */
/* the report. */
%macro macro1;
data fullnam (keep=deptnum empid full);
length full $16;
set empname;

* create full name from first and last;
* trimming extra blanks after first name;
full = trim(fname) !! ' ' !! lname;

* assign the format we created;
format deptnum $deptfmt.;

* assign labels to name and deptnumber;
label full='Employee name';
label deptnum='Department name';
label empid='employee id';

run;
%Mend macro1;

/* assign file with dept numbers/names for format */
filename deptnam 'c:\temp\deptname2.txt' ;

/* assign file to write out proc format code */
filename ftmpgm 'c:\temp\deptfmt.sas';

/* create the code for the department name format */
Data _null_;

* this is the file with the department number and name;
```

QUICK TIP

To read a random sample of roughly 15% of the data from any file, use the RANUNI(0) function:

```
DATA TESTING;
INFILE RAWIN;
INPUT @; /* read a record and hold it */
IF RANUNI(0) LT .15; /* allows about 15% of rows to be used */
INPUT rest of program.....
```



```
* we use the end of file option (end=) to allow writing;
* last line of code in proc format, and the pad to;
* prevent problems reading input lines that are too short;
```

```
infile deptnam end=eof pad lrecl=50;

* read in a record;
input @1 deptno $3.
      @5 deptname $15.
      ;

* send output from put statements to file;
file ftmpgm;

* if we are processing the first record, write;
* out the SAS code for the proc format first;
if n = 1 then
  put "proc format; value $deptfmt";

* for each record on the department name file, write;
* out a record for the values in the format;
* (this builds a line looking like: '1001' = 'Purchasing';
xline = "" !! deptno !! " = " !! deptname !! "";
put xline;

* if we have processed the last record then write;
* out the ending statements for the proc format;

if eof then
  put "; run;";

run;

/* include and run the proc format step in file ftmpgm */
%include ftmpgm;

/* step to read employee data and create dataset empnam */
data empnam;
  input deptnum $ empid $ fname $ lname $;
  datalines;
101 1001 steve last
102 1002 fred derf
103 1003 mike ekim
104 1004 mary gold
105 1005 penn teller
;
run;

/* macro reads empnam and creates fullnam and */
/* assigns the format we created */
%macro1;

/* this step prints our report */
proc print data=fullnam label;
  title 'revised report';
run;
```

Figure 5 - Revised Report

revised report			
OBS	Employee name	Department name	employee id
1	steve last	Purchasing	1001
2	fred derf	Payroll	1002
3	mike ekim	Personnel	1003
4	mary gold	Shipping	1004
5	penn teller	Receiving	1005

We have added 'empid' to the KEEP statement in the first data step, and a label statement for 'empid'. Overall, we have made the necessary changes, cleaned up and commented the code, and hopefully, helped anyone else who will read this program in the future.

Some simple documenting tips we SAS professionals might want to use

QUICK TIP

Submitting code with missing quote marks (single ' or double ") in Display Manager or Windows/SAS can be frustrating to fix. Try submitting the following to close off the mis-quoted string:

```
*'; *"; run;
```



include the following:

- Add comments when changing a program, including the date changed, per whose request, fields/files added, business rules involved, descriptions of tricky logic, etc.
- Keep a log of modifications (adding fields, revisions to reports, etc.)
- Use clean coding styles, use blank lines and indenting for readability.

The most important thing to remember is to assume that every program you write or change is likely to be used in the future, by you or someone else. Clean code and clear comments are invaluable to the next person who needs to work with the program.



TECHNICAL CREDIT AND RECOGNITION



Gerald Frey
Trainer/Consultant
Author of this issue's:
Tunnelling Through Inherited Code, Page 1



Steve First
President
Author of this issue's:
SAS® Numeric Variable Lengths, Page 2



David Beam
Vice President
Author of this issue's: Quick Tips, Pages 3-7 and
Creating A Comma-Delimited File, Page 4



Rosalind Gusinow
Trainer/Consultant
Author of this issue's:
SAS® Help Desk I/O, Page 5



Sue Faust
Trainer/Consultant
Author of this issue's:
Advantages of Face-to-Face Training, Page 8

PublisherCindy Kersten
EditorsDavid Beam, Steve First, & Cindy Kersten



ADVANTAGES OF FACE-TO-FACE TRAINING

REAL PEOPLE MAKE THE DIFFERENCE

In today's electronic world, corporate training programs as well as many educational institutions are revamping their curriculum to offer programs in a distance education mode. Sometimes this mode has completely replaced the face-to-face delivery method we have all experienced. How do you know if this approach will fit your training needs?

First of all, let's look at a definition of distance education. In general, it refers to some program or class, which has a separation of time, and/or space, between student and instructor. Methods currently being used to offer distance education programs include print media (correspondence courses), telephone (audio) based, audiographic conferencing, pre-produced videos, televised instruction, videoconferencing, and web-based technologies.

When considering which kind of training program to attend, consider the following advantages of face-to-face training:

- **Familiarity with the process:** Live, interactive learning is what we are familiar with, like the school environment we experienced while growing up.
- **Feeling connected:** Seeing fellow students, raising questions, and receiving instant feedback make us feel more connected and part of a 'real' class. This two-way communication also makes it easier for the facilitator to establish rapport among students.
- **Student response:** Feeling confused about something? In a face-to-face classroom your instructor can see your puzzled look, as well as other facial expressions and body language, that convey how you are responding to the material.
- **Feedback:** Your questions are responded to immediately, and not at some future point in time. (At which time you may have forgotten your questions!) Not only can questions be discussed verbally, but whiteboards, flip charts, or transparencies can be used to visually illustrate a response to questions.
- **Cost-effectiveness:** One author states that "there is little evidence in the the United States that distance education reduces costs over

SSC PUBLIC CLASS SCHEDULE ST PAUL, MN

Introduction to SAS® October 16-18 \$725 December 4-6 \$725	SAS® Efficiencies November 16 \$350
The SAS® SQL Procedure October 20 \$350	SAS Macros® November 14-15 \$525
Advanced SAS® December 11-13 \$725	SyncSort® November 17 \$350

**To register call (608) 278-9964 or visit
www.sys-seminar.com**

SAS INSTITUTE PUBLIC CLASS SCHEDULE MADISON, WI

Statistics I: Introduction to ANOVA, Regression, and Logistic Regression October 10-12 \$975
SAS® Programming II: Manipulating Data with the DATA Step October 17-19 \$975
SAS® Macro Language November 15-16 \$650

**To register call (800) 333-7660 or visit
www.sas.com/training**

traditional instruction." (Willis, 1994, p. 61) Distance education training classes tend to have higher costs in terms of course development, equipment, and ongoing support.

Deciding how to meet your training needs takes careful thought with consideration given to your content, how you like to learn, and your schedule. We hope the points mentioned above highlight advantages of face-to-face learning!



SYSTEMS SEMINAR CONSULTANTS, INC.

2997 Yarmouth Greenway Drive
Madison, WI 53711

PRSR T STD
U.S. POSTAGE
PAID
MADISON, WI
PERMIT #2783

Call or visit our web site for your
free subscription to
The Missing Semicolon™ .