Paper 235-26

A Bit More on Job Security: Long Names and Other V8 Tips

Arthur L. Carpenter, California Occidental Consultants
Tony Payne, SPSInfoquest

ABSTRACT

With the advent of the Nashville releases of the SAS® System (Versions 7 and 8), a number of new opportunities have been added to the toolboxes of Job Security afficionados. In previous presentations and papers (Carpenter, 1993; Carpenter and Payne, 1998) a series of tips and techniques have been foisted on the unsuspecting SAS user community with the objective of increasing the job security of SAS programmers by making them indispensable to their bosses.

The list of tips continues to grow. And now with the added features of the Nashville releases, the list is greatly expanded. One area of special interest for job security specialists has revolved around the naming of variables and data sets. Now with the relaxed restrictions for SAS names comes a large number of both challenges and opportunities. New system options and the advent of the Output Delivery System, ODS, also provide a number of techniques that will surely become favorites in the future. Of course new tips and techniques for all releases are constantly being uncovered, and a number of these are also included.

Become a Job Security Advocate.

KEY WORDS

Job Securtiy, datastmtchk, yearcutoff, literal, ODS, audit tail, integrity constraints

VARIABLE NAMES

Long Variable Names

With 8 characters we might employ naming variations such as:

```
QWRTXZQR HHHIIHIH WVWVWVVW
QWRTZXQR HHIHIHIH WVWWVWVW
```

Since variable and data set names can now have up to 32 characters, we no longer need to stop so soon.

```
WVWVWVVWVWWVWVWVVVWWWVVVVWWW
WVWWVWVWVWWVWVWVVVWWWVVVVWWW
```

This is of course not very subtle, try instead:

```
ThisVariableStoresThePatientHeight
ThisVariableStoresThePatientHieght
```

Odd Variable Names

You can make any character valid in a variable name by setting the system option VALIDVARNAME to ANY. The variable name is then enclosed in a name literal. The statement:

```
name='Tom'n;
```

assigns the value of the variable TOM (not the letters T-O-M) to the variable NAME. If this was not valuable enough by itself, add spaces and arithmetic operators:

```
data new;
'Tom; wt=5+wt'n = 5;
name='Tom; wt=5+wt'n + 6;
put 'Tom; wt=5+wt'n= name=;
run;
```

The LOG shows:

```
'Tom; wt=5+wt'n=5 name=11
```

Other possibilities include:

```
"a dog chases a cat"n=5.5;
"3+3"n="Six";
normvar_6="3+3"n;
```

The use of the name literal should prove quite useful because our eyes and minds have been trained to see quoted strings as literal values, not quoted variable names. We also will have to now look at statements with blanks and other symbols differently. At the very least we can now force those who have the misfortune to read our programs to spend a lot more time scratching their heads.

Misdirected Names and Labels

Since long names can contain more information make use of it. The following statement is used to store the patient's age.

```
height_of_patient=27;
```

You can now tell a whole story with long labels. For the same variable:

```
Label height_of_patient = 'The patients
height has been recorded in inches unless
the month was odd, then oddly enough, it
is recorded in years';
```

USING QUOTES AND COMMENTS

Unbalanced Quotes

BAD NEWS FOR JOB SECURITY EXPERTS: V8 has a very

nasty context editor which tells you when quotes are unbalanced. Techniques involving unbalanced quotes must be reviewed and are often no longer recommended. The Enhanced Editor also identifies keywords and may also invalidate some techniques using keyword-variable name confusion. Missing semi-colon and extended comment techniques are also jeopardized. Fortunately the Enhanced editor can be turned off and is not required at SAS installation.

Titles and Labels
Quotes are no longer required in the TITLE and LABEL statements. This was true in earlier releases also, but has not been heavily advertized. What is the value of AGE after applying these two assignment statements?

```
age = 10;       *Add a;label
age = age + 5; *Add 5 to age;
```

Oops, did I say assignment statements? I meant to mention the label statement. The value of AGE, of course, remains 10.

Invisible Comments
With the Enhanced Editor pointing out incomplete comments, it becomes more important to be able to hide dangling comments. Consider adding a portion of a comment at the end of a tried and true, known to be good program that is brought in through the %INCLUDE statement. HINT: Folks tend to not check things that are known to be good.

```
.....known good code.....
/* end of good code


%include knowngood;
data new; set old; run;
/* above step is VERY important */
```

The DATA step has now been completely commented, and it is actually the `*/` that terminates the comment started in the included code. The first `*/` that SAS comes across 'closes' the comment, and SAS starts processing again, having ignored all the code between the %include and the end of the comment.

V8 OPTIONS

Yearcutoff
The default value of YEARCUTOFF changes from 1900 to 1920. While this change may be sufficient, consider the following OPTION statement to assign YEARCUTOFF.

```
options
yearcutoff=%sysevalf(%sysfunc(ranuni(0))*1
00+1900,ceil);
```

Another interesting aspect of the YEARCUTOFF option is that although SAS can correctly handle 5 digit years none of the formats can display all five digits. As is demonstrated below, formats that show two digits of the year work fine (with the added benefit of being a tad misleading), while formats that attempt to show 4 digits fail.

```
options yearcutoff=19200;
data a;
date = '28jun00'd;
year = year(date);
put year=;
put date=date7. ;
put date=date9. ;
run;
```

The LOG shows:

```
year=19200
date=28JUN00
date=28JUN****
```

Datastmtchk
In Version 6 there were no restrictions on the use of keywords, such as, DATA, SET, MERGE as data set names. This is not necessarily true in Version 8. The following step is allowed in V6:

```
data set;
set data;
```

In Version 8 the DATASTMTCHK option must be reset to NONE to prevent this step from failing to compile.

Tuning Options
Use tuning options liberally and inappropriately - this is especially good, as people will assume that these were used for a purpose, and so will be reluctant to remove or change them. This can be helpful even if you just set them to their default values.

MACRO STUFF

Implied Macros
Implied macros have not gone away and can still be useful for creating interesting macros. Hide the following macro definition in someone else's AUTOEXEC.SAS:

```
options implmac;
%macro data ( a, b, c, d, e ) / stmt;
%* this macro has nothing in it;
%mend data;
```

Once this macro has been defined, DATA steps will no longer be recognized. In the following step, the statement `a=2;` is flagged as an error.

```
data new;
a=2;
run;
```

Using Quotes
Quote marks are generally not needed in the Macro Language, however, since DATA step enthusiasts are used to seeing them, they can be useful. Why is the following %IF always false?

```
%let city = Scottsdale;
%if &city='Scottsdale' %then %do;
```

This one is also always false.

```
%if "&city"='Scottsdale' %then %do;
```

Since the quotes are part of the text that is compared (unlike in the DATA step) the first character on the left (") is always different from the first character on the right (').

IN THE DATA STEP

Changing the BY Values in a Merge
In a DATA step merge consider changing the value of one or more BY variables through assignment statements or DO loops. Under some conditions this can produce pleasantly unpredictable consequences.

Dissimilar BY Variable Attributes in a Merge
In a MERGE the attributes of the BY variable are determined by the first data table noted in the MERGE statement. If the attributes of the BY variables in the other data set(s) are different (especially character variables that are longer), the correct observations will not necessarily be matched. Although the following two data sets (A and B) have different values of the BY variable, the observations are matched and joined together.

```
data a;
name='Tom';
ina='yes';

data b;
name='Tommy';
inb='yes';

data ab;
merge a b ;
by name;
```

The data set AB contains:

```
 Obs    name    ina    inb
  1     Tom     yes    yes
```

Creating Unanticipated Variables
When numbered range list notation is used, every variable in the list that is not already on the PDV will be added (to the end of the PDV), this includes implied elements of the list. This can have the advantage of actually adding variables to the PDV. Consider the following DATA step:

```
data new;
set old(keep=name yr97 yr99);
mean = mean(of yr97-yr99);
```

The data set NEW will contain the variable YR98 although it is never mentioned in the code. For the same reason the following SET statement will fail if OLD does not contain the variable YR98.

```
set old(keep=name yr97-yr99);
```

Unanticipated Variable Order
Variables are added to the Program Data Vector in the order that they are encountered during the compilation process. Assume that the data set OLD contains the variables X1 and X4. In the following step, the values of SUM1 and SUM2 are not the same:

```
32   data old;
33   x1=1; x4=4; output;
34   run;
35
36   data new;
37   set old;
38   x2=2; x3=3;
39   sum1 = sum(of x1-x3);
40   sum2 = sum(of x1--x3);
41   put sum1= sum2=;
42   run;

sum1=6 sum2=10
```

Unequal Variable Lists
When concatenating data sets the lists of variables in the incoming data set do not have to be the same. The `defer=open` option changes this behavior. In the following DATA step only those variables in the data table ONE will be found in the table BOTH.

```
data both;
set one two defer=open;
run;
```

Capitalization
With V8 the capitalization of the variable name is 'remembered' from the first time you reference the variable. You can later reference the variable with any capitalization scheme, however, the original capitalization is used by SAS procedures. PROC PRINT attempts to break long names at the capital letters. Consider the following (note the use of the LABEL option as a further distraction):

```
data test;
cAPItalIZedVaRIAble="Hello World";
output;
proc print label noobs;
var CapitaliZedVariable;
run;
```

The output shows:

```
  cAPItal
  IZedVa
  RIAble

Hello World
```

Integrity Constraints
The following integrity constraint appears to check for ages between 16 and 65, however since the comparison operators are backward it is impossible to add observations to this data set.

```
proc sql;
```

```
  create table test(
  age num,
  constraint chk1 check (age lt 16),
  constraint chk2 check (age gt 65)
  );
quit;
```

Of course if you assign integrity constraints to a data table and then replace the table in a DATA step the constraints are lost. This allows you to add data that should NOT be allowable.

Audit Trails
Emphasize in your program the use of audit trails. Then never actually use them. As with the integrity constraints the audit trail information will be lost if the data set is modified in the DATA step. Consider password protecting the separate audit table while leaving the primary data table unprotected.

Generation Data Tables
Through the use of the GENMAX= option on the DATA statement, multiple copies of a data set can now be automatically kept. Setting this option to a large number will force the system to save many copies of the data set. Of course you will never use these copies.

Indexing
Indexing data tables has the disadvantage of speeding data subsetting especially when used in conjunction with the WHERE statement. However, indexes also take time and effort to create and maintain. A further advantage is that indexes must be stored, and they can take a non-trivial amount of space. Consider indexing all variables in a data table (except of course those that are used in BY statements or WHERE clauses).

The BUFSIZE Data Set Option
The BUFSIZE option sets the data set page size. Unfortunately SAS normally chooses a sensible value for the BUFSIZE, but you can set your own value to great advantage. If you select a value which DOES NOT QUITE allow two observations per page, the resulting file will be nearly twice as big as it otherwise would be.

```
data big(bufsize=4096);
  length test $2048;
  retain test ' ';
  do i=1 to 100;
    output;
  end;
run;
```

Terminology
The use of certain terms used in SI documentation is in the process of being converted from what has been traditionally used in the DATA step. Until the transition is complete there will be opportunities for the Job Security Specialist. When talking to SQL programmers, always use the terms "Data Set", "Observation" and "Variable". Conversely, when talking to traditional SAS programmers, always use the terms "Table", "Row" and "Column".

SAS Spell Checker
Take advantage of the SAS spell checker. You may have noticed that SAS corrects some of your spelling mistakes – for instance if you miss-spell DATA as DAT, SAS will correct you. The two DATA steps below are equivalent!

Original

```
DATA new;
  SET sashelp.class;
  FILE OUTPUT;
  IF age>12 THEN DO;
    OUTPUT;
    PUT _all_;
  END;
RUN;
```

Corrected Alternative

```
DATE new;
  SE sashelp.class;
  FILO OUTPUT;
  FI age>12 THE DOO;
    OUTP;
    PU _all_;
  EN;
RUNE;
```

By the way, you can now happily discuss your “DATE” steps, and your “FI THE DOO” statements!

As an aside you can use this technique to beat the enhanced editor with valid code which looks invalid.

We will leave it to the gentle reader’s imagination as to what will happen if new definitions are added to the spell checker’s dictionary.

USING ODS

Templates and Styles
ODS output depends to a very great extent on both TEMPLATES and STYLEs. These have default definitions provided with the SAS System. Fortunately you have the ability to modify or replace them with your own. Hide your modified versions (with the original name) in a concatenated library for greatest effect. Suggested modifications to templates include the removal of selected columns (such as probability levels) from the output in traditional procedures. Modified styles allow you to change colors and fonts. This would include specifying the same color for the background and foreground.

Output Routing
By default the output goes to the output window (although this behavior can be changed through the DMS options). Additionally what is to be routed and to where is to be routed, can be controlled through the judicious use of ODS statements. Consider the following statement for inclusion in the AUTOEXEC.SAS:

```
ods listing close;
```

The user will now receive no output/listings in the output window. If the user is not explicitly using ODS, they may not think to try turning the ODS LISTING back on.

OUTSIDE THE JOB

Mixing Data Types
Always put V6 and V8 tables and catalogs together in the same library. The default engines for both V6 and V8 will allow you to 'see' only one type of data set. Mixing data set types from different versions in one directory (sorry this won't work on the mainframe) will render one of the types of data invisible to SAS. This is a great way to hide V6 catalogues and tables!

When the following LIBNAME statement is issued from within a Version 8 session, any Version 6 data sets in the \mystuff directory will not be recognized, when V8 data sets are also present.

```
libname alldata 'c:\mystuff';
```

If you then continue to use both V6 and V8 to update your tables and catalogs, you will find that changes made with V6 are not reflected in the V8 tables and visa versa. Soon you can have two data sets with the same name, in the same directory, with very different data values.

Use the Autoexec Option
The -AUTOEXEC option can be used at session initialization to execute a file of specialty options and techniques especially tailored for the job security specialist. By default the SAS will look for a file called AUTOEXEC.SAS, but of course the connoisseur will be able to pick a better name. Remember that although the file must contain SAS code the extension does not need to be SAS. Consider the following:

```
-autoexec 'c:\pictures\toesurgery.jpg'
```

Enhanced Editor Colors
The Enhanced Editor has the irritating habit of alerting users to the presence of comments and quoted strings that extend past their usual or expected boundaries. This limits the scope of a number of Job Security techniques. Fortunately you can fight back. The color of a variety of kinds of statements can be set independently of each other. Consider changing the color of comments or quoted strings to the same color as assignment statements. Or better yet change the color of comments to the background color of the editor (usually white). Now just for fun type a few random asterisks.

USING PROCEDURE OPTIONS

NODUPLICATES
The NODUPLICATES option in PROC SORT usually eliminates duplicate observations. Since most users will assume 'always' when the truth is closer to 'usually' there is room to construct a 'technique'. For the data set NAMES, shown here, the PROC SORT will find no duplicate observations when sorting by NAME..

```
NAME          AGE
Albert         15
Albert         25
Albert         15

proc sort data=names noduplicates;
by name;
```

Duplicates remain undetected when they are not adjacent after sorting. The V8 system option SORTDUP= should be set to its default (PHYSICAL), which mimics V6 behavior and maximizes the potential for errors such as the one shown above. When SORTDUP= is set to LOGICAL, observations are checked after applying any KEEP= or DROP= data set options, thus making duplicates easier to detect.

Multilabel Formats
Some procedures such as FORMAT and TABULATE now support multilabel formats. These formats allow overlapping ranges when providing grouping information. In the following table the sales values are grouped by a multilabel format with overlapping ranges. This causes the overall count (TOTAL) to appear to be incorrect. In fact the overall count is indeed 10 but the user will have no clue as to how to distribute those 10 values.

	Count
sales	
High	5.00
Low	4.00
Moderate	5.00
Total	10.00

The above table was generated by using the following program steps.

```
proc format;
  value salesgrp (multilabel)
     0-3 = 'Low'
     2-6 = 'Moderate'
     5-9 = 'High';

proc tabulate data=a;
  class sales / mlf;
  table sales all='Total',n='count';
  format sales salesgrp.;
run;
```

Formats that Misdirect
Formats are commonly used to change the way which a value is displayed without changing the value itself. The following format definition will cause SAS to display ‘FEMALE’ whenever the formatted variable takes on the value of ‘m’.

```
proc format;
value $sxvalue 'm'='Female' 'f'='Male';
```

Even more useful might be a format that reverses or changes the values of the variable.

```
proc format;
value $gender 'male' ='female'
'female'='male';
```

Formats and ODS

When working with ODS, formats can be used with a number of procedures, such as TABULATE and REPORT, to create what is known as traffic lighting. Formats can be used to change the color of the background, color of the foreground, the font, and so on. This can be used in several ways. An unsubtle approach would be to turn the background color red for low values when the TITLE specifies that high values have a red background. An alternative would be to change both the foreground and the background to the same color, thus rendering the text invisible. This is done for values over 200 in the following formats:

```
proc format;
   value ampback
      low-<0   = 'red'
      0-<200   = 'yellow'
      200-high= 'white';
   value ampfore
      low-<0   = 'white'
      0-<200   = 'blue'
      200-high= 'white';
   run;
```

These formats are then used with the STYLE= option in either a TABULATE or REPORT. The following shows a TABLE statement in a TABULATE step.

```
table product*ampida,
  (n*f=3.0 min median max)*
     {style={background=ampback
             foreground=ampfore.}};
```

SUMMARY

Things go wrong when programming. In actuality we need to do all we can just to NOT make the kinds of errors and mistakes advocated in this paper. Knowing that they exist provides us with knowledge that will help us avoid programming traps and pitfalls. Besides your job security is enhanced automatically if you can decode and understand programs written by another competing Job Security Specialist.

REFERENCES

Additional information on Job Security techniques can be found in:

Carpenter, Arthur L., Programming for Job Security: Tips and Techniques to Maximize Your Indispensability, presented at the 18th SAS User's Group International, SUGI, meetings (May 1993) and published in the Proceedings of the Eighteenth Annual SUGI Conference, 1993.

Carpenter, Arthur L. and Tony Payne, Programming For Job Security Revisited: Even More Tips and Techniques to Maximize Your Indispensability, Presented at the 23rd SAS User's Group International, SUGI, meetings (March, 1998) and published in the Proceedings of the Twenty-Third Annual SUGI Conference, 1998.

Both papers have been presented at numerous other conferences and can be found in their respective proceedings. Contact the authors for more details.

ABOUT THE AUTHORS

Art Carpenter's publications list includes three books (*Annotate: Simply the Basics*, *Quick Results with SAS/GRAPH® Software*, and *Carpenter's Complete Guide to the SAS® Macro Language*), two chapters in *Reporting from the Field*, and over three dozen papers and posters. Art has been using SAS since 1976 and has served in a variety of positions in user groups at the local, regional, and national level.

SAS Certified Professional™ Version 6

SAS Alliance Quality Partner

Art is a SAS Alliance Quality Partner™ and SAS Certified Professional™. Through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

SAS Institute Quality Partner

SAS Certified Professional™ Version 6

Tony Payne has worked as a SAS developer, project manager and course instructor for Software Product Services (now SPS-InfoQuest) since 1986. His main area of specialization is in applications development. Tony has written six papers presented at SEUGI and other conferences. Unlike Art he is yet to write a book or chair a conference. Tony is also a SAS Certified Professional™, and SPSInfoQuest is a Quality Partner of SAS Institute, UK.

ACKNOWLEDGMENTS

The authors would like to express their appreciation to Adam Crisp of SPSInfoQuest and FiTheDoo of Wimbledon for the generous contribution of a number of tested techniques, and to numerous other contributors who had the good sense to remain anonymous.

AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
P.O. Box 6199
Oceanside, CA 92058-6199

(760) 945-0613
art@caloxy.com
www.caloxy.com

Tony Payne
SPSInfoquest
The Mill, Abbey Mill Business Park
Eashing, Godalming,
Surrey , GU7 2QJ UK

+44 (0)1483 18422
tpayne@spsweb.com
www.spsweb.com

TRADEMARK INFORMATION