Programming for Job Security:
Maximize Your Indispensability - Become a Specialist

Arthur L. Carpenter
California Occidental Consultants

ABSTRACT

A great deal has been said about programming techniques designed for the efficiency and maintainability of SAS® programs. We are taught to write code that minimizes machine and programmer resources. Unfortunately, easily maintained code requires fewer programmers, and fewer programmers means pink slips and less job security. In these troubled times, programmers need to be able to maximize their indispensability. They need to become a specialist; a *Job Security Specialist*.

Job Security Specialists can protect themselves and their families by applying the tips and techniques discussed within this paper. The programmer will be advised when to apply the techniques, and whether the use of the techniques should be subtle or gross.

Techniques will cover programming style, editing style, naming conventions, as well as, statements to use and to avoid. You will learn to blur data steps, make non-assigning assignment statements, re-write functions, take advantage of obscure options, use language 'features' to full effect, and in general write code that not even you will be able to figure out how or why it works.

With understanding comes the first steps in becoming a Job Security Specialist.

KEY WORDS

Job Security, Programming Techniques, Style, Options, Compiled Code, Program Editor

INTRODUCTION

General rules for writing SAS programs have been suggested and promoted for a number of years. When followed, the suggested rules create efficient programs that are easily maintained by either the author or by other programmers. Obviously, the primary beneficiaries of well written code are the stockholders of the company that runs and maintains the programs. In a well ordered and polite society the programmer also receives benefits such as an hourly wage, self satisfaction and an occasional pat on the back.

Since well written code is easy to maintain, flaws in the logic or bugs in the code can be easily ferreted out using the documentation and by understanding the appropriate section of the program. Unfortunately this means that the original programmer may not even be necessary to the process; indeed the original programmer can be freed to work on other tasks that require the special talents of a programmer. However as you carefully code your programs, keep in mind that in general, easily maintained programs require fewer programmers and one of the fewer programmers could be you.

Fortunately it is possible to write programs that no one else could possibly maintain. Programs can be written that produce results that cannot be predicted from either a quick or even fairly careful inspection of

the code. Once you know these techniques and have learned to properly apply them, your job will be secure for as long as your programs are in use.

Although it is possible to use the described techniques to write a totally indecipherable program that works, often the subtle application of only one or two techniques in an otherwise ordinary program, will achieve the same results. The selection of a technique and its application to the program is an art form that can be achieved with practice and perseverance.

For ease of use and reference the techniques described in this paper have been grouped into general and specific techniques. As you familiarize yourself with these techniques, and as you put them into practice, you will be on your way to becoming a Job Security Specialist.

PROGRAMMING STYLE

Programming style refers to the general approach that a programmer takes when designing and coding programs. Although most programmers develop their own unique programming style, specific guidelines are often imposed by the employer or the client. In the absence of specific guidelines, consider incorporating some of the following.

✱ Designing the Coding Design

Think about how you design your program. Specifically concentrate on the order of steps and their logic.

✔ Avoid the logical separation of tasks, e.g. separate tasks only if it is logical to keep them together.

✔ Avoid structured programming approaches.

✔ Nest function calls whenever possible. Exceeding a depth of three provides added complexity. The following statement only nests three deep and therefore lacks necessary complexity.

```
depth2 = input(substr(station,index(station,'-')+1),3.);
```

✔ Use functions when they are not required.

```
* DATE is a SAS date;
date=mdy(month(date), day(date), year(date));

* NAME never contains a comma;
name=substr(name,index(name,',')+1,length(name));
```

✔ Write code to replace functions entirely. The ABS function becomes:

```
if a < 0 then b = a*-1;
else b = a;
```

✔ Use non-standard statement structures. A statement to average two positive numbers could be written as:

```
ab = (a*(a>0) + b*(b>0))/((a>0)+(b>0));
```

✔ Nest macro calls and definitions. The addition of macros that call macros that define macros that call

macros will always add a nice touch.

✔ When using macro variables the use of %LOCAL and %GLOBAL definitions allows you create symbolic variables that have more than one definition at any given time. The following macros print two different values for &AA. Resetting &AA in the macro %INSIDE does not change its value in the macro %OUTSIDE.

```
%macro inside(aa);
  %put inside &aa;
%mend inside;

%macro outside;
  %let aa = 5;
  %inside(3)
  %put outside &aa;
%mend outside;
%outside
```

The LOG shows:

```
inside 3
outside 5
```

which clearly shows that &AA had two different values at the same time.

✔ Maximize the number of steps and lines of code. The PROC SQL on the left is too compact and should be rewritten.

Poor Job Security Technique	Improved Job Security Technique

`proc sql;`
`create table report as`
`select * from sales`
`having saleprce gt`
`mean(saleprce)`
`group by region;`
	`proc sort data=sales;`
	`by region;`
	
	`proc summary data=sales nway;`
	`by region;`
	`var saleprce;`
	`output out=stats`
	`mean=meansale;`
	
	`data report;`
	`merge stats sales;`
	`by region;`
	`if saleprce gt meansale;`

✱ Assume the Assumptions of Others

Most programmers make assumptions about the code they are reading; the objective for the Job Security Specialist is to key in on those assumptions.

✔ Use LENGTH to assign the same variable different lengths in different data sets. This is especially useful if that variable is used in the BY statement during a MERGE.

More subtle technique (and subtle is good) is to define the length of the variables without using the LENGTH statement.

✔ Variables with the same names could be numeric in one data set and character in another. This is most useful when the variables are flags that take on the values of numbers.

✱ Eliminate, Hide (Lose) Source Code

If they cannot find your program they cannot 'fix' it. Code that is compiled no longer depends on the original source code (except when modifications are required) and this give us an opportunity to hide the program.

✔ After compilation eliminate, rename, or change:

- the SCL source code used with AF or FSP applications
- compiled DATA steps
- compiled stored macros
- DATA step views
- SQL views (although the source can still be recovered by using the DESCRIBE option).

✔ When editing SCL for SAS/AF® or SAS/FSP® applications change the color of the code to match the background color of the editor.

✱ Use the Data Step Effectively

The DATA step offers a number of opportunities to make our programs more effective.

✔ Use multiple steps when one would suffice.

The data set TWO should not be created in a single step.

Poor Job Security Technique	Improved Job Security Technique

`data two;`
`set master;`
`actual=saleprce+tax;`
`profit = actual-budget;`
	`data one;`
	`set master;`
	`actual=saleprce+tax;`
	`data two;`
	`set one;`
	`profit = actual-budget;`

The following PROC SORT should be rewritten using a completely unnecessary DATA step.

Poor Job Security Technique	Improved Job Security Technique

`proc sort data=old out=new;`
`by date;`
	`data new;`
	`set old;`
	`proc sort data=new;`
	`by date;`

✔ Create data sets that are never used or are used unnecessarily. The DATA step that creates them should be complex.

✔ Use implicit naming rules for data sets. This type of code is also very susceptible to minor problems that can cause major crashes.

```
data;set master;
actual=saleprce+tax;
proc print;
data;set;
profit=actual-budget;
proc means noprint;
output;
proc print;
run;
```

✔ Data set names can be reused thus preventing another programmer from getting overly familiar with the variables that they contain. The previous example becomes:

```
data temp;set master;
actual=saleprce+tax;
proc print data=temp;
data temp;set temp;
profit=actual-budget;
proc means data=temp noprint;
output out=temp;
proc print data=temp;
run;
```

✱ Using System Options

Several system options provide special programming opportunities for the Job Security Specialist.

✔ Debugging aids can be turned off by using NOSOURCE, NOSOURCE2, NONOTES, and NOLABEL.

✔ When turning off debugging aids, prevent full discovery by turning them off in several different places.

✔ When using macros, the use of the macro debugging options MPRINT, MLOGIC, SYMBOLGEN should be avoided.

✔ Some options remove your ability to do things that we take for granted *e.g.* NOMACRO.

✔ You can control the number of observations in a data set using the OBS= option. The FIRSTOBS= and LASTOBS=options tell SAS when to start and stop processing SAS data sets. Change these options and don't tell.

✔ Reroute the WORK or temporary data sets to another location by using the USER= option. In the following DATA step NEW is actually SASUSER.NEW.

```
options user=sasuser;
data new;
set project.master;
```

As an added bonus NEW will not be erased at the end of the SAS job. This can cause disk and clutter problems.

Changing the USER= option several times within a program, makes "WORK" files a bit tough to find.

Even if you do not change the USER= system option, if you create a *libref* of USER, single level data set names will automatically no longer be written to work. In the following Windows step NEW is written to the ABC directory.

```
library user 'c:\abc';
data new;
set project.master;
```

✔ The ERRORABEND option is designed for batch programming, and when an error occurs, the job immediately will abend. When used in an interactive session, the slightest error causes the end of the session and the LOG will not even be available to help determine why the session ended.

✔ Control perceptions for the use of two digit years set YEARCUTOFF=1800. The variable YEAR in the following step will have a value of 1898.

```
* Hide this option statement;
options yearcutoff=1800;

data a;
date = '23mar98'd;
year = year(date);
```

✔ The S= option limits the number of columns read from the program source. Only the first 10 columns are used in the following DATA step.

```
options s=10;
data new;
set olddata
    master
    adj;
profit =
   sales + tax;
cnt+1;
```

The LOG will show (unless you use the recommended NOSOURCE option) that the data set OLDDAT was used instead of OLDDATA and that the variable TAX is never used. Also the SUM statement (CNT+1) becomes part of the assignment statement that is used to create PROFIT. Effectively the code becomes:

```
options s=10;
data new;
set olddat
    master
    adj;
profit =
   sales +
cnt+1;
```

Unfortunately in SAS9.1 there are some problems with the S= option and it does not work correctly from the Enhanced Editor (it works ok in the old and recommended Program Editor). Hopefully this problem will be corrected in SAS9.2 or we may lose this valuable tool.

✔ The CAPS option can be used to change how literal strings are compared. The data set NEW will have zero observations.

```
options nocaps;
data old;
  x='a';
options caps;
data new;
  set old;
  if x='a';
  run;
```

EDITING STYLE

Most programmers will develop a style that they use to code their programs. Although there is merit to the argument that having no style is the best style, there are also specific techniques that you should consider.

✱ Code Layout

The layout of the program as seen by the programmer is determined by the programmer's editing style. Once again, in the absence of specified guidelines, consider these to improve the unreadableness of your code. The following DATA step can be fixed using several different techniques.

```
data sasclass.biomass;
infile rawdat missover;
input  @1 STATION $
       @12 DATE DATE7.
       @20 BMPOLY
       @25 BMCRUS
       @31 BMMOL
       @36 BMOTHR
       @41 BMTOTL ;
format date date7.;
label BMCRUS = 'CRUSTACEAN BIOMASS'
      BMMOL  = 'MOLLUSC BIOMASS'
      BMOTHR = 'OTHER BIOMASS'
      BMPOLY = 'POLYCHAETE BIOMASS'
      BMTOTL = 'TOTAL BIOMASS'
      DATE   = 'DATE'
      STATION  = 'STATION ID';
run;
```

It is quite apparent that this programmer is not only NOT a Job Security Specialist, but he is probably being paid by the hour. Let's fix this DATA step.

✔ Never indent.

```
data sasclass.biomass;
infile rawdat missover;
input  @1 STATION $
@12 DATE DATE7.
@20 BMPOLY
@25 BMCRUS
@31 BMMOL
@36 BMOTHR
@41 BMTOTL ;
format date date7.;
```

```
label BMCRUS = 'CRUSTACEAN BIOMASS'
BMMOL  = 'MOLLUSC BIOMASS'
BMOTHR = 'OTHER BIOMASS'
BMPOLY = 'POLYCHAETE BIOMASS'
BMTOTL = 'TOTAL BIOMASS'
DATE   = 'DATE'
STATION  = 'STATION ID';
run;
```

✔ Use multiple statements per logical line.

✔ Break statements in the middle.

```
data sasclass.biomass;infile rawdat
missover;
input  @1 STATION $ @12 DATE DATE7.
@20 BMPOLY @25 BMCRUS @31 BMMOL @36
BMOTHR @41 BMTOTL ; format date
date7.;label
BMCRUS = 'CRUSTACEAN BIOMASS'
BMMOL  = 'MOLLUSC BIOMASS'
BMOTHR = 'OTHER BIOMASS'
BMPOLY = 'POLYCHAETE BIOMASS'
BMTOTL = 'TOTAL BIOMASS'
DATE   = 'DATE'
STATION  = 'STATION ID';
run;
```

As an added bonus notice that the LABEL assignments look a bit like assignment statements.

✔ The TextFlow option is still available in the old Program Editor. Use it to reform your code.

```
data sasclass.biomass;infile rawdat
missover; input @1 STATION $ @12
DATE DATE7.  @20 BMPOLY @25 BMCRUS
@31 BMMOL @36 BMOTHR @41 BMTOTL
; format date date7.;label BMCRUS =
 'CRUSTACEAN BIOMASS' BMMOL =
'MOLLUSC BIOMASS' BMOTHR =
'OTHER BIOMASS' BMPOLY =        'POLYCHAETE
BIOMASS' BMTOTL = 'TOTAL BIOMASS'
DATE = 'DATE' STATION =
'STATION ID'; run;
```

✔ You can even reform your code to make pictures. The company logo would be a logical choice.

```
                  data
           sasclass.biomass;
      infile     cards   missover;
   input @1                  STATION $
   @12 DATE                     DATE7.
  @20                           BMPOLY
 @25         BMCRUS    @31        BMMOL
 @36         BMOTHR    @41       BMTOTL
 ;                               format
  date                           date7.
    ;label                    BMCRUS=
         'CRUSTACEAN BIOMASS'
                BMMOL=
'MOLLUSC BIOMASS' BMOTHR='OTHER BIOMASS'
BMPOLY=  'POLYCHAETE BIOMASS'   BMTOTL=
           'TOTAL BIOMASS'
              DATE='DATE'
               STATION
                  =
            'STATION ID';
                run;
```

✱ Use Editor Columns > 80

Occasionally use columns > 80 in the editor. Actually you may need to use a larger number of columns depending on the font, screen size, and screen resolution, but let's use 80 as a guideline.

✔ Place key variables out of sight.

```
                                  80
data newdata;
set olddata (drop=name          | fname
    address city state);        |
```

✔ Use the asterisk to comment out key formulas or statements.

```
data new; set old;              | *
       wt = wt/2.2;             |
```

✔ Place special equations entirely out of sight.

```
data new; set old;              | mysalary = mysalary*2;
```

✔ Place special equations partially out of sight.

```
data new; set old;              |
degc = (degf                    | +5
          -32)*5/9;             |
```

✱ Judicious Use of Comments

As you might imagine the Job Security Specialist tends to avoid the use of comments, however comments can be useful when they are used correctly and with discretion.

✔ The following comments contain an executable PROC MEANS.

```
* The comments in this section do more ;
* than it seems                        ;
*                                      ;
* modify data to prep for; proc means  ;
* after adjusting the data using       ;
* the; var for weight                  ;
```

✔ The /* ... */ style comments can be used to mask or unmask code by taking advantage of the fact that this style of comment cannot be nested.

The first comment is 'accidentally' not completed, thus commenting out the DATA step.

```
/* *****************
* Apply the
* ***very ***
* important adjustment;
data yearly;
set yearly;
income = income*adjust;
run;

/* Plot the adjusted income */
proc gplot data=yearly......
..... code not shown .....
```

An embedded comment can be used to cause a portion of the "removed" code to be executed.

```
/* *****************
REMOVE FOR PRODUCTION
proc print data=big obs=25;
title1 'Test print of BIG';
var company dept mgr /*clerk*/;
data big;
set big;
if name='me' then salary=salary+5;
*END OF REMOVED SECTION;
****************** */
* Next production step;
..... code not shown .....
```

Did you notice that the DATA step embedded in the removed section was not actually removed?

CHOICE OF STATEMENTS

Some statements when used properly have the capacity to add several layers of complexity to a SAS program. Others should be avoided as they tend to reduce confusion or provide the reader with information.

✱ Statements to Avoid

Avoid statements that tend to allow the programmer to retain less mental information.

✔ Always maintain extra variables in the Program Data Vector (PDV). Avoid the use of the KEEP and DROP statements.

✔ When variables must be eliminated from the PDV use the DROP statement. The KEEP statement lets the programmer know what variables remain in the PDV, while the DROP statement only reveals what was eliminated.

✔ Comments are to be avoided unless used in the 'special' ways mentioned in this paper.

✔ RUN and QUIT statements are rarely required, and usually only serve to reduce program complexity by separating steps.

✱ Statements to Use

Several statements and combinations of statements can be used to advantage.

✔ Statement style macros can be used to redefine variable relationships.

```
* hide this macro definition;
%macro sat (name) / stmt;
  set &name;
  wt = wt + 5;
%mend sat;

* Why is the value of WT always 6 in
* the data set NEW?;
data old;wt=1;output;
data new;sat old;
```

✔ Implicit arrays are preferred to explicit arrays.

✔ Define arrays using the names of other arrays. This technique was used on purpose prior to the advent of multi-dimensioned explicit arrays. Consider the following three implicit arrays.

```
array a (I) wt ht;
array b (I) xwt xht;
array c (j) a b;
x = c;
```

For I=1 and J=2 the variable X will be assigned the value contained in XWT.

✔ The GOTO and %GOTO statements can be used to breakup program structure; use them liberally.

✔ The label statement can be disguised.
SAS statement key words make excellent labels; consider:

```
do: I=1;
```

Labels can be hidden in comments.

```
* did you notice that, this comment
* contains a; label:
```

✔ Data steps with multiple SET statements or better yet SET and MERGE statements add complexity quickly, but not quietly.

✔ Macro quoting functions can be used to prevent the resolution of macro variables resulting in FALSE comparisons that are obviously TRUE. The macro %DOIT quotes &CITY so that it cannot be resolved.

```
%macro doit(city);
  %put &city;
  %* Hide the following statement;
  %let city=%nrstr(&city);
  %put &city;
  %if &city = LA %then  %put CITY is LOS ANGELES;
  %else %put city is not LA;
%mend doit;
```

When the macro is called with LA as the parameter:

```
%doit(LA)
```

The LOG will show:

```
LA
&city
city is not LA
```

&CITY is not resolved so it will NEVER be equal to LA.

NAMING CONVENTIONS

Naming conventions are perhaps the most useful tool in the arsenal of the Job Security Specialist. A novice may think that no naming conventions or the random selection of names will create the most secure program. In actuality there are a number of subtle and not so subtle techniques which can be applied by the sophisticated programmer. These rules can be applied to both variables and data sets, and come from the schools of confusion, misdirection, and inconsistency.

✱ Confusion

Create names that have no mental references or that make memorization difficult.

✔ Use meaningful names during the debugging process, then when the program is working use the editor CHANGE command to convert variable names:

```
===> c 'age' 'qwrtxzqr' all
```

✔ Use as many digits in your names as is possible.

✔ Avoid the use of vowels, include subtle variations.

```
QWRTXZQR, QWRTZXQR, QWRZTXQR
```

✔ Some letters go well together (the Courier New font is helpful here).

```
H & I   HHHIIHIH HHIHIHIH

V & W   WVWVWVVW WVWWVWVW
```

```
l & 1   testnum1 testnuml
(number 1 and lower case L)

0 & O   test0001 testOOO1
(number zero and letter O)

Z & 2   QWRTZXQR QWRT2XQR
```

✔ SAS statement keywords are not reserved. The following DATA step does NOT contain a DO loop.

```
DATA SET; SET DATA;
DO = 5+ TO -15;
```

In SAS 9 you will get an error in the LOG if you use SET as a data set name. To avoid the elimination of this technique make sure that you set the system option so that DATASTMTCHK=NONE.

✔ Starting with SAS Version 7 the 8 character restriction on names was lifted. Now we can use up to 32 characters - this will give the Job Security expert additional opportunities to utilize confusion. Think of the possibilities:

```
VWVWVVVWVWVWWWWVVVVVWVVVVWVWWWVW
VWVWVVVWVWVWWWVVWVVVWVVVVWVWWWVW
```

Consider creating variable names that tell a story - with subtle variations.

```
ThisVariableStoresThePatientHeight
ThisVariableStoresThePateintHeight
```

✔ The VALIDVARNAME=ANY option allows variable names to contain non-standard characters. Two numeric variables are created in NEW.

```
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
```

✱Misdirection

Normally we would want to avoid names with obvious meaning, however when the name used for a variable or data set has an obvious (implied) meaning and is then used for something else entirely, the reader may be caught unawares.

✔ Obvious names work well for other purposes.

```
SEX    ===> number of fish caught

WEIGHT ===> height of patient

INCHES ===> height in centimeters
```

✔ Placement of observations and data set names.

```
IF SEX = 'MALES' THEN OUTPUT FEMALES;
```

✔ The LABEL statement can be used to provide information to the user. For instance consider the variable SEX which contains the number of fish caught, then:

```
LABEL sex = 'Sex of the Patient';
```

✔ Users of WINDOWS may wish to start SAS sessions using a WORD icon.

✱ Inconsistency

Subtle inconsistencies are very difficult to detect and can be very useful, especially in a program that has what seem to be clearly defined parameters.

✔ YES/NO variables should take on the values of YES=0 and NO=1 (of course never Y & N), except somewhere for some variable that has YES=1 and NO=0.

A useful variation has ANSWER='N' when the response is YES and ANSWER='Y' when the response is NO.

✔ Variable and data set names should not have unique definitions throughout the program.

✔ Variables should on occasion disappear and reappear later with different definitions.

BLURRING THE DATA STEP

The SAS supervisor recognizes the end of steps by detecting a RUN, QUIT, or the start of the next step. We already know not to use the RUN or QUIT, so all we need to do to blur two steps into one is to hide the start of the second step.

✱ Using Comments

Once in a great while the use of comments can be forgiven. We have seen a couple of examples already; here are a few more.

✔ The data set NEW in this example will have the variable `x=zzstuff` as read from the data set GUDSTUFF. Notice that the second comment has no semicolon, hence the data set SECOND is never replaced. In the single DATA step below the value of Y in OLD is effectively never used.

```
* start of the step;
data new ; set old;
x = 5* y;

* this starts the second step
data second; set gudstuff;
x= zzstuff;
```

This data step has two SET statements which is usually good for a few laughs all by itself.

Adding a colon instead of a semicolon to close the comment is even harder to find. The second comment in the above example becomes:

```
* this starts the second step:
data second; set gudstuff;
```

✱ Using Unbalanced Quotes Legally

Usually unbalanced quotes cause syntax errors, but when two such strings are close together, and both have missing quotes, they can cancel each other out. This can leave interesting and syntactically correct code.

✔ The unbalanced quote for the variable NAME completely masks the creation of the data set SECOND and the use of the data set GUDSTUFF.

```
* start of the step;
data new;y=5;frankwt=0;x=5*y;
length name $6;
name='weight;
data second;set gudstuff;
*for weight use Franks';
x=frankwt;
proc print;run;
```

In this example the variable X in NEW will always be zero (not 25). Since GUDSTUFF is never read the value of FRANKWT in GUDSTUFF makes no difference.

✔ Unbalanced quotes can also be used in LABEL statements and to mask the end and beginning of macro definitions.

USING THE MACRO LANGUAGE

Because of its potential for complexity and because it generates code, the macro language provides many opportunities for the Job Security Specialist (Carpenter, 2005). In addition to those already mentioned, a couple of the more promising techniques are included here.

✱ Using the %LET in a DATA Step

Since macro language statements are executed before the DATA step is even compiled it is not possible to assign a the value of a DATA step variable value to a macro variable using the %LET statement. This of course should not discourage the Job Security Specialist, because many programmers new to the use of the macro language are unfamiliar with these timing issues. The data set SASHELP.CLASS has values for the variable NAME for each observation.

```
data new;
set sashelp.class;
%let fname = name;
if "&fname" < 'C' then put "Name starts with A or B " name=;
run;
```

Of course the %LET is executed before there are any values assigned to the PDV. This means that the text in the %LET is not seen as a variable name but merely as the letters n-a-m-e. In fact, since the macro variable &FNAME is assigned the text value n-a-m-e, and since letter lowercase n sorts after all uppercase letters, the IF is never true.

✱ Using an Asterisk to Comment a Macro Statement

It is not unusual to comment out a macro statement by using an asterisk style comment. Let's say that the user would like &DSN to contain the value of `mydata`, the following %LET statements will do the trick since the first and last %LET statements have been commented out.

```
*%let dsn = clinics;
%let dsn = mydata;
*%let dsn = testdat;
```

This works fine in open code, but something odd (also a code word for 'Job Security Opportunity') happens when these same three statements are included inside of a macro definition.

```
%macro testit;
   *%let dsn = clinics;
   %let dsn = mydata;
   *%let dsn = testdat;
   %put dsn is &dsn;
%mend testit;
%testit
```

The %PUT statement shows that the value of &DSN is `testdat`. Even though the Enhanced Editor marks the lines to be commented in green, all three %LET statements are executed (the final definition is the one that sticks). The * are then applied (after the %LET and %PUT have executed). As an aside, this could have other Job Security ramifications as the call to %TESTIT will resolve to a pair of asterisks (**).

ON THE ROUGH SIDE

This section contains an eclectic set of what may be rather extreme measures and should not be used by those with gentle dispositions or a sense of professional integrity.

✱ Learning More (Than the next guy)

Keeping current in the Job Security industry is difficult. Consider these additional sources of information.

✔ Virgile (1996) discusses the behavior of SAS under interesting conditions. All you need is some imagination.

✔ You may want to use the SASNOTES to find system 'features' that can be exploited. An example might be the PROC SORT option NODUPLICATES, which under some circumstances does not create the data set with the anticipated subset of the observations.

✱ In the DATA Step

✔ Assignment statements that change the values of BY variables in a merge can be used to promote interesting combinations of the resulting data.

✔ When merging data sets, variables that are in both incoming data sets (but not on the BY statement) can have interesting properties especially if the number of observations within the BY group is different for each data set.

✔ The POINT= and NOBS= options on the SET statement do not perform as you might expect when the

incoming data set contains deleted observations.

```
data a;
do i = 1 to 5;
output;
end;
run;

* Use FSEDIT to delete ;
* the second obs;
* (I=2);
proc fsedit data=a;
run;

data b;
do point=1 to nobs;
  set a point=point nobs=nobs;
  output;
end;
stop;
run;

proc print data=b;
title 'Obs = 2 was deleted';
run;
```

Assuming that observation #2 was deleted in the FSEDIT step, the OUTPUT window shows:

```
OBS     I
 1      1
 2      1
 3      3
 4      4
 5      5
```

Notice that, although it was deleted, OBS 2 was still in the data set A and is read into the data set B when the POINT= option is used. When this happens the variable I is incorrectly assigned a value of 1.

✱ Using AUTOEXEC.SAS and the Configuration File

Since many programmers do not use (or even know about) the AUTOEXEC.SAS program and the configuration file (named SASV9.CFG in SAS9), any options and code fragments that they contain may remain undetected for some time. Do I detect a Job Security Technique?

These may range in severity from what are merely irritants, such as the use of the ERRORABEND option which was mentioned above, to extreme measures that will halt further processing. A couple of the latter are shown below. Remember subtle is often most powerful. Folks must by necessity look harder when things are completely broken.

✔ The most hidden place to set the ERRORABEND option is in SASV9.CFG.

✔ The ENDSAS and ABORT statements can be conditionally executed and if they appear in AUTOEXEC.SAS, SAS execution will terminate before the user's program even starts.

✔ In the section on using comments we showed how to use the /* to inadvertently hide code. If the last

line of the AUTOEXEC.SAS is a /* all of the submitted code will be commented out until the end of the first /* ...*/ style comment.

A similar result can be achieved by placing the statement %MACRO DUMMY; in the AUTOEXEC.SAS. This will exclude all code until either a %MEND; or %MEND DUMMY; is encountered.

✔ The AUTOEXEC.SAS can also be used to house %INCLUDE statements that bring in and execute code that twists things a bit. Remember to use the NOSOURCE and NOSOURCE2 system options.

✱ Operating System Specifics

Each operation system has a few commands and options that are specific to that OS. These are found in the Companion for that system and are usually less well known.

✔ GENERAL

Write your own procedures using SAS/TOOLKIT®. Remember no documentation!

Use operating specific printer control characters to reset fonts or other printer characteristics.

✔ WINDOWS

The icons used to start SAS have associated properties that can be used to assign system options and to select specific AUTOEXEC.SAS and configuration files. In addition, many SAS options may be overridden in icon parameters where they are well hidden from casual observers.

Access and use external DLLs which are often complex and wonderfully obscure.

✔ VMS

The CONCUR engine allows shared access to data sets. Interesting things can happen when there are multiple updates to a data set at the same time.

✱ Special Options

As a final thought for this section, the Job Security Specialist might like to consider using some other less well known system options - we'll leave it as an exercise for the reader to decide how best to apply them.

✔ DKRICOND=NOWARN

Suppresses error message when variables on DROP, KEEP, and RENAME statements are missing from an input data set.

✔ NOREPLACE

Prevents any permanent data set from being replaced.

✔ NOWORKINIT/NOWORKTERM

Applied at invocation, these options can be used to suppress cleanup of the WORK library on the initialization/termination of SAS session.

✱ Fi the Doo

If you have read this far it is likely that you are hard core enough to become a true Job Security Specialist. The password for membership in the Society of Job Security Specialists, ‘Fi the Doo’, was coined by Tony

Payne and comes from the following example.

Each of the following data steps perform identically and each works without error. Can you find the password?

```
data new;
  set sashelp.class;
  file log;
  if age>12 then do;
    output;
    put _all_;
  end;
run;
```

```
date new;
  sent sashelp.class;
  fill log;
  fi age>12 the doo;
    outputNothing;
    putting _all_;
  endit;
runt;
```

SUMMARY

In reality there is a constant demand for skilled SAS programmers. Just knowing how to write good, clean, and tight SAS code provides a high level of job security. Obviously, the techniques discussed in this paper are to be avoided; however, they have all been encountered in actual programs. Usually they were the result of accidents and ignorance, but no matter the source, they still caused problems. Being aware of the potential of these types of problems is a major step in the direction of writing better code.

ACKNOWLEDGMENTS

Since this paper was first presented in 1993, a number of SAS users have made suggestions and contributions to this topic. Many of these tips have been included in this extended paper. For some reason many of these contributors have asked to remain anonymous. Those who were prepared to be named include Peter Crawford, John Nicholls, and Dave Smith. Tony Payne contributed a great deal as a coauthor on earlier versions of this paper, and I especially appreciate his style and sense of humor. As always I welcome further contributions on Job Security concepts.

ABOUT THE AUTHOR

Art Carpenter's publications list includes four books, and numerous papers and posters presented at SUGI, SAS Global Forum, and other user group conferences. Art has been using SAS® since 1977 and has served in various leadership positions in local, regional, national, and international user groups. He is a SAS Certified Advanced Programmer™ and through California Occidental Consultants he teaches SAS courses and provides contract SAS programming support nationwide.

AUTHOR CONTACT

Arthur L. Carpenter
California Occidental Consultants
10606 Ketch Circle
Anchorage, AK 99515

(907) 865-9167
art@caloxy.com
www.caloxy.com

ACKNOWLEDGMENTS

All the techniques discussed in this paper have been field tested by various SAS programming professionals including the authors of this and the referenced papers. I would express my particular appreciation to Tony Payne for his invaluable contributions to the 'Job Security' techniques.

REFERENCES

The original "Job Security" paper won a "Best Contributed Paper" award at SUGI 18 (Carpenter, 1993). During the subsequent years, *variations of the theme* have been presented at SUGI, and SEUGI 15, WUSS, NESUG, PNWSUG, MWSUG, SUGISA (South Africa), PharmaSUG, SANDS, and Views (UK). Tony Payne joined the Job Security team as a co-author for two of the following papers.

Carpenter, Arthur L., 2004, *Carpenter's Complete Guide to the SAS® Macro Language, 2nd Edition*, SAS Institute Inc., Cary NC.

Carpenter, Arthur L., 2005, "Job Security: Using the SAS® Macro Language to Full Advantage", proceedings of the Pharmaceutical SAS User Group Conference (PharmaSUG), 2005, Cary, NC: SAS Institute Inc., paper TT05. Also published in the proceedings of the 13th Annual Western Users of SAS Software, Inc. Users Group Conference (WUSS), Cary, NC: SAS Institute Inc., paper ESS_Job_Security_using.

Delaney, Kevin P. Delaney and Arthur L. Carpenter, 2004, "SAS® Macro: Symbols of Frustration? %Let us help! A Guide to Debugging Macros", Proceedings of the Twenty-ninth SAS User Group International Conference (SUGI), 2004, Cary, NC: SAS Institute Inc., paper 128-29, also in the proceedings of the Mid West SAS User Group Conference (MWSUG), 2005, Cary, NC: SAS Institute Inc.

Carpenter, Arthur L. and Tony Payne, 2000, A Bit More on Job Security: Long Names and Other V8 Tips, presented at the 8th Western Users of SAS Software Conference (September, 2000), 26th SAS User's Group International, SUGI, meetings (April, 2001), and at the Pharmaceutical SAS User Group meetings, PharmaSUG, (May 2001). The paper was published in the proceedings for each of these conferences.

Carpenter, Arthur L. and Tony Payne, 1998, 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.

Carpenter, Arthur L., 1996, "Programming For Job Security: Tips and techniques to Maximize Your Indispensability", presented at the Twenty-first Annual SAS Users Group International Conference and published in the conference proceedings.

Carpenter, Arthur L., 1993, Programming for Job Security: Tips and Techniques to Maximize Your Indispensability, selected as the best contributed paper, 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.

Virgile, Robert, 1996, *An Array of Challenges - Test Your SAS® Skills*, Cary, NC:,SAS Institute Inc., 174 pp.

TRADEMARK INFORMATION