What NOT to Code

Bob Newman, Amadeus Software Limited

I racked my brains for days, trying to think of a SAS topic on which I could truly claim expert knowledge. Finally it came to me: *writing bad code.*

Not all the examples in this paper are my own. Colleagues at Amadeus have contributed generously, but wish to remain anonymous. One or two Amadeus clients may have have contributed as well, though without realising. They will definitely remain anonymous too.

You may laugh or sneer at these examples, but let he (or she) who is extirely without sin cast the first stone!

TOO MANY DATASTEPS

Here's a sadly typical bit of code:

```
data some.stuff;
    set raw.material(keep=custcode limit balance n things);
    if substr(compress(custcode),1,4)="1234";
run;
data some stuff;
    set some.stuff;
    flag=0;
    if (500<=limit<=2000)
    then flag=1;
run;
data some.stuff;
    set some.stuff;
    margin=limit-balance;
run;
```

One of the rules here seems to be "no more than 4 or 5 lines per datastep". Every new idea, or change to the program, has been coded as a whole new datastep. It would be just as easy, and far more efficient, to do it in one:

```
data some.stuff;
    set raw.material(keep=custcode limit balance n things);
    if substr(compress(custcode),1,4)="1234";
    flag=(500<=limit<=2000);
    margin=limit-balance;
run;
```

Fewer lines, so easier to understand as well as more efficient.

Note the nifty way of setting the value of the flag.

COPYING A DATASET

How many times have we seen (or written):

```
data new.stuff;
            set old.stuff;
        run;
```

While not actually wrong, this is not the best way of doing it. It's better to use PROC COPY e.g.

```
proc copy in=old out=new;
    select st: ;
run;
```

(We take the opportunity of dropping in a reminder about the ":" notation - this step will copy every dataset whose name begins with the string "st").

There's slightly more syntax to remember here, but there's also one major advantage: if the dataset has indexes, those will be copied too (without your even needing to know whether there are any). If you use a datastep, the indexes will need to be re-created.

A related issue in this area, on sites with multiple platforms. Consider the following code, run on a PC:

```
data unixwork.dataset;
    set ntwork.dataset;
run;
```

Here both are RLS libnames, and the dataset is being transferred first from the NT system to our PC, and then on from the PC to the Unix system. We could clog up the network less by something like this:

```
rsubmit NT;
    rsubmit UNIX;
        proc upload in=work.dataset out=work.dataset;
        run;
    endrsubmit;
endrsubmit;
```

RENAMING A DATASET

The following abomination has been sighted in the field:

```
data new;
    set old;
run;

proc datasets lib=work nolist nodetails;
    delete old;
quit;
```

What was wanted was this:

```
proc datasets lib=work;
    change old=new;
run;
```

(PROC DATASETS has a “rename” statement as well, but that’s for renaming variables within a dataset.)

PROC DATASETS also has a way of swopping the names of two datasets, by the way e.g.

```
proc datasets lib=work;
    exchange this=that;
run;
```

Renaming variables

```
data new;
    set old;
    newvar=oldvar;
    drop oldvar;
run;
```

No, thank you. For one thing there’s a “rename” statement:

```
data new;
    set old;
    rename oldvar=newvar;
run;
```

For another, there’s a “rename” dataset option:

```
data new;
    set old(rename=(oldvar=newvar));
run;
```

But best of all there’s PROC DATASETS:

```
proc datasets lib=work;
    modify old;
    rename oldvar=newvar;
run;
```

Did I mention that this dataset has ten million observations? PROC DATASETS doesn’t need to read any of them.

COUNTING THE OBSERVATIONS IN A DATASET

```
data _null_;
    set file.example nobs=number;
    call symput('total',number);
run;
```

What's wrong with that? Well, nothing, in a way. But this is better:

```
data _null_;
    set file.example nobs=number;
    call symput('total',number);
    stop;
run;
```

Why? Because it only sets the value of the macro variable once. The original version sets it time after time, once for each observation in the dataset.

USING CORRECT ENGLISH PUNCTUATION IN COMMENTS

This is a mistake! (Which is great news for people who can never remember when "its" needs an apostrophe and when it doesn't.)

A SAS programmer must not use semicolons or apostrophes in comments. A semicolon can terminate a comment prematurely. An apostrophe can be interpreted as an unmatched quote, and cause all kinds of trouble.

(War story: I once wrote some code which worked so well I turned it into a macro. The macro was so useful I put into my site's main macro library. It worked perfectly well there for about a year. Then someone used the macro in a block of SAS code submitted from SCL, and it blew up because of an "unmatched quote" in a comment - which hadn't seemed to matter up till then!)

THE LAG FUNCTION

This one is notorious, and not all that easy to get your head around.

The danger is that you might think "lag(a)" means "the value of variable 'a' in the previous observation". It doesn't. It means "the value variable 'a' had last time we executed this particular line of code that uses the 'lag' function".

Consider the following code:

```
data dogs;
        set temp;
        if a=lag(a)
        then
           if b=lag(b)
           then
            c=100;
    run;
```

Presumably this is intended to set c to 100 if and only a and b both have the same values as in the preceding observation. If so, it's wrong. Here's why:

a	b	a=lag(a)?	lag(b)	b=lag(b)?	c
1	10	N			
1	***10***	Y	.	N	.
2	20	N			
2	***20***	Y	10	N	.
2	***20***	Y	20	Y	100
3	47	N			
3	20	Y	20	Y	100

Look at that last line. We've flagged c=100 when the value of a is the same as in the previous record, but the value of b is different. This is unlikely to be what we really wanted.

The simplest and most reliable way to avoid this kind of error is not to use “lag” at all. The next best is to evaluate right at the beginning of the datastep all the lag functions that could conceivably be of interest e.g.

```
data dogs;
    set temp;
    drop lag: ;
    laga=lag(a);
    lagb=lag(b);
    if a=laga
    then
    .
    .
    .
```

If you then always use the variables laga and lagb rather than calling the lag function, all will be well.

CONDITIONALS IN MACROS

Here’s an old favourite of mine:

```
%if (a = b) %then %sysexec rm oldfile.txt;
```

This is part of a macro definition. Conditionally, we are issuing a Unix command to delete a file. The trouble is, we have left off a semicolon. We needed to say:

```
%if (a = b) %then %sysexec rm oldfile.txt;;
```

One semicolon is needed to terminate the %if/%then construction, and another to terminate the %sysexec command.

With only one semicolon, it is the %sysexec command that remains unterminated, and so our next line of code is going to be interpreted as the names of more files to be deleted. (Better hope it doesn’t contain an asterisk anywhere!)

Moral: always use %do/%end, even if you’re only generating one line e.g.

```
%if (a = b)
%then
    %do;
        %sysexec rm oldfile.txt;
    %end;
```

SORTING HUGE DATASETS

```
data all;
    set a1 a2 a3;
run;

proc sort data=all;
    by varlist;
run;
```

Nothing drastically wrong with that. But suppose the “all” dataset is 30GB? The amount of workspace needed for the sort is then going to be about 75GB. This may not be available. (Even if it is available, it may be downright unsociable.)

What you can do is sort the smaller datasets separately, then interleave the results:

```
proc sort data=a1;
    by varlist;
run;
        (etc)
data all;
    set a1 a2 a3;
    by varlist;
run;
```

This technique is not much more efficient in terms of runtime, but in some cases the saving in workspace can be very worthwhile.

GETTING A LIST OF VARIABLE NAMES

Here's an impressive-looking way of getting a list of variable names into a macro variable:

```
%macro listvars(datain,macrovar);
    proc contents data=&datain noprint out=_vars(keep=name);
    run;
    data _null_;
        set _vars;
        call symput("nvars",_N_);
    run;
    %let list=;
            %do i = 1 %to &nvars;
        data _null_;
            set _vars;
            if _N_ = &i;
            call symput("var", name);
        run;
        %let list=&list &var;
    %end;
    %global &macrovar;
    %let &macrovar = &list;
%mend listvars;
```

But there's a better way, using the power of PROC SQL:

```
%macro listvars(datain,macrovar);
    %global &macrovar;
    proc contents data=&datain noprint out=_vars(keep=name);
    run;
    %let list=;
    proc sql noprint;
        select name into: list separated by ' ' from _vars;
    quit;
    %let &macrovar=&list;
%mend listvars;
```

CONCLUSION

Don't confuse complexity and apparent sophistication with quality and efficiency.

Don't confuse simplicity and ease of coding with quality and efficiency.

There is no substitute for keeping your wits about you!

CONTACT INFORMATION

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

Author Name	Bob Newman
Company:	Amadeus Software Limited
Address:	Mulberry House, 9 Church Green, Witney, Oxon OX28 4AZ
Work Phone:	+44 (0) 1993 848010
Email:	bob.newman@amadeus.co.uk
Web:	www.amadeus.co.uk