SAS (r) Software with Other Tools on a PC Platform: 
Experience in a Clinical Pharmacokinetic Laboratory

Jerold S. Harmatz, New England Medical Center Hospital

ABSTRACT

Irrespective of the platform on which SAS software is executed, personal computers should be appreciated as facilitating the production and running of SAS code. In the assessment of pharmacokinetic and pharmacodynamic drug effects, we confront a diversity of statistical and reporting tasks. Several computing tools, that typically are part and parcel of the “personalization” of personal computers, have proven helpful. In this presentation, we review applications of the command shell, of the editor and of supporting utilities that can be facilely cobbled together in Pascal and C.

INTRODUCTION

We all recognize that SAS software has evolved into a versatile toolkit usable on various computer platforms. Most of us can accomplish pretty much whatever we need to with SAS. It is also true, however, that we tend to become accustomed to a limited set of tools in a given environment. Workgroup procedures and organizational conventions can dictate restrictive guidelines and limitations. As enabling as SAS is, it is ironic that many of us operate in conformance to rather narrow work habits. It is the thesis of this presentation that personal computers and workstations can enhance our computing repertoire, permitting us alternative approaches to the jobs at hand.

The word “personal” in personal computers can be understood quite literally: these machines can and ought to be personalized according to individual workstyles and needs. The PC encourages you to assemble a customized collection of tools. A kind of creative freedom seems natural on a PC. Whether you choose to run your SAS jobs on a mainframe or on a PC is a decision that should not preclude our recognizing how PCs can facilitate data and code preparation as well as the presentation of reports. A myriad of options permit us to tailor-fit jobs to particular functional requirements and to do our work in comfortable ways.

In my clinical pharmacology research laboratory, we have used SAS since the 1972 release, and SAS/PC since 1986. Except for its 30-year longevity our laboratory probably resembles many other research installations which regularly use SAS for scientific and clinical investigation. In this presentation, I will illustrate how a few commonly available PC application programs have proven supportive of our SAS work.

We investigate the pharmacokinetic and pharmacodynamic properties of various pharmaceutical preparations. We examine how drugs, especially benzodiazepines, distribute differently across time and physiological structures, thereby producing different effects in different populations. Most of our work involves small samples of human volunteers. Parallel studies in animal populations complement our program.

In most of our work, multiple assessments across time lead to the construction of many variables relative to the small number of people studied. A given program might require recording data for some 300-2500 variables. Data have to be cleaned, edited and shaped into appropriately sized and consistent sets for analysis. Data step acrobatics frequently are important for the several axis changes necessary to understand different time-within-time questions. Analyses typically include linear and nonlinear regression characterization of quantitated drug and metabolite levels over time, as well as ANOVAs of treatment-induced changes over baseline. Preliminary reports exploit PROC PRINT, the PUT facility as well as PROC PLOT.

FACILITATION OF SAS WORK BY UTILITIES

ON A PERSONAL MACHINE

A cursory description of tools we have found handy in the PC environment will perhaps suggest further applications to other workers. We try to make full use of the command shell and DOS batch processing. Among the language compilers or interpreters commonly available in a PC environment, we generally use Pascal and C. In addition, as a kind of superset of C, we take advantage of a public domain MS-DOS version of ARK [1]. It is also important to recognize that SAS itself can serve as a program generator. Finally, we want to comment on the usefulness of data compression routines, e.g. PKZIP (tm) [2] for data storage and communication.

Command Shell

Among the most personal of instrumental accommodations to the PC is one most readily taken for granted; the customized file directory and command structure, usually called the command shell. Apart from organizational advantages, most shells will allow the user simply to point at a file of
We have used C to write a little supplementary utility (SASALARM.C) which is called by the shell upon completion of SAS processing. Since many of our SAS applications are routine production programs, reviewing SAS LOG files to look for obvious error strings like "ERROR", "INSUFFICIENT", "HAS O OBSERVATIONS", "NO OBSERVATIONS" or "UNINITIALIZED" consumes time and patience. SASALARM.C is not merely a convenience. Similarities across jobs can provoke a kind of "blindness" such that we fail to notice that something is critically wrong given that the unwanted event occurs rarely relative to the frequency of our watching out for it. SASALARM.C will scan the output LOG file looking for problems we might hitherto have been too blinded by familiarity to notice.

When SASALARM.C has made its perusal, it issues distinctive auditory and visual signals to indicate the evident success or failure of the job. We can relegate the running SAS job to the background. In the case of a multitasking operating environment like Microsoft Windows, we can do some other computer work; otherwise, we remain free to attend to non-computer business. The logic of SASALARM.C certainly is extensible to the inclusion of additional admonitory elements and to alternative cueing patterns.

**Programmer's Editor**

For many people, the tool they use most routinely on a personal machine is a spreadsheet. For others, including ourselves, the editor serves as the principal interface. We use a so-called programmer's editor, one marketed under the name of BRIEF (r) [3]. BRIEF has built-in macro facilities help with the kinds of textual changes and structures needed by people who work with code. In addition, such editors permit users to create macros for modifying or automating certain operations.

The BRIEF macro facility provides an OS-interfacing language with a very strong structural and functional resemblance to C. The following BRIEF macros were readily put together and represent something of the diversity of functional applications to SAS processing possible:

- **COMMENT.CM** offers both line and block-oriented keystroke-command commenting and uncommenting, and is particularly suitable for quickly commenting out or restoring sections of SAS code. We are not cost or resource conscious when using SAS/PC, and the ability to include and exclude code so readily encourages welcome experimentation. Furthermore, for some kinds of code, it is possible to use commented blocks as a rough-and-ready kind of version-control.

- **REPSTR.CM** helps build up largely repetitive SAS code. Marked blocks of code are pasted seriatim with each pasting conditional upon a prompted-for alteration in the original. Full regular expression translations can be used. Regular expressions allow facile reference to text patterns by metacharacters and sequence indices as well as by literals. In this code extract (Figure 1), the REPSTR statement for a long list of enzymes has been constructed for a repeated measures design analysis: The lines containing the first enzyme "trypt" (tryptophan) are placed in a buffer. REPSTR then pastes copies of those lines immediately following the original block, and prompts for any text that should be changed; in other words, only the names of the successive enzymes need be entered. A similar facility will be presented in the discussion of AWK.

- **PROOFIN.CM** Sometimes it is convenient quickly to proofread lines of code or data at the terminal, without the aid of hardcopy or of some listener to double-check. This macro sets up a one-line window in the current buffer at the cursor position. It is very easy to keep one's eye on course within the demarcated window; necessary corrections can be made within the window and be immediately effective for the whole file.

MAGSAS.CM combines the functionality of an edit macro with SAS' own MACRO facility. After a SAS MACRO has been run with the MPRINT option, MAGSAS cleans the MPRINTed output so that it can subsequently be used as ordinary SAS program code.

LOGLST.CM Like some other editors, BRIEF has the ability to compile a source program according to the designation of its file extension. BRIEF will move most of itself out of memory leaving room for SAS to execute. Control returns automatically to the editor. LOGLST is automatically called subsequent to the execution of SAS and SASALARM. LOGLST loads the SAS LOG and LST files into the edit buffer to co-reside with the SAS source file. Therefore, having put together a SAS source file, a simple hot-key command will invoke SAS processing, the automatic review of the LOG
file and the loading of the LOG and LST files for browsing or printing within the same edit environment. If it's necessary to make a change to the SAS source and to begin the execution cycle anew, a hot-key command will instantly accomplish that.

Code Generation by Programming Language

Our biochemistry people set up time-serum concentration data the way they have for many years, in a format (Figure 2) originally required for so-called CARD READER entry on an IBM370. We need to generate the SAS code to get listings and log-linear plots for initial review. A Pascal program, FPNSAS reads the CARD READER data and outputs the needed SAS program (Figure 3). Since sometimes 100 or more datasets need to be appended together, such a utility is very convenient.

For oral medications, PROC REG is commonly used to compute the slope of the terminal log-linear phase of plasma concentrations. For each person-drug dataset, the Pascal program SASPDATA reads the original CARD READER dataset and prompts for the data-pairs to be excluded from the PROC REG run. In Figure 4, we see the SAS input code generated by SASPDATA.

SAS can itself effectively generate secondary SAS program code: In studies where time values remain the same across datasets, it is desirable to enter the times only once, and subsequently to record only the concentration levels as we cry to output datasets in CARD READER format. We see GRID.SAS in Figure 5. A row of time values, concluding with a decimal point, is followed by each identifier and its set of concentration values. Figure 6 shows the output datasets.

AWK Source Code Generation

AWK was designed specifically to help construct source code. Several convenient and reasonable assumptions are implicit in the syntax of the language. For example, data types, rather than being declared, exist potentially as both string and floating point values, and are interpreted according to context. AWK is comprised of pattern-action sequences, with syntax and control structure closely resembling that of C.

Consider a study design which has several dependent variable dimensions; each dimension, in turn, represents multiple observations across time, and for each dimension every observation after the first couple needs to be corrected by a baseline comprised of the first two observations. Figure 7 depicts the repetitive SAS code that is needed, in this case just for SEtation, ANX1ety and SADness as three dependent dimensions. This combining of AWK and SAS macro LET statements is well appreciated when, as is usually the case, we have 50 to 100 dimensions. Figure 8 shows the AWK file used to yield this repetitive SAS code. In this example, AWK's work is done before any pattern matching. In other applications, we might exploit the fact that AWK functions can be made conditional on regular expression pattern matching.

File Compression

It is helpful to remember that several compression schemes are widely available on the PC. Most compression needs are met quite well by Katz' PKZIPS. Compressed files usually reside conveniently in archived sets, and so provide organizational benefits. Text files, e.g. SAS code files, can easily achieve about 50% compression. SAS users should note that SAS system files, i.e. LIBRARIES, will often be compressed by over 90%. So we enjoy dramatic space savings and organizational advantages in storage. Furthermore, it follows that we achieve faster file transfers among machines.

SAS Evolving on a PC

SAS Institute is currently invested in providing a common user interface and functionality across various machine platforms. At the same time, PCs are becoming ever more prevalent. Software development and availability will grow in proportion to this expanding user base. Although SAS may remain the focus of our attention, we look forward to a burgeoning of ancillary and utility programs: commercial, public domain, and custom-written with the help of accessible, convenient language implementations.

REFERENCES


Figure 1

```plaintext
input
#1 @ 6 (tryptpre trypt0_5 tryptl_0 tryptl_5
trypt2_0 trypt4_0 trypt6_0 trypt8_0) (5*3.1 +3 3*3.1)
#2 @ 6 (aspartpre aspart0_5 aspart1_0 aspart1_5
aspart2_0 aspart4_0 aspart6_0 aspart8_0) (5*3.1 +3 3*3.1)
#3 @ 6 (asparagpre asparag0_5 asparag1_0 asparag1_5
asparag2_0 asparag4_0 asparag6_0 asparag8_0) (5*3.1 +3 3*3.1)
```

Figure 2

```plaintext
:READ L SCOTT 5448 3 E5
0 0 .5 0 1 .4318 1.5 .2325 2 .1661 2.5 .1661
3 .0664 4 0 6 1.4614 8 0 2 0 0
:READ H SCOTT 5448 4 E5
0 0 .5 .8303 1 .3321 1.5 .6975 2 .8303 2.5 .7307
3 .5314 4 .3325 6 0 8 0 2 0 0
:READ L WILSON 5450 2 E5
0 0 .5 1 .9167 1 .5 .697 2 .5227 2.5 .3485
3 .3485 4 .3135 6 .2788 8 .1742 2 0
:READ H WILSON 5450 4 E5
0 0 .5 3 .1364 1 1.5 1.3591 2 1.15 2.5 1.15
3 .9409 4 .5015 6 .9833 8 .2788
```

137
Figure 3

OPTIONS LS=132 PS=58;
DATA ALL;LENGTH NAME $ 20;NAME='L_SCOTT_5448_3 ';INPUT TIME CONC @@;
CARDS;
  0 0 .5 0 1 .4318 1.5 .2325 2 .1661 2.5 .1661
  3 .9664 4 0 6 1.4614 8 0 24 0
;DATA TRANS;LENGTH NAME $ 20;NAME='H_SCOTT_5448_4 ';INPUT TIME CONC @@;
CARDS;
  0 0 .5 .803 1 .3321 1.5 .6975 2 .803 2.5 .7307
  3 .5314 4 .2325 6 0 8 0 24 0
;PROC APPEND BASE-ALL DATA=TRANS;
DATA TRANS;LENGTH NAME $ 20;NAME='L_SCOTT_5448_3 ';INPUT TIME CONC @@;
CARDS;
  0 0 .5 1.9167 1 .9409 1.5 .697 2 .5227 2.5 .3485
  3 .3485 4 .313 6 .2788 8 .1762 24 0
;PROC APPEND BASE-ALL DATA=TRANS;
DATA TRANS;LENGTH NAME $ 20;NAME='H_SCOTT_5448_4 ';INPUT TIME CONC @@;
CARDS;
  0 0 .5 3.1364 1 1.847 1.5 1.3591 2 1.15 2.5 1.15
  3 9.409 4 .8015 6 .3833 8 .2788
;PROC APPEND BASE-ALL DATA=TRANS;
DATA ALL;SET ALL; IF CONC > 0 THEN LOGCONC=LOG(CONC);PROC PRINT;ID TIME;BY NAME NOTSORTED;
PROC PLOT NOMISS=PLOT LOGCONC*TIME;BY NAME NOTSORTED;RUN;

Figure 4

DATA ALL;LENGTH NAME $ 20;NAME='L_SCOTT_5448_3 ';INPUT TIME CONC @@;
IF TIME LT 2.5 OR TIME GT 6 THEN DELETE;
CARDS;
  0 0 .5 0 1 .4318 1.5 .2325 2 .1661 2.5 .1661
  3 .9664 4 0 6 1.4614 8 0 24 0
;DATA TRANS;LENGTH NAME $ 20;NAME='H_SCOTT_5448_4 ';INPUT TIME CONC @@;
IF TIME LT 1.5 OR TIME GT 4 THEN DELETE;
CARDS;
  0 0 .5 .803 1 .3321 1.5 .6975 2 .803 2.5 .7307
  3 .5314 4 .2325 6 0 8 0 24 0
;PROC APPEND BASE-ALL DATA=TRANS;
DATA TRANS;LENGTH NAME $ 20;NAME='L_SCOTT_5448_3 ';INPUT TIME CONC @@;
IF TIME LT 2.5 OR TIME GT 8 THEN DELETE;
CARDS;
  0 0 .5 1.9167 1 .9409 1.5 .697 2 .5227 2.5 .3485
  3 .3485 4 .313 6 .2788 8 .1762 24 0
;PROC APPEND BASE-ALL DATA=TRANS;
DATA TRANS;LENGTH NAME $ 20;NAME='H_SCOTT_5448_4 ';INPUT TIME CONC @@;
IF TIME LT 2.5 OR TIME GT 4 THEN DELETE;
CARDS;
  0 0 .5 3.1364 1 1.847 1.5 1.3591 2 1.15 2.5 1.15
  3 9.409 4 .8015 6 .3833 8 .2788
;PROC APPEND BASE-ALL DATA=TRANS;
DATA ALL;SET ALL; IF CONC > 0 THEN LOGCONC=LOG(CONC);PROC REG;MODEL LOGCONC TIME;PRINT MODELDATA;BY NAME NOTSORTED;RUN;
Figure 5

/*INPUT TIME ARRAY WITH A PERIOD FOLLOWING LAST TIME, THEN NAME, THEN CONCS, THEN NEW NAME AND CONCS*/

FILENAME OUT "TEMP1.DUM";
DATA ALL;
ARRAY X [*] X1-X40;
ARRAY Y [*] Y1-Y40;
IF N = 1 THEN DO I = 1 TO 40 UNTIL (X[I] = .);
INPUT X[I] @;
IF X[I] = . THEN N = 1 - 1;
END;
RETAIN X1-X40 N;
INPUT NAME $ @;
DO I = 1 TO N;
IF X[I] ME . THEN INPUT Y[I] @;
END;
END=":READ \";L2=" E5\";FILE OUT LS=65;S=\" ";
OUTPUT;
PUT 1 NAME L2;
DO I = 1 TO N; IF Y[I] ME . THEN PUT X[I] S Y[I] S @ ; END;
PUT /;
CARDS;
O .25 .5 .75 1 1.5 2 2.5 3 4 6 8 12 18 24 30 36 48 .
JAB 0 29.4 390 463 503 572 506 638 585 462 366 216 18 98.1 65.9 48.2 27.2 13.4
ADF 0 0 25.4 390 463 503 572 506 638 585 462 366 216 18 98.1 65.9 48.2 27.2 13.4
DOA 0 61.5 705 820 730 564 537 384 369 292 209 135 63.8 48.7 30 19.5 10 .
;
RUN;

Figure 6

:READ JAB E5
0 0 0.25 25.4 0.5 390 0.75 463 1 503 1.5
572 2 506 2.5 638 3 585 4 462 6 366 8 216
12 146 18 61.6 24 29.3 30 22.4 36 12.9

:READ AFD E5
0 0 0.25 0 0.5 121 0.75 305 1 458 1.5 593
2 735 2.5 772 3 612 4 546 6 434 8 303 12
192 18 98.1 24 65.9 30 48.2 36 27.2 48 13.4

:READ DOA E5
0 0 0.25 61.5 0.5 705 0.75 820 1 730 1.5
564 2 337 2.5 384 3 369 4 292 6 209 8 135
12 63.8 18 48.7 24 30 30 19.5 36 10
Figure 7

```plaintext
\$let sed=sed_5 sed1 sedl_5 sed2 sed2_5 sed3 sed4 sed5 sed6 sed8;  
\$let anx=anx_5 anx1 anx1_5 anx2 anx2_5 anx3 anx4 anx5 anx6 anx8;  
\$let sad=sad_5 sad1 sadl_5 sad2 sad2_5 sad3 sad4 sad5 sad6 sad8;  
array sed[10] &sed;  
array anx[10] &anx;  
array sad[10] &sad;  
do i=1 to 10;  
sed[i]=sed[i].mean(of sedpl sedp2);  
anx[i]=anx[i].mean(of anxpl anxp2);  
sad[i]=sad[i].mean(of sadpl sadp2);  
end;  
BEGIN{  
a="sed anx sad";  
ndims=split(a,x," ");  
for(i=1;i<ndims;i++)  
  printf ("%s %s %s %s %s %s %s %s %s %s 
  x[i],x[i],x[i],x[i],x[i],x[i],x[i],x[i],x[i],x[i];  
  for(i=1;i<ndims;i++)  
    printf ("array %8s &amp;\n",x[i],x[i],x[i],x[i],x[i],x[i],x[i],x[i],x[i],x[i]);  
  printf ("do i=1 to 10;\n;  
  for(i=1;i<ndims;i++)  
    printf (" do i=1 to 10;\n;  
    printf (" end; \n)  
```

Figure 8
Comparing File Size to Free Disk Space when Copying DOS Files from within SAS/AF®
Applications in Release 6.03

N. Craig Bickel, Lasley & Hopkins Inc.

Abstract
Applications built for the novice user may often include a front-end to facilitate copying DOS files from disk to disk. This requires comparing the size of the file to the available space on the destination disk. SAS/AF software does not include functions to test either file size or available disk space. This paper presents a method to perform the comparison and create a signal file using (1) an external DOS program (written in C) and (2) SAS/AF Screen Control Language to check the signal file before proceeding with the copy.

Introduction
Building complex systems for users that have no interest in learning the SAS® System or even simple computer operation is always a challenge. The client whose application prompted this paper was very insistent that the system be "idiot-proof". The particular system under development in SAS/AF included a module to copy data from the hard drive on to a floppy disk for transport. As system design and analysis proceeded, keeping the "idiot-proof" requirement in mind, the problem that prompted the writing of this paper was soon presented.

The Problem
To ensure the system met all of the client's needs, especially the "idiot-proof" requirement, I needed to find a way to protect the client from any potentially "threatening" events. In particular, I wanted to provide the client with more than the DOS "insufficient Disk Space" message if there was not enough room on the target disk for the file to be copied.

Overview of the Solution
Because SAS 6.03 does not include functions that return the available space on a disk or the size of a file, making the calculation outside of SAS was the only answer. This presented me with the question of how to let the SAS application know whether to continue with the copy or to notify the user that their target disk is full and to insert a blank disk. The solution arrived at follows:

Step 1: Write a DOS executable program, called from SAS, that compares the size of the file to be moved to the available space on the target disk. If there is enough room on the target disk for the source file to be copied, the program will create a "Signal" file named 'OK2COPY.IT'.

Step 2: Generate the appropriate SAS/AF SCL code to execute the above program and then test to see if 'OK2COPY.IT' was created. If the file was created, proceed with the copy. If it was not created, signal the user to insert a new disk.

The Program Code

CHK2COPY.EXE
CHKH2COPY is the program I developed to compare the size of the file to be copied to the available space on the target disk. The program first deletes 'OK2COPY.IT'. If it already exists. Next, the program checks the amount of available space on the target disk and the size of the file being copied and compares the numbers. If there is room for the file to be copied to the target disk, 'OK2COPY.IT' is not created. Arguments passed to CHKH2COPY (when the program is executed) are the target drive ID (A, B, C, etc.) and the name of the file to be copied. This version of the program prints several messages notifying the user at each step of the program execution, and the results of that step. Figure 1 below depicts the output from CHK2COPY when there is not sufficient space to proceed with the copy. Figure 2 shows the display when there is enough space for the copy to proceed.

Figure 1

Figure 2

The version of this program passed to the client with the finished system did not include the screen I/O seen above so there was no chance for confusion. Executing SAS with the -XWAIT OFF system option also returns the user to SAS immediately following