Paper 20-26

SAS Tips I learnt while at Oxford

By Philip Mason, Wood Street Consultants Ltd., Oxfordshire, England

Abstract

Here is a collection of SAS Tips & Techniques from my 15 years experience as a SAS Consultant. You can read about some of them in my book "In the Know: SAS Tips & Techniques from around the globe", which is available from SAS Institute and amazon.com. Others are from my second book which is currently being finalised, and there are also a few extras.

Debugging Complex Macros

It's possible to write code generated by macros to an external file. The file can't be access until the SAS session has ended. In SAS 6.12 you use the options RESERVEDB1 & MPRINT to achieve this. In version 8 you need to use the options MFILE & MPRINT. You then also need to define a fileref for MPRINT, to which the generated SAS code is written.

```
filename mprint 'c:\macro.sas' ;
```

This technique is very useful for complex macros with many loops and multiple ampersands since you can see the code that is generated. You can then run the code generated through data step debugger to resolve any problems or understand what is happening.

Connect to yourself

This requires SAS/Connect. In SAS version 8 you can use MP Connect to kick off multiple SAS processes from a single SAS session. But in SAS 6.12 that feature is not available. You can achieve a similar effect by running multiple SAS sessions. To do so you need to run the SAS spawner. On my machine I would start the spawner with the following command:

```
"C:\Program                    Files\SAS
Institute\SAS\V8\spawner.exe" -c tcp ╥z
```

I could then start the SAS session by using the following code:

```
%let tcpsec=_prompt_;
options comamid=tcp ;
signon notebook noscript ;
```

This technique is very useful to test out client server code on a single machine, or simply to take advantage of SMP (multiple processor) machines.

Adding operators or functions to macro language

The operator "=:" exists in data step, but not macro language. It is possible to use the macro language to create a macro which carries out the required function and returns a result. The following code achieves this:

```
%macro eq(op1,op2) ;
  %if
%substr(&op1,1,%length(&op2))=&op2
%then
    1 ;
  %else
    0 ;
%mend eq ;
      ┬
%if %eq(abcde,ac) %then
  %put yes 1 ;
%else
  %put no 1 ;
```

Searching a catalog for text

To my knowledge SAS does not provide a tool to search through catalog members for specific text, but there is a way to do this. Using PROC BUILD you can write all the source text from catalog entries to an external file. Then you can use PROC FSLIST to view the file (or notepad/MS Word) to search it. The following macro can be used to do this.

```
%macro catscan(cat,file) ;
  proc build catalog=&cat batch ;
    print source prtfile="&file" ;
  run ;
  dm 'fslist "&file"' fslist ;
%mend catscan ;
```

Making code restartable

Why make code restartable? Restartable code is more robust and saves reprocessing data. To achieve this you need to take note of where you are in the program, at least the last successful part of the code that completed. You then need to save datasets & macro variables as they were at that point in time. The tricky part of this can be saving the macro variables. This can be done by doing a %put _user_ to a dataset or file, and then reading them back in. Alternatively you can copy sashelp.vmacro to a dataset, e.g.

```
proc sql ;
```

```
create table sasuser.macros as
  select name,value from
    sashelp.vmacro
        where scope='GLOBAL' ;
```

Using progress bars in base SAS

Progress bars are fairly easy to implement in SAS/AF however a little harder from base SAS. You can however produce progress bars for use in Data Steps or Macros. These are quite useful to let users know how a long macro is progressing, for instance. This example shows how to do so for a data step, though a similar technique can be used from macro language using the %WINDOW and %DISPLAY statements.

```
data _null_;
  length pct $ 10 ruler progress $ 50 ;
  left=byte(147) ;
  right=byte(147) ;
  ul=byte(149) ;
  ur=byte(151) ;
  ll=byte(155) ;
  lr=byte(159) ;
  ruler=repeat(byte(148),49) ;

bit=byte(164)||byte(164)||byte(164)||by
te(164)||byte(164) ;
  window bar irow=5 rows=9 icolumn=15
columns=64
        #1 @11 ul
           @12 ruler
           @62 ur
        #2 @2 'Progress' color=orange
           @11 left
           @12 progress  color=blue
           @62 right
        #3 @11 ll
           @12 ruler
           @62 lr
        #4 @35 pct ;
  progress=bit ;
  pct=putn(.1,'percent.') ;
  display bar noinput;
  link pause ;
  do i=1 to 9 ;
    progress=repeat(bit,i) ;
    pct=putn(.1+i*.1,'percent.') ;
    display bar noinput;
    link pause ;
  end ;
  stop;
return ;

pause:
  now=time() ;
  do while((now+1)>time()) ;
  end ;
return ;
run;
```

The key points are:

–Define a window shaped like a bar

–Redisplay it each time we have some progress

–Just change value and link to routine to update bar

Some of the important Statements in Code are:

–"Window bar " defines progress bar display window

- Specifies position on screen
- Variable & colour used to display progressing bar

–"display bar noinput" refreshes the bar display and requires no input from user to continue

Windows API calls

You can use the MODULE function to call windows APIs (Application Programming Interfaces). To do so you need to make a SASCBTBL definition in a text file and then assign that to the SASCBTBL fileref. For example:

```
routine    MessageBoxA    module=USER32
minarg=4 maxarg=4
       stackpop=called returns=short;
arg 1 input format=pib4. byvalue;
arg 2 input format=$cstr200.;
arg 3 input format=$cstr200.;
arg 4 input format=pib4. byvalue;
```

Documentation of the Windows APIs can be obtained from books that can be purchased or is also available in the Linux WINE project. That documentation covers "all" APIs and is freely downloadable.

There are also a lot of SASCBTBL definitions defined in:

```
Sampsrc.pcsamp.sascbtbl.source.
```

Some Module Definitions

ExitWindows
GetDiskFreeSpaceA
GetDriveTypeA
GetModuleFileNameA
GetModuleHandleA
GetPrivateProfileIntA

GetPrivateProfileStringA
GetProfileIntA
GetProfileStringA
GetSystemDirectoryA
GetSystemMetrics
GetTempPathA
GetTempFileNameA
GetVersion
GetVersionExA
GetVolumeInformationA
GetWindowsDirectoryA
GetSystemInfo
MessageBeep
MessageBoxA
SwapMouseButton
WritePrivateProfileStringA
WriteProfileStringA
GetWin32sInfo

Mixed data step & SQL code

There are some features available in a data step that are not available in SQL, and visa-versa. However you can have data step code in an SQL statement by creating a view and referring to it from SQL. For example:

```
data out / view=out ;
  set sasuser.houses ;
  if style='CONDO' then
    put 'obs=' _n_ price= ;
run ;
proc sql ;
  create table more as select * from out where price
>100000 ;
quit;
run;
```

This creates a dataset and writes some variable information to the log You can similarly have SQL code within a data step, by using views

Some sort techniques

- A general rule-of-thumb for calculating sort space required is three times the space of the dataset.
- You can simply set Compress=yes on a dataset and then sort it. This can result in time savings, in my example I saved 8%.
- The Tagsort option is very good on large datasets where key is small. In one example I saved 49% of the elapsed time. When the sort key approaches the size of the observation, the effectiveness of this technique decreases and can eventually increase time taken.
- You should almost always use the Noequals option on PROC SORT. This often saves me around 5% of the time. All that this option does is tells SAS not to worry about the order of observations within BY groups. EQUALS causes SAS to keep that order the same as the observations were in the input data. An extra overhead that is virtually never required.
- You can often combine datastep code with a sort by using a VIEW. This generally saves me at least 27% of the time. The next tip explains this further.

Views can move pre-processing into procedures for efficiency

•Inefficient

```
Data test;
  if ┬ flag=1 ;
run;
Proc sort data=test;
  table flag;
run;
```

•Efficient

```
Data test/view=test;
  if ┬ flag=1 ;
run;
Proc sort data=test;
  table flag;
run;
```

Changing data step to a view causes less I/O to be done, since data is read once. The IF condition is applied and the record fed into proc sort directly.

Finding secret SAS options

You can find out the settings of all documented and undocumented SAS options by using the following code:

```
Proc options internal ;
run ;
```

Some of the options that are revealed include

```
BELL                Enables/disables the
warning bell
CDE=H                Display SAS System
information
CTRYDECIMALSEPARATOR=.          Country
specific decimal number separator.
CTRYTHOUSANDSEPARATOR=,         Country
specific thousands number separator.
DEBUGLEVEL=         Controls display of
debug information. There are five
levels (TESTING, NORMAL, DEBUG,
FULLDEBUG, and DEMO).
ENHANCEDEDITOR      Invoke the enhanced
editor at SAS startup
```

Making log available during non-interactive SAS

One of the undocumented options which PROC OPTIONS INTERNAL reveals is:

```
$logflush ┬ closes LOG after each line is written
```

This is very useful for looking at log during non-interactive runs. You usually can't see log until the SAS session finishes. This works in SAS 6.12 as well as version 8, but the option must be invoked at startup of the SAS session.

Put a zip in your pipe

Pipes read live output from programs. They originated on UNIX systems but are also available on Windows. You can define a pipe to a program as in this example,

```
filename testpipe pipe 'pkunzip.exe c:\temp\test -c' ;
```

This uses PKZIP to decompress an archive. The "-c" option sends data to the console, flowing into pipe, able to be read by SAS. The PIPE parameter is required.

One of the major uses of this is to allow the processing of files too large for disk.

```
** Define pipe to get output from PKZIP ;
filename testpipe pipe 'pkunzip.exe c:\tips\colon -c' ;

data report ;
 * Read data from pipe ;
  infile testpipe ;
  input line & $200. ;
 * Skipping PKZIP header ;
  if _n_<=9 then
    delete ;
run ;

proc print data=report ;
run ;
```

Exporting data using ODBC

Data can be made available to users who don't have any SAS experience by using the ODBC interface. Once it is setup you can simply copy SAS datasets to a directory and they will be instantly available to users of ODBC compliant applications such as Microsoft access, excel, etc.

Steps to carry out this process.

- Open the Control Panel and select 32-bit OBDC
- Define a [User DSN]
  - [Add], [SAS], [Finish]
- [Servers]
  - Server Name: mySAS
- [Configure]
  - SAS Path: "c:\sas\sas.exe"
  - Working Directory: "c:\sas"
  - SAS Parameters: "-initstmt %sasodbc(mySAS) -comamid dde -icon –nolog -noautoexec"
- [OK], [<<Add<<]
- [Libraries],
  - Library Name: "mySAS"
  - Host File Name: "c:\temp"
  - [<<Add<<]
- [General]
  - Data Source Name: "mySAS"
  - Description: "SAS ODBC"
  - [ok]
- Can also define a [File DSN]...
- MS Access–Make blank database
  - Get External Data, Input
  - Link Tables
  - File of Type: OBDC Database()
  - Look in: Machine Data Source
  - DSN Name: sas … OK
- SAS should now start
- SAS datasets under "c:\temp\" should appear
- Select a SAS dataset … OK … OK … Open your SAS dataset

Some ways to speed up your SAS programs

- It is usually much faster to use native operating system commands for copying & deleting datasets, catalogs, etc. You can use the new version 8 command to execute o/s commands asynchronously in order to save even more time.
- It is advisable to clean up work space during a SAS session since SAS only does so at the end. Particularly large work datasets should be deleted by you when they are no longer needed, otherwise they may take valuable space
- You should clean up memory every so often, by using the "CDE P" command to purge unused modules from memory.
- You can split data into smaller bits for sorting. This is especially useful if they can fit in memory. In one of my

examples I saved 36% of the elapsed time.

- You can use compressed single keys made up of several variables for sorting and indexing. This usually takes less time and results in your dataset being in the same order.
- You should try to reduce I/O contention by putting work, swap and data on different physical disks where possible.

Mixed numeric informats

Unquoted numerics in the informat definition are treated as numbers. Quoted text is treated as character. This can be quite useful if reading data which has mixed values, which need to be interpreted in different ways.

```
Proc format ;
  invalue mixed
    'LOW' = -99
    1-10 = 1
    11-20 = 2
    'BIG' = 99
    other = 0 ;
Run ;
```

Building long selection lists in SQL

In SAS version 6 macro variables may be up to 32k long. This is very useful to store long text strings, such as variable lists. For instance we could make a list of employees in one dataset to select from another.

```
proc sql ;
  select name into :namelist separated
by ',' from sasuser.class where sex='F'
;
quit ;
%put namelist=&namelist;
```

Where on output dataset

Where clauses can be used for filtering data. Usually they are used with input data, but they can be used with output data too. One good use of this is for keeping selected _type_ values produced by a PROC SUMMARY.

```
proc summary data=sasuser.houses ;
  class style street ;
  var _numeric_ ;
  output sum=
         out=temp(where=(_type_      in
(1,3))) ;
run ;
```

```
proc freq ;
  table _type_ ;
run ;
```

SAS OLE automation server

You might be wondering how I ran my demos from PowerPoint? PowerPoint can run programs by associating an action setting with some text or graphic. The program I run is the SAS OLE automation server, SASOACT.EXE. Some examples of using this are as follows:

```
SASOACT.EXE                    action=Open
datatype=SASFile filename=="Test.sas"
SASOACT.EXE                  action=Submit
datatype=SASFile filename=="Test.sas"
SASOACT.EXE  action=Open  datatype=Data
filename="Houses.sd2"
```

Some nice stuff on the web

- You can access the latest SAS Online documentation at
  - http://V8doc.sas.com/sashtml
- You can get the last 3 SUGI proceedings at
  - http://www2.sas.com/proceedings/sugi25/procced.pdf
- There are lots of great resources at
  - www.sashelp.com
- Some newsletters with tips & techniques are:
  - The missing semicolon – www.sys-seminar.com
  - VIEWS news – www.views-uk.org

Contact Information

Email: philipmason@email.com
Web: http://how.to/usesas