SAS® Software Tips : My experience as a SAS-L MVP

Philip Mason, Independent Consultant

Abstract

This paper covers a diverse range of SAS tips. I have broken the sections up into AF, General and 6.11 tips. The AF tips mainly apply to Frame programming in a GUI environment. The General tips should be applicable to most platforms. The 6.11 tips apply to the latest release of SAS, version 6.11. These tips have been tried out under MS Windows, but also should apply to other 6.11 platforms, such as OS/2.

Introduction

Since August 1994 I have produced a SAS Tip most days which is sent out over the Internet to various people, including the SAS-L user group. For my efforts I was voted the Most Valuable SAS-Ler (jointly with Ian Whitlock) at the last SUGI conference. In this paper I will share some of the best SAS Tips that I have come across in that time. Some are simple, elegant and undocumented. Others are clever techniques which make use of more familiar features of SAS in interesting ways. The tips are applicable to SAS users with a variety of levels of experience.

SAS/AF Tips

These tips apply to the SAS/AF product, which is used for interactive applications development.

Cursor shapes

Usually the shape of the mouse pointer is an arrow. But by using the _set_cursor_shape_ method you can tell SAS what shape that the mouse pointer should take when moving over a widget in a frame. This works, but is not supported in SAS 6.10. This is supported and documented in SAS 6.11.

Cursor shapes can be very useful in conveying information to the user as to what actions can be undertaken by clicking with the mouse. For instance using cursor shape 18 you get a circle with a diagonal line through the middle, which can be used to indicate that whatever the cursor is currently on can not be clicked on. Whereas cursor shape 19 is an arrow with an attached question mark. This symbol is often used to indicate that clicking on an object will provide help with its function.

Example instructions.

To try out cursor shapes, make 20 widgets in a frame (eg. pushbuttons). As you make them they will be automatically named obj1 to obj20. Then copy the following code into the source window. Compile it. Testaf it. Now you can see the mouse pointer change shape as it moves over each of the widgets.

Source code

```
init:
  call notify('obj1','_set_cursor_shape_',1) ; * Hourglass ;
  call notify('obj2','_set_cursor_shape_',2) ; * Arrow ;
  call notify('obj3','_set_cursor_shape_',3) ; * Cross-hair ;
  call notify('obj4','_set_cursor_shape_',4) ; * Med. Magnifying glass ;
  call notify('obj5','_set_cursor_shape_',5) ; * Hand ;
  call notify('obj6','_set_cursor_shape_',6) ; * Copy ;
  call notify('obj7','_set_cursor_shape_',7) ; * Question mark ;
  call notify('obj8','_set_cursor_shape_',8) ; * Small Magnifying glass ;
  call notify('obj9','_set_cursor_shape_',9) ; * Big Magnifying glass ;
  call notify('obj10','_set_cursor_shape_',10) ; * Horiz. double arrow ;
  call notify('obj11','_set_cursor_shape_',11) ; * Vert. double arrow ;
  call notify('obj12','_set_cursor_shape_',12) ; * 4-way arrow ;
  call notify('obj13','_set_cursor_shape_',13) ; * TL to BR dble arrow ;
  call notify('obj14','_set_cursor_shape_',14) ; * BL to TR dble arrow ;
  call notify('obj15','_set_cursor_shape_',15) ; * Screen ;
  call notify('obj16','_set_cursor_shape_',16) ; * 3 charts ;
  call notify('obj17','_set_cursor_shape_',17) ; * OK ;
  call notify('obj18','_set_cursor_shape_',18) ; * Dont ;
  call notify('obj19','_set_cursor_shape_',19) ; * Pntr & qstn mark ;
  call notify('obj20','_set_cursor_shape_',0) ;  * Vertical cursor ;
return ;
```

Alternate way to test out cursor shapes

Make a slider or scrollbar. Set the min to 0 and the max to 19 and use the following code:

```
sbar: call notify( 'sbar', '_set_cursor_shape_', sbar );
      return;
```

This code will cause the shape of the cursor to change as you move the slider/scrollbar to new positions. Clicking on the arrow at the right will take you through all the available cursor shapes.

You can also set the cursor shape on the FRAME itself.

```
init: call notify( '.', '_set_cursor_shape_', number );
```

Listing AF entries

A listing can be obtained of the attributes of certain AF catalog entries by entering an asterisk ('*') beside them in the "BUILD: DIRECTORY" window. The listing is sent to the MESSAGE window. The command will not work from the other BUILD window (accessed by entering 'BUILD'), nor will it work from the CATALOG window.

To get to the "BUILD: DIRECTORY" window enter:

BUILD libref.catalog

eg. *build sasuser.profile*

Sample output from a '*' next to a SLIST entry.

```
SLIST(
  AUDIO_OUT='Speaker' (F)
  AUDIO_LEVEL=80 (N)
  VIDEO_SIZE=1 (N)
)[3]
```

Sample output from a '*' next to an SCL entry.

```
_SCLD1_(
  RELEASE=6073 (I)
  EXESIZE=0 (I)
  FLAGS='1' (F)
```

```
)[3]
```

Sample output from a '*' next to a FRAME entry.

```
APPL(
   OBJECTS=(
      OBJ1=(
         NAME='OBJ1' (P)
         LABEL='OBJ1' (P)
         USERATTR=()[185] (L)
         GRSEG='MIS.PILOT.B2BK.GRSEG' (T)
         HSIZE=6.3333336923 (N)
         VSIZE=4.0625 (N)
         _CLASSNAME_='GRAPH.GRAPH' (T)
         _LEN_=40 (I)
         CHILDREN=()[11] (L)
      )[9] (L)
      MESSAGE=(
         NAME='MESSAGE' (P)
         LABEL='MESSAGE' (P)
         USERATTR=()[187] (L)
         TEXT='No message' (T)
         FONT='ZAPFBI   ' (T)
         COLOR='YELLOW' (P)
         _CLASSNAME_='GTEXT.GTEXT' (T)
         _LEN_=200 (I)
      )[13] (L)
   )[19] (L)
   RESNAME='BUILD.RESOURCE' (T)
   GATTR=(
      _WINDOW_TYPE_='DIALOG' (P)
      _BANNER_='NONE' (P)
      NAME='Message' (P)
      SROW=10 (S)
      SCOL=20 (S)
      NROW=10 (S)
      NCOL=50 (S)
      _CLASSNAME_='FRAME.FRAME' (T)
      EXESIZE=644 (I)
      _REGION_=()[33] (L)
      _APPL_=23 (I)
   )[189] (L)
)[3]
```

Making & using Windows Help

You can use build your own windows help files and then use them from your SAS/AF applications.

1. Get a tool to make input to the help compiler. There are many shareware tools available, such as "HELLLP!" which can be found in many places, including the Internet CICA archive. "HELLLP!" is a template for MS Word 2 or 6 which allows you to create help files using those word processors. You can get it from one of these sites:

   ftp://ftp.monash.edu.au/pub/win3/winword/helllp27.zip

   ftp://ftp.winsite.com/ftp/pub/win3/winword/helllp27.zip
2. Get the Microsoft Help compiler, to compile the Help source. This is freely available from this Internet URL:

   ftp://ftp.microsoft.com/Softlib/MSLFILES/HCP505.EXE
3. Optionally get the Microsoft Shed editor, which lets you add graphics and hotspots to your help. This is also freely available from this URL:

   ftp://ftp.microsoft.com/Softlib/MSLFILES/SHED.EXE
4. Configure your tool, "HELLLP!" for instance, to work with the help compiler and SHED editor (if you are using it).
5. Build, compile & test a Help file. Examples are provided in the packages, along with extensive online help.
6. Call your Help from your application, using the WINHELP SCL function.

Example

```
init:
  rc=winhelp('help_contents','c:\windows\cardfile.hlp') ;
  put rc= ;
return ;
```

Syntax

```
WINHELP('h-method', 'h-methodparm1', h-methodparm2);
```

The WINHELP function starts the Windows Help application (WINHELP.EXE) with the help information that you request using the parameters.

The h-method argument must be one of eight help methods that this function supports. Depending on the value of h-method, the h-methodparm1 and h-methodparm2 arguments have various meanings.

The eight supported methods[1] are:

HELP_CONTENTS

HELP_QUIT

HELP_CONTEXT

HELP_INDEX

HELP_HELPONHELP

HELP_CONTEXTPOPUP

HELP_KEY

HELP_PARTIALKEY

Saving catalog space

To compress and reuse the space in a catalog, which would otherwise be wasted after doing many compiles, use REPAIR in PROC DATASETS. I have managed to save large amounts of space in some cases. There are other ways of compressing the space used by catalogs such as copying the catalog to a new catalog, deleting the old one and renaming the new one.

Example: this will compress sasuser.profile

```
Proc datasets library=sasuser ;
  repair profile / mt=cat ;
quit ;
```

Catalog member Stats

In Proc Catalog you can use the undocumented STAT parameter on the CONTENTS keyword to give you space used by catalog members.

Example Code

```
options ls=132 ; * Set linesize wide or output looks a bit weird ;
PROC CATALOG C=sashelp.eis ;
  CONTENTS STAT ;
run ;
```

Left side of output

```
# Name      Type      Date      Description

 1 ADDOBJ    CBT       04/06/94  EIS: Add a new object type
 2 ADDOBJ2   CBT       04/06/94  EIS: Browse Attribute Definition
 3 APPLDS    CBT       04/06/94  EIS: CBT for setup applications
```

1See online help for more information. The methods are described in more detail there.

```
 4 APPLICAT  CBT       04/06/94  ASSIST: CBT for EIS builder
 5 APPLSEL   CBT       04/06/94  HELP: List of applications in data set
```

Right side of Output

Page Size	Block Size	Num of Blocks	Last Block Bytes	Last Block Size	Pages
6144	6144	3	125	256	3
6144	6144	2	2588	6144	2
6144	6144	5	4041	6144	5
6144	6144	1	4833	6144	1
6144	1024	4	178	256	1

General Tips

Where on Output dataset

In SAS 6.11 you can use a where clause on an output dataset. However in previous versions of SAS (I tried 6.08 on MVS) you are unable to do this.

Useful application

When doing a PROC SUMMARY you may want to keep a range of _TYPE_ values, perhaps for use in a data warehouse. Using this tip you can only write out the values of _TYPE_ that you wish to keep. This can be very useful, particularly in saving space.

Example: SAS 6.08 TS425, under MVS (IBM m/f)

Note: the WHERE= option causes a warning message telling us that it is not used in this context.

```
1    data x ;
2    do i=1 to 10; output ;end;run;

NOTE: The data set WORK.X has 10 observations and 1 variables.
NOTE: The DATA statement used 0.06 CPU seconds and 4193K.

3    data y(where=(i gt 5)) ;
                   -----
                   70
4      set x ;
5    run ;

WARNING 70-63: The option WHERE is not valid in this context.  Option
               ignored.

NOTE: The data set WORK.Y has 10 observations and 1 variables.
NOTE: The DATA statement used 0.03 CPU seconds and 4294K.
```

Example: SAS 6.11, under Microsoft Windows 3.1

Note: the WHERE= option works correctly in this case.

```
9    data x ;do i=1 to 10;output;end;run;

NOTE: The data set WORK.X has 10 observations and 1 variables.
NOTE: The DATA statement used 0.71 seconds.

10   data y(where=(i gt 5)) ;
11     set x ;
12   run ;

NOTE: The data set WORK.Y has 5 observations and 1 variables.
NOTE: The DATA statement used 0.55 seconds.
```

Operating other programs from SAS with DDE[2]

Using DDE SAS is able to control other programs by issuing commands to them which they recognise. The commands used depend on the application being controlled. A simple way to find out which commands to use is to turn on the applications macro recorder, do something and then edit the macro recorded to see what the commands are.

Steps to "remote control" applications from SAS.

1. Work out what commands to use, either with manual or macro recorder.
2. Allocate the fileref. Remember to use the applications program name and the word SYSTEM separated by a vertical bar.
3. Write commands to the fileref. Commands should be contained within square brackets.

The following are 2 examples, one controlling Lotus 123 and the other Microsoft Word.

Sample LOG Listing: Example 1

```
filename lotus dde '123w|system' notab ;   *** Program name is 123w.exe ;
data _null_ ;
  file lotus ;
  put '[run({CHART-NEW A:A1..A:F14})]' ; *** Create a new chart ;
  put '[run({SELECT "CHART 1";;"CHART"})]' ; *** Select it ;
  put '[run({CHART-RANGE "X";A:A1..A:A4;"Line";"NO"})]' ; *** Set the X
range ;
  put '[run({CHART-RANGE "A";A:B1..A:B4;"Bar";"NO"}dd)]' ; * Set the Y
range and
                                                          plot a bar
chart ;
run ;

NOTE: The file LOTUS is:
      FILENAME=123w|system,
      RECFM=V,LRECL=256

NOTE: 4 records were written to the file LOTUS.
      The minimum record length was 30.
      The maximum record length was 48.
NOTE: The DATA statement used 1.37 seconds.
```

Sample LOG Listing: Example 2

```
*** Program name is Winword.exe ;
filename word dde 'winword|system' notab ;
data _null_ ;
  file word ;
  put '[FileOpen .Name = "phill.DOC"]' ; *** Open file called phill.doc ;
  put '[macro1]' ; *** Execute a macro called macro1 ;
run ;

NOTE: The file WORD is:
      FILENAME=WinWord|system,
      RECFM=V,LRECL=256

NOTE: 1 record was written to the file WORD.
      The minimum record length was 30.
      The maximum record length was 30.
NOTE: The DATA statement used 3.62 seconds.
```

Making sure numbers are numeric when using DDE

When using DDE to write numbers to a spreadsheet/database, they can sometimes be inadvertently read in as character values. This can happen if using a variable to hold your tab character, since after a variable is output a space is written. To get around this position the column pointer over the space that was written (see example).

[2]Dynamic Data Exchange

Example

```
filename lotus dde '123w|test.wk4!a:a1..a:b3' notab ;

* Writing numeric values directly to spreadsheet via DDE ;
data _null_ ;
  retain tab '09'x ; ** Define a tab character ;
  file lotus ; ** Directs output to spreadsheet via DDE link ;
 * In Lotus: 1 is numeric, but 2 is character due to implicit space after
variable TAB ;
 *          3 & 4 are numeric, since implicit space is overwritten ;
 *           5 & 6 are numeric since there is no implicit space, due to
constant being used ;
  put '1'      tab
      '2'      /
      '3'      tab +(-1)
      '4'      /
      '5'      '09'x
      '6'      ;
run ;

NOTE: The file LOTUS is:
      FILENAME=123w|test.wk4!a:a1..a:b3,
      RECFM=V,LRECL=256

NOTE: 3 records were written to the file LOTUS.
      The minimum record length was 3.
      The maximum record length was 4.
NOTE: The DATA statement used 0.66 seconds.
```

Using the colon in variables lists

The colon can be used as a wildcard in variable lists.

Example

ABC: means all variable names beginning with ABC.

```
      data x(keep=a:) ;
      a1=1 ;
      a2=10 ;
      a3=100 ;
      b1=1000 ;
      b2=10000 ;
      run;


NOTE: The data set WORK.X has 1 observations and 3 variables.
NOTE: The DATA statement used 0.01 CPU seconds and 1435K.
      proc print ;
      var a:;
      run ;


NOTE: The PROCEDURE PRINT used 0.01 CPU seconds and 1489K.
```

Notice that the 3 variables starting with the letter 'a' have been kept. We can also use 'a:' in procedures.

The colon must be at the end of the name, not embedded—AB:C is invalid. Colons can be used in most places where other abbreviated variable lists such ABC1-ABC99 are allowed.

10 ways to minimise I/O

Generally SAS is I/O intensive, rather than CPU intensive. As my friend Ken Williams say "The Best I/O is the one you didn't do". Thus a saving in I/O will improve the performance of your SAS program.

10 ways to minimise I/O

1. Use the LENGTH statement to minimise variable lengths. Numerics are 8 bytes by default, however it only takes 3 bytes to store a SAS date value. Under MVS to minimum number of bytes for a numeric is 2, whereas under windows it is 3.
2. Use the CLASS statement rather than the BY statement, which might require a SORT. When doing a summary, for example, a CLASS statement will summarise all the data in memory, not requiring data to be pre-sorted. Whereas a BY statement requires that the data be sorted before it is summarised, since each BY group is processed one at a time. Having to sort the data as well as summarise it requires much more I/O.
3. Use the DROP and/or KEEP statements to minimise observation length. Either use DROP to drop variables not required from the dataset. Otherwise use KEEP to specify which variables are to be kept. This minimises the bytes per observation, which means more observations can be written for each I/O, resulting in less I/Os.
4. Only SORT when necessary. Sorting does I/O to read data in and then some more to write it out. Often I/O is also done to temporary sort work datasets during the sort process itself.
5. Read raw data only once, keeping it in permanent datasets if required again. This avoids having to read through the raw data again when it is required.
6. Create multiple datasets from the one datastep if possible, rather than reading through one dataset once for each dataset that is to be created.
7. Use the WHERE statement with procedures to avoid doing a datastep. Otherwise you need to do a datastep before the proc to select the data required.
8. Use the _NULL_ datastep when you don't need to keep the output. Even just specifying "DATA;" will still create a dataset. If you don't want to write data to a dataset then use "data _null_".
9. Compress large SAS datasets, but beware that compression can use more space. Compression works best when there is a lot of text data in a dataset, it doesn't work well where data is all numeric. If compression does save space, then I/Os will be reduced. Remember to check the compression ratios in the LOG.
10. Develop & test programs on a small subset of the data. This means that you do less I/O while testing/developing the code. Once the code is tested then it can be run against the entire dataset. Test data can sometimes be placed on faster devices which do I/O quicker - eg. RAM disk, SCSI drives, cached disk drives.

Note: the list is not comprehensive, it merely attempts to provide a few ideas for investigation. Not all of the points will always reduce I/O time.

Do-it-yourself log messages

Entering code like this ...

```
data _null_;
  put 'NOTE: this is a note';
  put 'WARNING: this is a warning';
  put 'ERROR: this is an error';
run;
```

causes the SAS log to highlight this in exactly the same way it highlights its own Note, Warning and Error messages[3]. What

[3]It does not seem to be documented in any of the manuals or SAS Notes anywhere.

appears to be happening is SAS examines the first word that is sent to the log and if it is NOTE:, WARNING: or ERROR: it highlights it. This works on MVS SAS 6.07+, Windows SAS 6.08+ and most other versions. Under MVS overprinting is used for highlighting, whereas in GUI environments colours are used.

Under OS/2 Display Manager session, the NOTE statement appeared with black print, WARNING with yellow, ERROR with red, all just as though they had been issued by the SAS system itself.

Note: the keywords must be entered in uppercase, or it will not work - ie. ERROR, not error.

SQL Coalesce function

In PROC SQL the COALESCE function can be used to set default values when variables have missing values and is almost essential when doing left, right or full joins where the key fields are included in the output set only once.[4]

Example

To replace missing values of name from dataset x, with values of name from y ...

```
Select address,
  coalesce(x.name,
           y.name) as name
  from y
    left join
      x
    on x.name=y.name ;
```

Format Nesting

From SAS 6.07 onwards you can refer to other formats from within formats (Value formats, not Pictures). You can also nest your formats up to five levels deep[5], although this can degrade performance.

Example

```
proc format ;
     value loads
       5000-<6000 = 'Over 5,000'
       6000-<7000 = 'Over 6,000'
       7000-<8000 = 'Over 7,000'
       8000-<9000 = 'Over 8,000'
       other      = 'Mega!' ;
NOTE: Format LOADS has been output.

     value couple
       1 = 'Bingo!'
       5000-<10000 = (!!loads10.!)
       other=(!!comma6.!) ;
NOTE: Format COUPLE has been output.

NOTE: The PROCEDURE FORMAT used 0.01 CPU seconds and 1480K.

   data _null_ ;
     input x ;
     put x couple. ;
   cards ;
1
2
3
12
1234
5678
8888
9999
12345
;
NOTE: The DATA statement used 0.01 CPU seconds and 1480K.
   run ;
```

[4]See "SAS Guide to the SQL Procedure, Usage & Reference, Version 6, First Edition", page 73.

[5]See "Changes & Enhancements for SAS 6.07", page 213

Output of datastep

```
      1
Bingo!
      3
     12
  1,234
Over 5,000
Over 8,000
Mega!
12,345
```

New SAS 6.11 Features

Where on Output dataset[6]

In SAS 6.11 you can use a where clause on an output dataset. However in previous versions of SAS (I tried 6.08 on MVS) you are unable to do this.

Useful application

When doing a PROC SUMMARY you may want to keep a range of _TYPE_ values, perhaps for use in a data warehouse. Using this tip you can only write out the values of _TYPE_ that you wish to keep. This can be very useful, particularly in saving space.

Example using SAS 6.08 TS425, under MVS

Note: the WHERE= option causes a warning message telling us that it is not used in this context.

```
1    data x ;
2    do i=1 to 10; output ;end;run;

NOTE: The data set WORK.X has 10 observations and 1 variables.
NOTE: The DATA statement used 0.06 CPU seconds and 4193K.

3    data y(where=(i gt 5)) ;
                  -----
                  70
4      set x ;
5    run ;

WARNING 70-63: The option WHERE is not valid in this context.  Option
               ignored.

NOTE: The data set WORK.Y has 10 observations and 1 variables.
NOTE: The DATA statement used 0.03 CPU seconds and 4294K.
```

Example using SAS 6.11, under MS Windows 3.1

Note: the WHERE= option works correctly in this case.

```
9    data x ;do i=1 to 10;output;end;run;

NOTE: The data set WORK.X has 10 observations and 1 variables.
NOTE: The DATA statement used 0.71 seconds.

10   data y(where=(i gt 5)) ;
11     set x ;
12   run ;

NOTE: The data set WORK.Y has 5 observations and 1 variables.
NOTE: The DATA statement used 0.55 seconds.
```

[6]Please note that this paper was written without any printed SAS 6.11 documentation, so my 6.11 tips are based on online documentation and/or trial and error.

Data Encryption

From SAS 6.11 on data can now be natively encrypted by SAS using the ENCRYPT= dataset name option. When using ENCRYPT=, you must specify a read password (READ=).

To test the encryption I loaded the unencrypted file into an editor and could see text clearly. I then loaded the encrypted file and couldn't make out anything, showing that the encryption worked.

Example

```
data test(read=yeah encrypt=yes) ;
  set sasuser.houses ;
run ;
```

Log

```
1     data test(read=XXXX encrypt=yes) ;
2       set sasuser.houses ;
3     run ;

WARNING: The file WORK.TEST.DATA is not ALTER protected.
         It could be deleted or replaced without knowing the password.
NOTE: The data set WORK.TEST has 15 observations and 6 variables.
NOTE: The DATA statement used 4.17 seconds.
```

CATALOG engine

There is a new engine in SAS 6.11 which lets you read SAS Catalog members using a datastep. Usage is like this ...

```
FILENAME test CATALOG 'mylib.mycat.mainmenu.scl' ;
```

I thought that a good way to demonstrate what this can do was to include a sample macro which makes use of it.

The macro reads all the members of a particular type from a SAS catalog and then creates a SAS dataset for each one. It then combines all of these datasets into one and writes the contents out to a text file. I wrote this so that I could take all my SCL away with me and view it with and text editor.

The macro uses "Call Execute" functions, which put SAS code onto a stack which is then run after the current datastep finishes.

Example macro

```
%macro catdset(libname,catname,objtype) ;

 * Libname ... libname catalog is in ;
 * Catname ... Catalog which contains members ;
 * Objtype ... Type of members that you will analyse ;

* Delete the dataset that I am about to create ;
  proc datasets lib=work ; delete _all ; run ; quit ;
  run ;
* Define the 2 fields that I want on my dataset ;
  data _all ;
    length member $ 8
           line $ 78 ;
  run ;
* Create the SAS code to be run ;
  data _null_ ;
* Read in the name of each catalog member of the specified type
  from libname.catname ;
    set sashelp.vcatalg(where=(libname="%upcase(&libname)" &
                               memname="%upcase(&catname)" &
                               memtype='CATALOG' &
                               objtype="%upcase(&objtype)"))
        end=end ;
* Write code out to define a fileref to read the catalog member ;
    call execute('filename x'!!left(_n_)!!
        ' catalog "'!!
        trim(libname)!!'.'!!
        trim(catname)!!'.'!!
        trim(objname)!!'.scl" ; ') ;
* Write code to read the contents of each catalog member ;
    call execute('data work.'!!objname!!' ;') ;
    call execute(' retain member "'!!trim(objname)!!'" ;') ;
    call execute(' infile x'!!left(_n_)!!' firstobs=5 ;') ;
    call execute('input line $char78. ;') ;
    call execute('run ;') ;
* Write code to append the member read to the other members read ;
    call execute('data work._all ;') ;
    call execute(' set work._all work.'!!trim(objname)!!' ;') ;
    call execute('run ;') ;
  run ;
```

```
 * Write all the SCL code out to a text file ;
  data _null_ ;
    file "c:\&catname..&objtype" ;
    set _all ;
    put member $8. '> ' line $char78. ;
  run ;
%mend catdset ;
```

Example call of example macro

```
*** Read all the source code from the SCL members in MIS.PILOT ;
* I use this to put all of my SCL code onto a text file
  which means I can access it without the need for SAS ;
%catdset(mis,pilot,scl)
```

A better way to view & edit data

There is a nice new DM command in SAS 6.11 called VIEWTABLE. You can use it instead of FSView, FSEdit or FSBrowse to look at data and optionally change it. You can invoke the command on its' own or put a dataset name after it. It lets you browse and/or edit a dataset with a "spreadsheet" look and feel. It seems to use the new data table object to do this.

It also provides a lot more functionality including letting you build where clauses with a where clause builder. Quite a useful tool, especially for those who aren't very familiar with SAS.

Conclusion

Although a paper such as this doesn't strictly require a conclusion I thought I would put one in to make several recommendations to those of you who want more SAS tips. There are many sources of information (Tips!) about SAS.

How to get more tips

If you want more tips then you can obtain them in one of the following ways:

- If you are using a SAS product then make sure you have all of the available manuals for it that are available from SAS Institute. Make sure you go through each manual to ensure you are familiar with what is there. I have discovered many tips by doing that.
- Look through the online help, including the sample programs. This is a great source of information.
- Subscribe to the internet SAS user group SAS-L. Here you will not only receive my tips, but many other tips from the SAS-L community. To subscribe to SAS-L send a message to "listserv@uga.cc.uga.edu" with the following in the body of the message, "subscribe sas-l firstname surname" - leave quotes off and substitute your name.
- Find out about the books available from SAS Institute through the "Books By Users" programme. There are many excellent books available there.
- Get yourself onto the "SAS Communications" mailing list since that free quarterly magazine has a section with Tips & Techniques. Call your local SAS office to receive this. Be aware that there is an American and European version of this. The European one is called "Inform" and is also well worth receiving if you can.

- Subscribe to the SAS journal "Observations". This one is not free, but has several lengthy articles covering a topic in depth. You can subscribe to this through your local SAS office.
- Buys the books that SAS provides covering tips & techniques. eg. "Macro tips & techniques", "SAS Tips & Techniques", etc.
- Watch out for my new book coming soon from the SAS Books By Users program. It will contain over 100 tips.

Author contact Details

Philip Mason
16 Wood Street,
Wallingford, Oxfordshire,
OX10 OAY
England

Phone: +44 1491 824905