Australian Government
Department of Social Services

What's new in SAS 9.4
(and what's not but neat)

16 May 2017

So what is new in 9.4?

Actually not very much for the SAS user.

There have been major changes in the platform (web and security mainly with some I/O optimisation) which are of no real interest to most but add to the stability and maintainability of SAS. External dependencies are also being whittled away.

The user related changes are mainly in:

- The DS2 language (lower level programming control)
- The FedSQL language (ANSI SQL:1999 standard, vendor neutral)
- Hadoop support
- ODS enhancements (eg CSS application, HTML5 and PowerPoint output, and new options and procedures)
- AES dataset encryption (including indexes and metadata)
- Some new functions (incl COT,SEC&CSC) and formats (mainly time)

PROC DELETE

PROC DELETE has been re-instated as a formally supported production procedure, rather than the undocumented and unsupported experimental version.

The main advantage over the usual PROC DATASETS; delete …; is speed as it bypasses existence checks.

Not sure how useful that would be, but it is an option.

The FCOPY function

The FCOPY function takes two files references as parameters and copies the first file pointed at to the second. This is a massive improvement over the previous methods which were typically tricky to get right (the best general solution was to do a byte to byte copy). The syntax is:

```
rc = fcopy('src','dest');
```

Even though there are a few ways to use it via the options on the filename statement, there is only one way it should be used by using RECFM='N', and if information is required, MSGLEVEL='I'.

RECFM='N' specifies a binary copy without record boundaries.

The FCOPY function (cont)

An example of the general usage is:

```
options msglevel=I;
filename src  "\path\src file"  recfm=n;
filename dest "\path\dest file" recfm=n;

data _null_;
  length msg $384.;

  rc=fcopy('src','dest');

  if rc=0 then
    put 'Copied src to dest';
  else do;
    msg=sysmsg();
    put rc= msg=;
  end;
run;

filename src  clear;
filename dest clear;
```

The ZIP file engine

The FILENAME ZIP access method makes processing standard WinZip-like zip files much easier compared to using the undocumented SASZIPAM filename engine or unnamed pipes.

It makes the zip file look and act like a directory, allowing selective file read/write access. It does have a limitation in that it won't handle other zip types like bzip2, so pipes still have their place, so long as the data is in a line feed delimited format not binary.

The ZIP file engine (cont)

For example, this is a complicated pipe construct to read a group of related datasets (ID_DATA_01, ID_DATA_02 etc) from a zip file containing bzipped members without having to unzip any of them, something the new ZIP engine can't handle. The data is CSV-like data and the –p directive extracts the data to pipe and in binary format, which is then piped to the bunzip2 command for final unzipping.

```
filename archive pipe "unzip -p '&latest_archive' 'ID_DATA_*.csv.bz2' | bunzip2";

data id_data;
  infile archive dsd dlm='~' termstr=lf missover lrecl=300;

  length id        $20.
         type_code $6.
         <etc>
  ;
  input id
        type_code
        <etc>
  ;
run;
```

The ZIP file engine (cont)

The engine has a number of useful options for reading/writing unusual formats such as RECFM = <F>|<N>|<S>|<V> and TERMSTR = <CR>|<CRLF>|<LF>|<NULL>, and subfolders can be written to the zip file as well. The MEMBER="mem" option can be used instead of the aggregate location syntax [fileref(mem)], and wildcards are accepted. However, getting the directory isn't as easy as it should be. One way is:

```
filename archive zip "path\archive name.zip";

data zip_contents(keep=memname);
  length memname $200;
  fid=dopen("archive");
  if fid=0 then
    stop;
  memcount=dnum(fid);
  do i=1 to memcount;
    memname=dread(fid,i);
    output;
  end;
  rc=dclose(fid);
run;
```

	memname
1	acmps_care_approval.txt
2	acmps_care_approval_exceptions.txt
3	acmps_client.txt
4	acmps_merged_client.txt
5	acmps_service_entry.txt
6	acmps_service_entry_exceptions.txt
7	casper_lov.txt

The ZIP file engine (cont)

A simple (concatenated) read of all the text files beginning with ‘a’ would be:

```
filename archive zip "path\archive name.zip" member='a*.txt';

data partial_read;
  infile archive length=len;
  input line $varying500. len;
run;
```

or:

```
filename archive zip "path\archive name.zip";

data partial_read;
  infile archive('a*.txt') length=len;
  input line $varying500. len;
run;
```

The ZIP file engine (cont)

The advantage of the FILENAME ZIP access method is that all the standard, and more importantly, the less used filename options are available (and work properly) . Probably the most useful at Health was the binary streaming, or RECFM=S option, reducing 14TB to 2TB.

```
filename inzip zip "path/ebcdic_data.zip" member="VB_data";

* Reads a variable blocked mainframe sourced EBCDIC file with RDW from a ZIP archive ;
data ebcdic_data;
  infile inzip recfm=s nbyte=_datalen;
  length line $300.; * maximum variable line length ;

  * Read the (4 byte) Record Descriptor Word to determine the line length ;
  _datalen = 4;
  input;

  * Reset the amount of data to read next based on the RDW (only the first 2 bytes used);
  * and save the line length in the dataset.                                            ;
  _datalen = input(_infile_,s370fibu2.)-4;
  data_len = _datalen;

  * Read the exact number of bytes in the variable length line ;
  input;
  line = _infile_;
run;
```

The SHA256HMACHEX function

This function returns the message digest (in expanded hex form) of the (keyed) Hash-based Message Authentication (HMAC) algorithm using the SHA256 (256 bit) hash function.

This is a standard message authentication method, and has been used with MD5 (128 bit HMAC-MD5) and SHA1 (160 bit HMAC-SHA1) as well, but these need a bit of work to build the equivalent in SAS.

This is a good way to securely ‘sign’ data over and above just ensuring it hasn’t been altered via a straight hash by using a secret key.

The call is:
[length digest $64.;]
digest = sha256hmachex(‘key’,’message’<,string indicator>);

The SHA256HMACHEX function (cont)

It does have the disadvantage of only producing standard results for up to 32 kibibyte – 1 byte messages (unlike the specification of 2^{64} -1 bits) but can be easily extended in non-standard ways.

The string indicator (0-3) flags which (if any) of the input parameters is in expanded hex format.

For example:

```
data _null_;
  digest = SHA256HMACHEX('key', 'The quick brown fox jumps over the lazy dog', 0);

  if digest = upcase('f7bc83f430538424b13298e6aa6fb143ef4d59a14946175997479dbc2d1a3cd8') then
    put 'matched';
  else
    put 'not matched';
run;
```

The SHA256HMACHEX function (cont)

If you are really serious about security, it can be hardened against GPU brute force attacks. It took 1 min 40 sec to break the key 'key' with no sophistication, but would take 58-odd days using the following example also using the key 'key':

```
%macro hmac(var=digest,key=,msg=,iterations=0);
  do;
    length &var $64.;
    &var = sha256hmachex(&key,&msg,0);

    drop _i;
    do _i = 1 to &iterations;
      &var = sha256hmachex(&var,&msg,2);
    end;
  end
%mend hmac;

data sign;
  %hmac(key='key',msg='The quick brown fox jumps over the lazy dog',iterations=50000);
run;
```

	digest
1	E02F7B5BEF8B316C4AA2A9695090246AEAF1412FBD658643979AB14D264BAFE3

Tips and Tricks

(but not that new)

SORTSEQ=LINGUISTIC option

This option in PROC SORT is most useful with the numeric awareness sub option. It can find numerics in a string (not necessarily at the front) and use them in the sort process.

```
data list;
  length addresses filenum $50.;
  do i = 15 to 1 by -1;
    addresses = catt(i**2+13,' Smith St');
    filenum   = catt('SMITH',16-i);
    output;
  end;
run;

proc sort data=list;
  by addresses;
run;

proc sort data=list;
  by filenum;
run;

proc sort data=list
    sortseq=linguistic(numeric_collation=on);
  by addresses;
run;

proc sort data=list
    sortseq=linguistic(numeric_collation=on);
  by filenum;
run;
```

	addresses	filenum	i
1	113 Smith St	SMITH6	10
2	134 Smith St	SMITH5	11
3	14 Smith St	SMITH15	1
4	157 Smith St	SMITH4	12
5	17 Smith St	SMITH14	2
6	182 Smith St	SMITH3	13
7	209 Smith St	SMITH2	14
8	22 Smith St	SMITH13	3
9	238 Smith St	SMITH1	15
10	29 Smith St	SMITH12	4
11	38 Smith St	SMITH11	5
12	49 Smith St	SMITH10	6
13	62 Smith St	SMITH9	7
14	77 Smith St	SMITH8	8
15	94 Smith St	SMITH7	9

	addresses	filenum	i
1	14 Smith St	SMITH15	1
2	17 Smith St	SMITH14	2
3	22 Smith St	SMITH13	3
4	29 Smith St	SMITH12	4
5	38 Smith St	SMITH11	5
6	49 Smith St	SMITH10	6
7	62 Smith St	SMITH9	7
8	77 Smith St	SMITH8	8
9	94 Smith St	SMITH7	9
10	113 Smith St	SMITH6	10
11	134 Smith St	SMITH5	11
12	157 Smith St	SMITH4	12
13	182 Smith St	SMITH3	13
14	209 Smith St	SMITH2	14
15	238 Smith St	SMITH1	15

Option DLCREATEDIR - Creating subfolders with the libname statement

The option DLCREATEDIR is off by default, but if enabled allows the LIBNAME statement to create a new subfolder as part of the call. This can be used even if NOXCMD is set. (The usual method is to use OS commands via the X command). The disadvantage is that only one sub-level can be created at a time, and the other problem is that this is a powerful option with consequences if there is a programming error, but not as disastrous as a bad X call.

There are a couple of neat tricks (stolen from Scott Bass's SNUG tips and tricks, and Chris Hemedinger's blog) which makes creating multiple sublevels easier.

Option DLCREATEDIR (cont)

Conventional:

```
* create two subfolders in the WORK area;
options dlcreatedir;

%let outdir=%sysfunc(getoption(work)); * &sasworklocation doesn't work ;
libname res "&outdir./results";
libname res "&outdir./results/images";
* clear the libref, note separate librefs could have been used;
libname res clear;
```

Or a concatenated libname trick:

```
libname res ("&outdir./results", "&outdir./results/images");
libname res clear;
```

Option DLCREATEDIR (cont)

Or a variation which uses previous results to simplify the form:

```
options dlcreatedir;
libname res  &sasworklocation; * set the base (existing) location ;
libname res "%sysfunc(pathname(res))/results";
libname res "%sysfunc(pathname(res))/images";
libname res "%sysfunc(pathname(res))/subfolder";
libname res "%sysfunc(pathname(res))/subsubfolder";
libname res clear;
```

This suggests a macro which loops through the path levels creating each in turn.

Single quoting of values

The quote function is useful to prepare strings for building into external programs (such as email) as it resolves the balancing of quotes, but has the disadvantage that it encloses in *double* quotes which can cause issues related to macro substitution. Sometimes a single quoted value is required (e.g. RDMSs) but there isn't a function (or option in QUOTE) which does this. There is a easy way to do this using a couple of simple functions.

```
var = cats("'",tranwrd(var,"'","''"),"'");
```

```
data email_list;
  length email_addr $256.;
  email_addr = "john.smith@dss.gov.au"; output;
  email_addr = "patrick.o'donnelly@dss.gov.au"; output;
  email_addr = 'fred&jane <fj299@gmail.com>'; output;
run;

data email_list;
  set email_list;
  email_addr = cats("'",tranwrd(email_addr,"'","''"),"'");
run;
```

```
'john.smith@dss.gov.au'  'patrick.o''donnelly@dss.gov.au'  'fred&jane <fj299@gmail.com>'
```

INTO :mvar SEPARATED BY ‘str’ option

Another useful piece of functionality is the SQL directive which creates a macro variable containing a list of values from a dataset variable separated by a constant string. This example combines the previous single quote code to generate a macro variable containing a list of email addresses to be used in an SMTP call.

```
data email_list;
  length email_addr $256.;
  email_addr = "john.smith@dss.gov.au"; output;
  email_addr = "patrick.o'donnelly@dss.gov.au"; output;
  email_addr = 'fred&jane <fj299@gmail.com>'; output;
run;

data email_list;
  set email_list;
  email_addr = cats("'",tranwrd(email_addr,"'","''"),"'");
run;

proc sql noprint;
  select email_addr into :to_addr separated by ', '
  from email_list
  ;
quit;

%put &to_addr;
```

```
'john.smith@dss.gov.au', 'patrick.o''donnelly@dss.gov.au', 'fred&jane <fj299@gmail.com>'
```

Email Macro

Putting it together, an example of usage:

```
%macro build_email_list(mvar_name=to_addr,csv_posn=1,path=,list=,firstobs=1,numobs=max,dlm=',');
  %global &mvar_name;

  data _email_list_view / view=_email_list_view;
    infile "&path\&list" dsd dlm=&dlm firstobs=&firstobs obs=&numobs lrecl=32767;

    length %do i=1 %to &csv_posn; dummy&i %end; $1.;
    length email_addr $256.;
    input %do i=1 %to %eval(&csv_posn - 1); dummy&i %end;
          email_addr
    ;
    email_addr = cats("'",tranwrd(email_addr,"'","''"),"'");
  run;

  proc sql noprint;
    select email_addr into :&mvar_name separated by ', '
    from _email_list_view
    ;
  quit;
%mend build_email_list;

%build_email_list(mvar_name = to_addr
                 ,csv_posn  = 1
                 ,path      = \path\list
                 ,list      = Email_List.csv
                 ,firstobs  = 2);
run;

filename mail email;
data _null_;
  file mail
       TO=(&to_addr)
       FROM='<Support@dss.gov.au>'
       subject="email subject";

       put "Some text";
run;
```

Thank you

Greg Boag

Tips and Tricks cont

(more for reference)

PROC FCMP

FCMP stands for Function Compiler and allows you to create, test and store SAS functions and subroutines for use by other procedures and in data steps. The syntax within the function/subroutine declarations closely follows normal SAS syntax. An example out of the manual, (note the return and outargs statements for returning values):

```
proc fcmp outlib=work.MySubs.MathFncs;
  function day_date(indate, type $);
    if type="DAYS" then
      wkday=weekday(indate);
    if type="YEARS" then
      wkday=weekday(indate*365);

    return(wkday);
  endsub;

  subroutine inverse(in, inv);
    outargs inv;

    if in=0 then
      inv=.;
    else
      inv=1/in;
  endsub;
run;
```

```
options cmplib=work.MySubs;
data test;
  wd = day_date('23feb2017'd,'DAYS'); output;
  wd = day_date(1,'YEARS'); output; * makes no sense ;
  n  = 3; wd = .;
  call inverse(n,reciprocal); output;
run;
```

	wd	n	reciprocal
1	5	.	.
2	7	.	.
3	.	3	0.3333333333

PROC FCMP (cont)

PROC FCMP routines can be recursive, another example from the doco (which shows some non-base tricks):

```
proc fcmp outlib=work.funcs.math;
   subroutine allpermk(n, k);
      array scratch[1] / nosymbols;
      call dynamic_array(scratch, n);
      call permk(n, k, scratch, 1,0);
   endsub;

subroutine permk(n, k, scratch[*], m, i);
   outargs scratch;
   if m-1=n then do;
      if i=k then
         put scratch[*];
      end;
   else do;
      scratch[m]=1;
      call permk(n, k, scratch, m+1, i+1);
      scratch[m]=0;
      call permk(n, k, scratch, m+1, i);
   end;
endsub;
run;
quit;

options cmplib=work.funcs;
data _null_;
  call allpermk(5,3);
run;
```

```
23          options cmplib=work.funcs;
24          data _null_;
25            call allpermk(5,3);
26          run;

1 1 1 0 0
1 1 0 1 0
1 1 0 0 1
1 0 1 1 0
1 0 1 0 1
1 0 0 1 1
0 1 1 1 0
0 1 1 0 1
0 1 0 1 1
0 0 1 1 1
NOTE: DATA statement used (Total process time):
      real time           0.03 seconds
      cpu time            0.03 seconds
```

PROC FCMP (cont)

PROC FCMP is very rich and complicated, but needs a bit of effort to understand, but is well worth it for some specialised functionality (such as building encrypted functions that users can use but can't view).

PROC PROTO can be used to define C language constructs for use in PROC FMCP.

See https://support.sas.com/documentation/onlinedoc/base/91/fcmp.pdf etc for more info.

The DOSUB and DOSUBL functions

The DOSUB functions allows for the immediate execution (and return) of a command *within* a currently running data step without have to wait for the step end, unlike a call execute which stacks code for execution.

DOSUB takes a quoted literal string which is a file reference containing code to be executed, and DOSUBL takes (only) a literal string of the code to be executed.

For example:

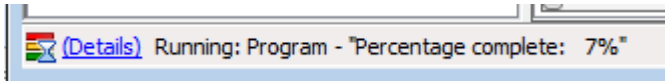
```
data dosubtst;
  rc1 = dosubl('data tst; a=42; run;');
  rc2 = dosubl('%runcode(parm);');
run;
```

Most uses would be to call a macro for convenience. A return code of 0 means the code could be executed, non zero not.

DOSUB and DOSUBL (cont)

This is an example of using a SYSECHO global statement to keep a running update of percent complete in the EG status bar.

```
%macro display_pct_complete(totalrecs,dataname=,by_pct=1,delay=0,incr_var=_n_);
  %* Utility macro to display the % complete ;
  %let decimal_places = %length(%scan(0&by_pct,2,.));
  %let decimal_adj    = %eval(10**&decimal_places);
  %let display_pct    = %eval(3+&decimal_places+(1 and &decimal_places)).&decimal_places;
  if (int(100*&decimal_adj*(&incr_var-1)/(&totalrecs)) ne int(100*&decimal_adj*&incr_var/(&totalrecs))) or
(&incr_var = 1) then do;
    if mod(int(100*&decimal_adj*(&incr_var-1)/(&totalrecs)),&by_pct*&decimal_adj) = 0 then do; * Report on
each complete by_pct ;
      drop _rc;
      _rc = dosubl(cat('SYSECHO "Percentage complete: ', "&dataname", put(int(100*&decimal_adj*(&incr_var-
1)/(&totalrecs))/&decimal_adj,&display_pct), '%";'));
      call sleep(&delay,1); * A delay for fast code ;
    end;
  end
%mend display_pct_complete;
```



DOSUB and DOSUBL (cont)

This opens the possibility of executing global statements such as LIBNAME, macros and data step(s) within the code and then accessing the results (via macro variables) or via the OPEN, FETCH and CLOSE functions.

There are plenty of examples of interesting usage on the net, for example:
http://support.sas.com/resources/papers/proceedings12/227-2012.pdf
https://support.sas.com/resources/papers/proceedings13/032-2013.pdf

Over is an example of how it can be used to create recursive code (but this can be dangerous!)

DOSUB and DOSUBL (cont)

Just for later perusal, (overly complicated) recursive code to calculate a factorial:

```
filename code temp;

data _null_;
  file code;

  put 'data _null_;';
  put '  x = input(symget("parm"),32.);';
  put '  y = coalesce(input(symget("control"),32.),1);';

  put '  if x > 0 then do;';
  put '    call symputx("control",x*y);';
  put '    call symputx("parm"   ,x-1);';
  put '    rc = dosub("code");';
  put '  end;';
  put '  else';
  put '    call symputx("result",y);';
  put 'run;';
run;

data factorial;
  x  = 5;
  call symputx('parm'   ,x);
  call symputx('control',.);
  rc = dosub('code');
  y  = input(symget('result'),32.);
run;
```

FACTORIAL

Filter and Sort | Query Builder | Data | Export | Send

	x	rc	y
1	5	0	120