Fall 2014 OASUS Q&A

The following answers are provided to the benefit of the OASUS Users Group and are not meant to replace SAS Technical Support. The Enterprise Guide project and the PDF document are available on the OASUS web site (www.oasus.ca).

Question 1 : I need an efficient way to split a large SAS file into many files based on the value of a variable on the input file. What do you suggest?

Splitting a SAS file based on a value of a input variable is similar to By-Group processing which is very common in SAS. One would wish that a PROC is readly available to generate one file per By-Group. Unfortunetaly, this is not the case... however the SAS programming language provides many facilities to implement such a process fairly easily. Depending on the size of the input dataset and the number of By-Groups, one may or may not need to reduce the number of passes against the input dataset. Two methods are proposed (both assumed that the input is sorted on the By-Group variable):

Using CALL EXECUTE: many passes but efficient for small dataset and especially a small number of By-Groups.

```
/* Macro that creates creates one By-Group file */
%macro break(byval);

      data Group1_&byval;
            set sortedsplit(where=(groupid="&byval"));
      run;

%mend;

/* sort, necessary for By-Group processing */
proc sort  data=work.inputsplit out=sortedsplit;
      by groupid;

      /* Use CALL EXECUTE to generate the required
calls */
      /* break macr.   */
data _null_;
      set sortedsplit;
      by groupid;

      if first.groupid then
            call
execute('%break('||trim(groupid)||')');
run;
```

Using Hash tables: one pass and very efficient for larger dataset with many By-Groups.

```
/* Sort, necessary for By Group processing */
proc sort  data=work.inputsplit out=sortedsplit;
      by groupid;
/* Split With on DATA step using Hash tabkles */
data _null_;
      if 0 then set sortedsplit; /* create variable types */

      dcl hash groups ( ordered: 'a');
      groups.definekey ('id');
      groups.definedata ('id','value' );
      groups.definedone ();

      do _n_ = 1 by 1 until (last.groupid);
            set sortedsplit;
            by groupid;
            groups.add();
      end;

      groups.output (dataset: compress('GROUP2_'||groupid));
run;
```

References

Pahmer, Emmy. 2006. "Combining, Combining, Combining, Splitting, Splitting, Splitting". NESUG 2006 Conference. Philadelphia, Pennsylvania. http://www.lexjansen.com/nesug/nesug06/dm/da30.pdf

Question 2 : Is it possible to call a DATA step from another DATA step or even a PROC step from a DATA step ?

Yes it is possible !

The simplest way is to use the functions DOSUB or DOSUBL available in SAS 9.4 (experimental in SAS 9.3).

DOSUB and DOSUBL are BASE SAS functions and submit any code that are passed to them. Contrary to CALL EXECUTE, the code is executed immediatly even within a DATA step that is already running. The code can consist of one or many steps including DATA steps and PROC steps! The difference between DOSUB and DOSUBL is that DOSUB will execute code contained in a file whereas DOSUBL will execute code passed as a string.

DOSUB and DOSUBL return a value of zero if SAS code was able to execute, and return a nonzero value if SAS code was not able to execute.

Call a DATA inside a DATA step

```
data _null_;
      rc = dosubl('data x; y=1; run; %let abc=1;');
      abc=symget('abc');
      put abc=;
run;

data _null_;
      set x;
      put _all_;
run;
```

Call a PROC inside a DATA step

```
data _null_;
  rc = dosubl('proc sql;
                      create table work.regions as
                      select distinct region from
sashelp.shoes;');
  putlog rc=;
run;
```

Use of DOSUBL mimics a PROC FCMP function

```
/* This macro retuns a list of variables contained in the */
/* in the specified dataset. */
%MACRO ExpandVarList(data=_LAST_, var=_ALL_);
      %if %upcase(%superq(data)) = _LAST_ %then
            %let data = &SYSLAST;
      %let rc = %sysfunc(dosubl(%str(
            proc transpose data=&DATA(obs=0)
out=ExpandVarList_temp;
            var &VAR;
            run;
            proc sql noprint;
            select _name_ into :temp_varnames separated by ' '
            from ExpandVarList_temp
            ;
            drop table ExpandVarList_temp;
            quit;
            )));
      &temp_varnames
%MEND ExpandVarList;

/* Use it as a macro"function" */
data newclass(keep=newvar1
%ExpandVarList(data=sashelp.class));
      newvar1=1;
      tempvar=0;
      set sashelp.class;
run;
```

References

Langston, Rick. 2013. "Submitting SAS® Code On The Side". Paper 032-2013. SAS Global Forum 2013, San Francisco, California. http://support.sas.com/resources/papers/proceedings13/032-2013.pdf

Question 3 : I need to post-process the results of PROC MEANS. I know it possible to save the output of PROC MEANS into a SAS dataset and then run a DATA step to produce the final output. Is there a way to achieve this post-processing without writing a file to disk?

It is possible to do so with MP-CONNECT piping. Piping enables you to overlap the execution of SAS data steps and/or certain SAS procedures. This is accomplished by spawning one SAS session to run one data step or proc and pipes its output through a TCP/IP socket as input into another SAS session running another DATA step or PROC. This pipeline can be extended to include any number of steps and can even extend between different physical machines. A good paper on the subject is provided in the references.

MP-CONNECT piping may be overkill to simply post-process an output from PROC MEANS. An interesting alternative is a special view called "Data Step Output View". It enables a PROC step or even a DATA step to write directly into a DATA step view! Data Step Views are usually input views. They are common and well documented. Conversely, output DATA step views are experimental, not very well documented and not officially supported; so use them with care!

Create an Output View

```
/* Set-up an output to be called by PROC MEANS */
data final2 / view=out_vw;
      /* These match exactly the output of PROC MEANS */
      length _TYPE_ _FREQ_ 8 _STAT_ $ 8 height 8 weight 8;

      set out_vw;

      if _STAT_ in ('MIN','MAX','MEAN') then
            hwratio=height/weight;
      else hwratio=.;
run;
```

Use the Output View

```
/* PROC MEANS will automatically invoke the output view to
post-process the data */
proc means data=sashelp.class noprint;
      var height weight;
      output out=out_vw;
run;
```

References

Landry, Leonard. "Using MP Connect". OASUS, Ottawa, Ontario
Available at http://www.oasus.ca/MP-Conne.pdf

Buchecker, Michelle. 2005. "Pipeline Parallelism Performance Practicalities". SUGI 30 Conference. Philadelphia, Pensylvania.
https://support.sas.com/rnd/papers/sugi30/pipeline.pdf

First,Steven. 1997. "Faster SAS® Jobs and Fewer Passes Via DATA Step Views". SUGI 22 Conference. San Diego, California.
http://www2.sas.com/proceedings/sugi22/SYSARCH/PAPER312.PDF

Billings, Thomas E. "Output SAS® DATA Step Views: an Experimental Feature", WUSS 2011 Conference. San Francisco, California.
http://www.wuss.org/proceedings11/Papers_Billings_T_73653.pdf

Question 4 : What is the most efficient way to delete datasets, especially if I have lots of datasets stored in a library?

Deleting a dataset efficiently is normally not a big issue. If there is a handful of datasets stored in a library, the operation is fast and simple. However, if a library contains several thousands datasets, you need to be careful as to what method you will be using.

With PROC DATASETS

Typically, you delete a dataset using PROC DATASETS with the DELETE statement. PROC DATASET is a great tool to perform various operations on datasets and on entire libraries. PROC DATASETS maintains in memory the full list of members of a library. This requires time to set-up the environment before any operation can take place. If you have a library with thousands of datasets, loading the in-memory directory may take several seconds.

```
proc datasets library=mylib memtype=data nolist nowarn;
   delete test100;
quit;
```

With PROC DELETE

If you want to quickly delete datasets in a large library, PROD DELETE can be used. It is a simple procedure that provides a simple function: delete datasets! It is quick: it tries to delete datasets without asking any questions! PROC DELETE has been in-and-out of mainstream SAS but has been re-introduced with full support with SAS 9.4. Although it is fast, it has less features than PROC DATASETS. For example you cannot use wildcards (however name suffixes are available).

```
proc delete  library=mylib data=test500;
   run;
quit;
```

With PROC SQL

If you are not running SAS 9.4 or you are a big fan of PROC SQL, you can use the SQL DROP TABLE statement. It is almost as fast as PROC DELETE but is more cumbersome to delete a large number of tables since it does not support neither name suffixes nor wildcards.

```
proc sql;
   drop table mylib.test1000;
quit;
```

References

Richardson, Brad. 2013. "Optimize Your Delete". Paper 022-2013. SAS Global Forum 2013, San Francisco, California. http://support.sas.com/resources/papers/proceedings13/022-2013.pdf.

Ewing, Daphne. 2006. "PROC DATASETS: Managing Data Efficiently". Paper 190-28. Seattle, Washington. http://www2.sas.com/proceedings/sugi28/190-28.pdf.

Hemedinger, Chris. 2013. "PROC DELETE: it's not dead yet". The SAS Dummy blog. http://blogs.sas.com/content/sasdummy/2013/07/08/proc-delete-its-not-dead-yet