SAS® FORUM
UNITED KINGDOM 2015

Top Coding Tips

Neil Merchant – Technical Specialist - SAS

THE POWER TO KNOW.

Bio

- Work in the ANSWERS team at SAS
  - “Analytics as a Service” and Visual Analytics “Try before you buy”
- SAS user for 12 years
  - Base SAS and O/S integration
  - Worked across Clinical, Finance, Marketing and Energy
  - More recently specialising around “In Memory” and Administration

Agenda

5 Top Coding Tips to improve and automate your code

1. In Memory Processing
2. O/S Manipulation (XCMD)
3. Locking / Updating Datasets
4. Data Dynamic Code
5. Hash Tables

I will be in the quad all day following this presentation.

In Memory Processing

In Memory Processing

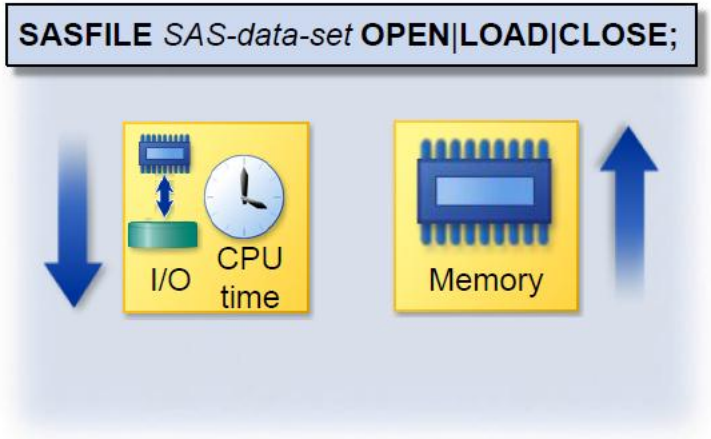
- I/O (disk) is often the biggest constraint on SAS systems.
- Memory is often abundant and cheap.

Imagine the following scenario:-

```
proc freq data=orion.customerdim;
   tables CustomerCountry CustomerType;
run;
proc print data=orion.customerdim noobs;
   where CustomerType='Orion Club Gold members high activity';
   var CustomerID CustomerName CustomerAgeGroup;
run;
proc means data=orion.customerdim mean median max min;
   var CustomerAge;
   class CustomerGroup;
run;
proc tabulate data=orion.customerdim format=8.;
   class CustomerAgeGroup CustomerType;
   table CustomerType All=Total,
         CustomerAgeGroup*n=' ' All=Total*n=' '/rts=45;
run;
```

SASFILE

The SASFILE global statement loads an entire SAS dataset into memory for subsequent DATA and PROC steps. Loading only performs one I/O action.



OPEN	Opens the file and allocates the buffers, but defers reading the data into memory until a procedure or a statement that references the file is executed.
LOAD	Opens the file, allocates the buffers, and reads the data into memory.
CLOSE	Releases the buffers and closes the file.

SASFILE

Before using the dataset use the LOAD (or OPEN) option. Ensure you use the CLOSE option once you have finished with the table to free up system resource.

```
sasfile orion.customerdim load;

proc freq da
   tables Cu
run;
proc print data=orion.customerdim noobs;
   where CustomerType='Orion Club Gold members high acti
   var CustomerID CustomerName CustomerAgeGroup;
run;
proc means data=orion.customerdim mean median max min;
   var CustomerAge;
   class CustomerGroup;
run;
proc tabulate data=orion.customerdim format=8.;
   class CustomerAgeGroup CustomerType;
   table CustomerType All=Total
         Cu                    n=' '/rts=45;
run;
sasfile orion.customerdim close;
```

SASFILE *SAS-data-set* **LOAD;**

SASFILE *SAS-data-set* **CLOSE;**

SASFILE

CAUTION:

- If your dataset is larger than the amount of memory available this can degrade the performance.

RECOMMENDATIONS:

- Run your code with and without the SASFILE option to compare the results.
- SASFILE will be a greater benefit during peak loads on your system.

O/S Manipulation (XCMD)

O/S Manipulation (XCMD)

- SAS has the ability to run O/S scripting. Often referred to as the X command.
- Eg DOS commands in Windows, BASH Commands in Linux/Unix.

X Command Examples

- List Files
- Move Files
- Rename files (including my personal favorite I wrote to run a Windows PowerShell script to get the date taken property of a photo and then update the file names accordingly)
- Compress files
- Run scripts (such as an FTP client script)
- Email Users when they forget to logout of client machine
- ETC

Three Main Methods

Call System (Datastep)

```
options noxwait; /*USE THIS OPTION IF YOU WANT SAS TO CLOSE EACH COMMAND*/

data _null_;
   call system("move /Y c:\test\dataset.sas7bdat c:\live\dataset.sas7bdat");
run;
```

X Command (Raw Code)

```
x "move /Y c:\test\dataset.sas7bdat c:\live\dataset.sas7bdat";
```

Pipe (Filename Statement)

```
/*FILENAME PIPE TO READ IN ALL FILES WITH CSV EXTENSION*/
filename cvs pipe 'dir /b C:\Example\*.csv';
```

Example

```
/*LIBNAME*/
libname example "C:\Example";
/*ALLOW COMMAND TO EXIT ITSELF*/
options noxwait;

/*FILENAME PIPE TO READ IN ALL FILES WITH CSV EXTENSION*/
filename os pipe 'dir /b C:\Example\*.csv';

data csv_filename;
   /*READ IN FILENAME REQUEST AS A PIPE*/
   infile os length=reclen;
   length filename $64.;
   /*READ IN EACH PROCESS*/
   input filename $varying64. reclen;
   /*LOAD FILENAME INTO A MACRO VARIABLE*/
   call symput('fn',filename);
   /*RENAME THE CSV FILE TO SHOW OTHERS I AM USING IT*/
   call system("REN C:\Example\"||left(filename)||" MINE_"|| left(filename));
run;

/*READ IN CSV FILE*/
proc import datafile="C:\Example\MINE_&fn."
                          out=example.csv_file
                          dbms=csv
                          replace;
     getnames=yes;
run;

/*DELETE CSV FILE*/
x "del C:\Example\MINE_&fn.";
```

Example

```
/*LIBNAME*/
libname example "C:\Example";
/*ALLOW COMMAND TO EXIT ITSELF*/
options noxwait;

/*FILENAME PIPE TO READ IN ALL FILES WITH CSV EXTENSION*/
filename os pipe 'dir /b C:\Example\*.csv';

data csv_filename;
   /*READ IN FILENAME REQUEST AS A PIPE*/
   infile os length=reclen;
   length filename $64.;
   /*READ IN EACH PROCESS*/
   input filename $varying64. reclen;
   /*LOAD FILENAME INTO A MACRO VARIABLE*/
   call symput('fn',filename);
   /*RENAME THE CSV FILE TO SHOW OTHERS I AM USING IT*/
   call system("REN C:\Example\"||left(filename)||" MINE_"|| left(filename));
run;

/*READ IN CSV FILE*/
proc import datafile="C:\Example\MINE_&fn."
                          out=example.csv_file
                          dbms=csv
                          replace;
     getnames=yes;
run;

/*DELETE CSV FILE*/
x "del C:\Example\MINE_&fn.";
```

- Options noxwait (and xwait)
- noxwait – means that you don't have to exit your X commend
- xwait – means you have to type exit in your code or manually exit

- xsync and noxsync will wait/not wait for the command to finish running before returning to your SAS session

Example

```
/*LIBNAME*/
libname example "C:\Example";
/*ALLOW COMMAND TO EXIT ITSELF*/
options noxwait;

/*FILENAME PIPE TO READ IN ALL FILES WITH CSV EXTENSION*/
filename os pipe 'dir /b C:\Example\*.csv';

data csv_filename;
   /*READ IN FILENAME REQUEST AS A PIPE*/
   infile os length=reclen;
   length filename $64.;
   /*READ IN EACH PROCESS*/
   input filename $varying64. reclen;
   /*LOAD FILENAME INTO A MACRO VARIABLE*/
   call symput('fn',filename);
   /*RENAME THE CSV FILE TO SHOW OTHERS I AM USING IT*/
   call system("REN C:\Example\"||left(filename)||" MINE_"|| left(filename));
run;

/*READ IN CSV FILE*/
proc import datafile="C:\Example\MINE_&fn."
                          out=example.csv_file
                          dbms=csv
                          replace;
     getnames=yes;
run;

/*DELETE CSV FILE*/
x "del C:\Example\MINE_&fn.";
```

- Create a pipe file name
- Everything in the quotes will run against the OS
- As it is a pipe the results of the command will be passed back into SAS

Example

```
/*LIBNAME*/
libname example "C:\Example";
/*ALLOW COMMAND TO EXIT ITSELF*/
options noxwait;

/*FILENAME PIPE TO READ IN ALL FILES WITH CSV EXTENSION*/
filename os pipe 'dir /b C:\Example\*.csv';

data csv_filename;
   /*READ IN FILENAME REQUEST AS A PIPE*/
   infile os length=reclen;
   length filename $64.;
   /*READ IN EACH PROCESS*/
   input filename $varying64. reclen;
   /*LOAD FILENAME INTO A MACRO VARIABLE*/
   call symput('fn',filename);
   /*RENAME THE CSV FILE TO SHOW OTHERS I AM USING IT*/
   call system("REN C:\Example\"||left(filename)||" MINE_"|| left(filename));
run;

/*READ IN CSV FILE*/
proc import datafile="C:\Example\MINE_&fn."
                          out=example.csv_file
                          dbms=csv
                          replace;
     getnames=yes;
run;

/*DELETE CSV FILE*/
x "del C:\Example\MINE_&fn.";
```

- Next read in the filename statement the same as you would a text file
- The pipe gets submitted against the OS
- The input statement reads the output – this being the name of the file in the directory

Example

```
/*LIBNAME*/
libname example "C:\Example";
/*ALLOW COMMAND TO EXIT ITSELF*/
options noxwait;

/*FILENAME PIPE TO READ IN ALL FILES WITH CSV EXTENSION*/
filename os pipe 'dir /b C:\Example\*.csv';

data csv_filename;
   /*READ IN FILENAME REQUEST AS A PIPE*/
   infile os length=reclen;
   length filename $64.;
   /*READ IN EACH PROCESS*/
   input filename $varying64. reclen;
   /*LOAD FILENAME INTO A MACRO VARIABLE*/
   call symput('fn',filename);
   /*RENAME THE CSV FILE TO SHOW OTHERS I AM USING IT*/
   call system("REN C:\Example\"||left(filename)||" MINE_"|| left(filename));
run;

/*READ IN CSV FILE*/
proc import datafile="C:\Example\MINE_&fn."
                          out=example.csv_file
                          dbms=csv
                          replace;
     getnames=yes;
run;

/*DELETE CSV FILE*/
x "del C:\Example\MINE_&fn.";
```

- Then create a macro variable with the filename so we can save it for later

Example

```
/*LIBNAME*/
libname example "C:\Example";
/*ALLOW COMMAND TO EXIT ITSELF*/
options noxwait;

/*FILENAME PIPE TO READ IN ALL FILES WITH CSV EXTENSION*/
filename os pipe 'dir /b C:\Example\*.csv';

data csv_filename;
   /*READ IN FILENAME REQUEST AS A PIPE*/
   infile os length=reclen;
   length filename $64.;
   /*READ IN EACH PROCESS*/
   input filename $varying64. reclen;
   /*LOAD FILENAME INTO A MACRO VARIABLE*/
   call symput('fn',filename);
   /*RENAME THE CSV FILE TO SHOW OTHERS I AM USING IT*/
   call system("REN C:\Example\"||left(filename)||" MINE_"|| left(filename));
run;

/*READ IN CSV FILE*/
proc import datafile="C:\Example\MINE_&fn."
                          out=example.csv_file
                          dbms=csv
                          replace;
     getnames=yes;
run;

/*DELETE CSV FILE*/
x "del C:\Example\MINE_&fn.";
```

- Next to rename the file to prefix the name of it with “MINE_” so that other users know I am processing it
- Use call system to execute a OS command to process the rename
- The command is built from hardcoded text and the variable name taken from the pipe

Example

```
/*LIBNAME*/
libname example "C:\Example";
/*ALLOW COMMAND TO EXIT ITSELF*/
options noxwait;

/*FILENAME PIPE TO READ IN ALL FILES WITH CSV EXTENSION*/
filename os pipe 'dir /b C:\Example\*.csv';

data csv_filename;
   /*READ IN FILENAME REQUEST AS A PIPE*/
   infile os length=reclen;
   length filename $64.;
   /*READ IN EACH PROCESS*/
   input filename $varying64. reclen;
   /*LOAD FILENAME INTO A MACRO VARIABLE*/
   call symput('fn',filename);
   /*RENAME THE CSV FILE TO SHOW OTHERS I AM USING IT*/
   call system("REN C:\Example\"||left(filename)||" MINE_"|| left(filename));
run;

/*READ IN CSV FILE*/
proc import datafile="C:\Example\MINE_&fn."
                          out=example.csv_file
                          dbms=csv
                          replace;
     getnames=yes;
run;

/*DELETE CSV FILE*/
x "del C:\Example\MINE_&fn.";
```

- Use proc import to import the CSV and turn it onto a dataset
- Use the macro variable created above to get the filename

Example

```
/*LIBNAME*/
libname example "C:\Example";
/*ALLOW COMMAND TO EXIT ITSELF*/
options noxwait;

/*FILENAME PIPE TO READ IN ALL FILES WITH CSV EXTENSION*/
filename os pipe 'dir /b C:\Example\*.csv';

data csv_filename;
   /*READ IN FILENAME REQUEST AS A PIPE*/
   infile os length=reclen;
   length filename $64.;
   /*READ IN EACH PROCESS*/
   input filename $varying64. reclen;
   /*LOAD FILENAME INTO A MACRO VARIABLE*/
   call symput('fn',filename);
   /*RENAME THE CSV FILE TO SHOW OTHERS I AM USING IT*/
   call system("REN C:\Example\"||left(filename)||" MINE_"|| left(filename));
run;

/*READ IN CSV FILE*/
proc import datafile="C:\Example\MINE_&fn."
                          out=example.csv_file
                          dbms=csv
                          replace;
     getnames=yes;
run;

/*DELETE CSV FILE*/
x "del C:\Example\MINE_&fn.";
```

- Finally remove the raw CSV file so that it is not processed again
- Use standalone X command to submit a delete statement
- Again the filename is based on the macro variable

X COMMAND

CAUTION:

- X Commend is turned off by default. It can be turned on in the metadata.
- It does give access to O/S level commands and the file systems so ensure your server is locked down correctly.

WORK AROUNDS:

- Look at sashelp.v* tables. These contain information about your SAS session such as all SAS Libnames and Datasets

Locking

Locking

Nobody wants to see this in there log:-

```
ERROR: A lock is not available for X.SAMPLE2.DATA.
ERROR: Lock held by process 9062.
NOTE: The SAS System stopped processing this step because of errors.
```

These occur if someone is updating the dataset you need, or if someone else is using the dataset you want to update.

%TryLock and %CheckLock Macros

- TryLock can be found with an internet search (or come see me in the quad). (http://www.lexjansen.com/pharmasug/2005/posters/po33.pdf).
- Macros to pause (sleep) your code until you can access the dataset in question.
- Use trylock if you want to update the dataset. Remember to clear the lock when you are done.
- CheckLock can be created by coping TryLock but macro loop removed. To be used when you want to read a dataset (i.e. you don't have to lock other users out).

***Caveat**: SAS does not warranty this code so use it at your discretion.*

%TryLock

```
%macro trylock(member,timeout=3600,retry=10);
   /*GET CURRENT TIME FOR TIME OUT HANDLE*/
   %let starttime = %sysfunc(datetime());

   /*LOOP UNTIL A LOCK HAS BEEN GRANTED OR THE MACRO TIMES OUT*/
   %do %until(&syslckrc <= 0
       or %sysevalf(%sysfunc(datetime()) > (&starttime + &timeout)));

      data _null_;
         dsid = 0;
         /*LOOP UNTIL DATASET OPEN OR TIMEOUT OCCURS*/
         do until (dsid > 0 or datetime() > (&starttime + &timeout));
            /*TRY AND OPEN DATASET - WILL RETURN A NUMBER GT 0
            IF IT CAN OPEN (AND HENCE ISNT NOT LOCKED)*/
            dsid = open("&member");
            /*IF DSID IS 0 UNABLE TO OPEN SO SLEEP FOR THE RETRY TIME*/
            if (dsid = 0) then rc = sleep(&retry);
         end;
         /*CLOSE THE DATASET*/
         if (dsid > 0) then rc = close(dsid);
      run;
      lock &member; /*SAS 9.4 YOU CAN ADD NOMSG COMMAND TO STOP ERRORS*/
   %end;
%mend trylock;
```

%TryLock

```
%macro trylock(member,timeout=3600,retry=10);
   /*GET CURRENT TIME FOR TIME OUT HANDLE*/
   %let starttime = %sysfunc(datetime());

   /*LOOP UNTIL A LOCK HAS BEEN GRANTED OR THE MACRO TIMES OUT*/
   %do %until(&syslckrc <= 0
       or %sysevalf(%sysfunc(datetime()) > (&starttime + &timeout)));

      data _null_;
         dsid = 0;
         /*LOOP UNTIL DATASET OPEN OR TIMEOUT OCCURS*/
         do until (dsid > 0 or datetime() > (&starttime + &timeout));
            /*TRY AND OPEN DATASET - WILL RETURN A NUMBER GT 0
            IF IT CAN OPEN (AND HENCE ISNT NOT LOCKED)*/
            dsid = open("&member");
            /*IF DSID IS 0 UNABLE TO OPEN SO SLEEP FOR THE RETRY TIME*/
            if (dsid = 0) then rc = sleep(&retry);
         end;
         /*CLOSE THE DATASET*/
         if (dsid > 0) then rc = close(dsid);
      run;

      lock &member; /*SAS 9.4 YOU CAN ADD NOMSG COMMAND TO STOP ERRORS*/
   %end;
%mend trylock;
```

- Member – Name of the dataset
- Timeout* – How long to try for
- Retry* - How often to check for a lock

- Take start time for timeout function

*Time in seconds for Windows and milliseconds for Linux.

%TryLock

```
%macro trylock(member,timeout=3600,retry=10);
   /*GET CURRENT TIME FOR TIME OUT HANDLE*/
   %let starttime = %sysfunc(datetime());

   /*LOOP UNTIL A LOCK HAS BEEN GRANTED OR THE MACRO TIMES OUT*/
   %do %until(&syslckrc <= 0
       or %sysevalf(%sysfunc(datetime()) > (&starttime + &timeout)));

      data _null_;
         dsid = 0;
         /*LOOP UNTIL DATASET OPEN OR TIMEOUT OCCURS*/
         do until (dsid > 0 or datetime() > (&starttime + &timeout));
            /*TRY AND OPEN DATASET - WILL RETURN A NUMBER GT 0
            IF IT CAN OPEN (AND HENCE ISNT NOT LOCKED)*/
            dsid = open("&member");
            /*IF DSID IS 0 UNABLE TO OPEN SO SLEEP FOR THE RETRY TIME*/
            if (dsid = 0) then rc = sleep(&retry);
         end;
         /*CLOSE THE DATASET*/
         if (dsid > 0) then rc = close(dsid);
      run;

      lock &member; /*SAS 9.4 YOU CAN ADD NOMSG COMMAND TO STOP ERRORS*/
   %end;
%mend trylock;
```

- Marco loop to keep looping until a lock is obtained or the timeout is reached

%TryLock

```
%macro trylock(member,timeout=3600,retry=10);
   /*GET CURRENT TIME FOR TIME OUT HANDLE*/
   %let starttime = %sysfunc(datetime());

   /*LOOP UNTIL A LOCK HAS BEEN GRANTED OR THE MACRO TIMES OUT*/
   %do %until(&syslckrc <= 0
       or %sysevalf(%sysfunc(datetime()) > (&starttime + &timeout)));

      data _null_;
         dsid = 0;
         /*LOOP UNTIL DATASET OPEN OR TIMEOUT OCCURS*/
         do until (dsid > 0 or datetime() > (&starttime + &timeout));
            /*TRY AND OPEN DATASET - WILL RETURN A NUMBER GT 0
            IF IT CAN OPEN (AND HENCE ISNT NOT LOCKED)*/
            dsid = open("&member");
            /*IF DSID IS 0 UNABLE TO OPEN SO SLEEP FOR THE RETRY TIME*/
            if (dsid = 0) then rc = sleep(&retry);
         end;
         /*CLOSE THE DATASET*/
         if (dsid > 0) then rc = close(dsid);
      run;
      lock &member; /*SAS 9.4 YOU CAN ADD NOMSG COMMAND TO STOP ERRORS*/
   %end;
%mend trylock;
```

- Datastep to test the lock
- Another loop – this time in the datastep
- Loops until the dataset can be open or the timeout is reached

%TryLock

```
%macro trylock(member,timeout=3600,retry=10);
   /*GET CURRENT TIME FOR TIME OUT HANDLE*/
   %let starttime = %sysfunc(datetime());

   /*LOOP UNTIL A LOCK HAS BEEN GRANTED OR THE MACRO TIMES OUT*/
   %do %until(&syslckrc <= 0
       or %sysevalf(%sysfunc(datetime()) > (&starttime + &timeout)));

      data _null_;
         dsid = 0;
         /*LOOP UNTIL DATASET OPEN OR TIMEOUT OCCURS*/
         do until (dsid > 0 or datetime() > (&starttime + &timeout));
            /*TRY AND OPEN DATASET - WILL RETURN A NUMBER GT 0
            IF IT CAN OPEN (AND HENCE ISNT NOT LOCKED)*/
            dsid = open("&member");
            /*IF DSID IS 0 UNABLE TO OPEN SO SLEEP FOR THE RETRY TIME*/
            if (dsid = 0) then rc = sleep(&retry);
         end;
         /*CLOSE THE DATASET*/
         if (dsid > 0) then rc = close(dsid);
      run;
      lock &member; /*SAS 9.4 YOU CAN ADD NOMSG COMMAND TO STOP ERRORS*/
   %end;
%mend trylock;
```

- Try to open dataset
- If dataset opens (i.e. no current lock) then dsid will be a positive number
- If in use then dsid will be equal to 0 and code sleeps and the loop runs again

%TryLock

```
%macro trylock(member,timeout=3600,retry=10);
   /*GET CURRENT TIME FOR TIME OUT HANDLE*/
   %let starttime = %sysfunc(datetime());

   /*LOOP UNTIL A LOCK HAS BEEN GRANTED OR THE MACRO TIMES OUT*/
   %do %until(&syslckrc <= 0
       or %sysevalf(%sysfunc(datetime()) > (&starttime + &timeout)));

      data _null_;
         dsid = 0;
         /*LOOP UNTIL DATASET OPEN OR TIMEOUT OCCURS*/
         do until (dsid > 0 or datetime() > (&starttime + &timeout));
            /*TRY AND OPEN DATASET - WILL RETURN A NUMBER GT 0
            IF IT CAN OPEN (AND HENCE ISNT NOT LOCKED)*/
            dsid = open("&member");
            /*IF DSID IS 0 UNABLE TO OPEN SO SLEEP FOR THE RETRY TIME*/
            if (dsid = 0) then rc = sleep(&retry);
         end;
         /*CLOSE THE DATASET*/
         if (dsid > 0) then rc = close(dsid);
      run;
      lock &member; /*SAS 9.4 YOU CAN ADD NOMSG COMMAND TO STOP ERRORS*/
   %end;
%mend trylock;
```

- Once dataset no longer locked
- Close the connection to the dataset
- Run the lock statement

- If user somehow re-locks the table the syslckrc is not set the macro loop starts again

- 9.4 suppresses ERROR: message

%CheckLock

```
%macro checklock(member,timeout=3600,retry=10);
   /*GET CURRENT TIME FOR TIME OUT HANDLE*/
   %let starttime = %sysfunc(datetime());

   /*LOOP UNTIL A LOCK HAS BEEN GRANTED OR THE MACRO TIMES OUT*/
   %do %until(&syslckrc <= 0
         or %sysevalf(%sysfunc(datetime()) > (&starttime + &timeout)));

      data _null_;
         dsid = 0;
         /*LOOP UNTIL DATASET OPEN OR TIMEOUT OCCURS*/
         do until (dsid > 0 or datetime() > (&starttime + &timeout));
            /*TRY AND OPEN DATASET - WILL RETURN A NUMBER GT 0
            IF IT CAN OPEN (AND HENCE ISNT NOT LOCKED)*/
            dsid = open("&member");
            /*IF DSID IS 0 UNABLE TO OPEN SO SLEEP FOR THE RETRY TIME*/
            if (dsid = 0) then rc = sleep(&retry);
         end;
         /*CLOSE THE DATASET*/
         if (dsid > 0) then rc = close(dsid);
      run;
      lock &member; /*SAS 9.4 YOU CAN ADD NOMSG COMMAND TO STOP ERRORS*/
   %end;
%mend checklock;
```

- Check Lock is exactly the same but without the macro loop to re-loop if lock is not obtained
- No need to clear once done as nothing is locked.

Linux Workaround

In Linux you can manipulate the file system to get around locking issues when updating tables:-

```
/*LIBNAME*/
libname warehous "/ukans/warehouse/default";

/*WRITE THE DATASET THAT NEEDS UPDATING OUT TO A DIFFERENT NAME TO THE ONE YOU WANT TO REPLACE*/
data warehous.example_new;
   set example;
run;

/*USE X COMMAND TO REPLACE THE OLD DATASET WITH THE NEW ONE*/
X "mv /ukans/warehouse/default/example_new.sas7bdat /ukans/warehouse/default/example.sas7bdat";
```

- Any body who references the dataset after the mv takes use the new version.
- Disk Space is freed up when last person finishes using old version.

***Caveat**: This is something I discovered and not a SAS approved method.*

Other hints and tips

- Wait for a table to update by checking proc contents CRDATE or MODATE variable
- Place a trigger file beside your table which is created when it is updated

Data Dynamic Code

Data Dynamic Code

- How do you write code for distinct data items when new data items are always being added?
- What if you don’t know how many input files will exist when you run your code?

Use PROC SQL, Macro Variables and Loops to build powerful and dynamic code.

Example 1

- You have a dataset with the Makes of cars.
- You want to split the data so that you have each make in a separate dataset.
- However new Makes are being added to the source data every day.

How do you do this without a long if-then-else clause that you have to change daily?

	Make	Model
1	Acura	MDX
2	Acura	RSX Type S 2dr
3	Acura	TSX 4dr
4	Acura	TL 4dr
5	Acura	3.5 RL 4dr
6	Acura	3.5 RL w/Naviga...
7	Acura	NSX coupe 2dr...
8	Audi	A4 1.8T 4dr
9	Audi	A4 1.8T converti...
10	Audi	A4 3.0 4dr
11	Audi	A4 3.0 Quattro 4...
12	Audi	A4 3.0 Quattro 4...
13	Audi	A6 3.0 4dr
14	Audi	A6 3.0 Quattro 4...
15	Audi	A4 3.0 convertib...
16	Audi	A4 3.0 Quattro c...
17	Audi	A6 2.7 Turbo Qu...
18	Audi	A6 4.2 Quattro 4...
19	Audi	A8 L Quattro 4dr
20	Audi	S4 Quattro 4dr
21	Audi	RS 6 4dr
22	Audi	TT 1.8 convertib...
23	Audi	TT 1.8 Quattro 2...
24	Audi	TT 3.2 coupe 2d...
25	Audi	A6 3.0 Avant Qu...
26	Audi	S4 Avant Quattro
27	BMW	X3 3.0i
28	BMW	X5 4.4i
29	BMW	325i 4dr
30	BMW	325Ci 2dr
31	BMW	325Ci convertibl...
32	BMW	325xi 4dr

Example 1

```
proc sql noprint;
   /*SELECT ALL UNQUIE MODELS*/
   select distinct make
   /*PUT THE NAMES OF THE MODELS INTO */
   into: make1-
   /*FROM THE DATASET IN QUESTION*/
   from cars;
quit;

/*THIS WILL BUILD A UNIQUE MACRO VARIABLE FOR EACH MAKE EG:-
  MAKE1=Acura
  MAKE2=Audi
  ETC
  ALONG WITH ANOTHER VARIABLE CALLED SQLOBS WHICH CONTAINS THE NUMBER
  OF RECORDS - IN THIS CASE 40*/

/*MACRO TO CREATE INDIVIDUAL DATASETS*/
%macro split_cars;
   /*LOOP FROM 1 TO SQLOBS (40)*/
   %do i=1 %to &sqlobs;
      /*CREATE INDIVIDUAL DATASET*/
      data &&make&i..;
         /*WHERE THE MAKE IS EQUAL TO THE MAKE OF THE I VALUE IN THIS LOOP*/
         set cars(where=(make="&&make&i..")); 
      run;
   %end;
%mend split_cars;
%split_cars;
```

Example 1

```
proc sql noprint;
   /*SELECT ALL UNQUIE MODELS*/
   select distinct make
   /*PUT THE NAMES OF THE MODELS INTO */
   into: make1-
   /*FROM THE DATASET IN QUESTION*/
   from cars;
quit;

/*THIS WILL BUILD A UNIQUE MACRO VARIABLE FOR EACH MAKE EG:-
  MAKE1=Acura
  MAKE2=Audi
  ETC
  ALONG WITH ANOTHER VARIABLE CALLED SQLOBS WHICH CONTAINS THE NUMBER
  OF RECORDS - IN THIS CASE 40*/

/*MACRO TO CREATE INDIVIDUAL DATASETS*/
%macro split_cars;
   /*LOOP FROM 1 TO SQLOBS (40)*/
   %do i=1 %to &sqlobs;
      /*CREATE INDIVIDUAL DATASET*/
      data &&make&i..;
         /*WHERE THE MAKE IS EQUAL TO THE MAKE OF THE I VALUE IN THIS LOOP*/
         set cars(where=(make="&&make&i..")); 
      run;
   %end;
%mend split_cars;
%split_cars;
```

- Select all the distinct makes.
- From the cars dataset.
- Don’t print the output.

Example 1

```
proc sql noprint;
   /*SELECT ALL UNQUIE MODELS*/
   select distinct make
   /*PUT THE NAMES OF THE MODELS INTO */
   into: make1-
   /*FROM THE DATASET IN QUESTION*/
   from cars;
quit;

/*THIS WILL BUILD A UNIQUE MACRO VARIABLE FOR EACH MAKE EG:-
  MAKE1=Acura
  MAKE2=Audi
  ETC
  ALONG WITH ANOTHER VARIABLE CALLED SQLOBS WHICH CONTAINS THE NUMBER
  OF RECORDS - IN THIS CASE 40*/

/*MACRO TO CREATE INDIVIDUAL DATASETS*/
%macro split_cars;
   /*LOOP FROM 1 TO SQLOBS (40)*/
   %do i=1 %to &sqlobs;
      /*CREATE INDIVIDUAL DATASET*/
      data &&make&i..;
         /*WHERE THE MAKE IS EQUAL TO THE MAKE OF THE I VALUE IN THIS LOOP*/
         set cars(where=(make="&&make&i.."));
      run;
   %end;
%mend split_cars;
%split_cars;
```

- Put these values into marco variables
- MAKE1=Acura
- MAKE2=Audi
- …
- MAKE40=Volvo

Example 1

```
proc sql noprint;
   /*SELECT ALL UNQUIE MODELS*/
   select distinct make
   /*PUT THE NAMES OF THE MODELS INTO */
   into: make1- :make100
   /*FROM THE DATASET IN QUESTION*/
   from cars;
quit;

/*THIS WILL BUILD A UNIQUE MACRO VARIABLE FOR EACH MAKE EG:-
  MAKE1=Acura
  MAKE2=Audi
  ETC
  ALONG WITH ANOTHER VARIABLE CALLED SQLOBS WHICH CONTAINS THE NUMBER
  OF RECORDS - IN THIS CASE 40*/

/*MACRO TO CREATE INDIVIDUAL DATASETS*/
%macro split_cars;
   /*LOOP FROM 1 TO SQLOBS (40)*/
   %do i=1 %to &sqlobs;
      /*CREATE INDIVIDUAL DATASET*/
      data &&make&i..;
         /*WHERE THE MAKE IS EQUAL TO THE MAKE OF THE I VALUE IN THIS LOOP*/
         set cars(where=(make="&&make&i..")); 
      run;
   %end;
%mend split_cars;
%split_cars;
```

- You can put a limit on the upper number of variables if you wish

Example 1

```
proc sql noprint;
   /*SELECT ALL UNQUIE MODELS*/
   select distinct make
   /*PUT THE NAMES OF THE MODELS INTO */
   into: make1-
   /*FROM THE DATASET IN QUESTION*/
   from cars;
quit;

/*THIS WILL BUILD A UNIQUE MACRO VARIABLE FOR EACH MAKE EG:-
  MAKE1=Acura
  MAKE2=Audi
  ETC
  ALONG WITH ANOTHER VARIABLE CALLED SQLOBS WHICH CONTAINS THE NUMBER
  OF RECORDS - IN THIS CASE 40*/

/*MACRO TO CREATE INDIVIDUAL DATASETS*/
%macro split_cars;
   /*LOOP FROM 1 TO SQLOBS (40)*/
   %do i=1 %to &sqlobs;
      /*CREATE INDIVIDUAL DATASET*/
      data &&make&i..;
         /*WHERE THE MAKE IS EQUAL TO THE MAKE OF THE I VALUE IN THIS LOOP*/
         set cars(where=(make="&&make&i.."));
      run;
   %end;
%mend split_cars;
%split_cars;
```

- Macro variable called sqlobs is created with the count of the number of macro variables created.
- Used to drive a macro loop.

Example 1

```
proc sql noprint;
   /*SELECT ALL UNQUIE MODELS*/
   select distinct make
   /*PUT THE NAMES OF THE MODELS INTO */
   into: make1-
   /*FROM THE DATASET IN QUESTION*/
   from cars;
quit;

/*THIS WILL BUILD A UNIQUE MACRO VARIABLE FOR EACH MAKE EG:-
  MAKE1=Acura
  MAKE2=Audi
  ETC
  ALONG WITH ANOTHER VARIABLE CALLED SQLOBS WHICH CONTAINS THE NUMBER
  OF RECORDS - IN THIS CASE 40*/

/*MACRO TO CREATE INDIVIDUAL DATASETS*/
%macro split_cars;
   /*LOOP FROM 1 TO SQLOBS (40)*/
   %do i=1 %to &sqlobs;
      /*CREATE INDIVIDUAL DATASET*/
      data &&make&i..;
         /*WHERE THE MAKE IS EQUAL TO THE MAKE OF THE I VALUE IN THIS LOOP*/
         set cars(where=(make="&&make&i.."));
      run;
   %end;
%mend split_cars;
%split_cars;
```

- We then create a macro loop to loop from i to sqlobs (1 to 40)

Example 1

```
proc sql noprint;
   /*SELECT ALL UNQUIE MODELS*/
   select distinct make
   /*PUT THE NAMES OF THE MODELS INTO */
   into: make1-
   /*FROM THE DATASET IN QUESTION*/
   from cars;
quit;

/*THIS WILL BUILD A UNIQUE MACRO VARIABLE FOR EACH MAKE EG:-
  MAKE1=Acura
  MAKE2=Audi
  ETC
  ALONG WITH ANOTHER VARIABLE CALLED SQLOBS WHICH CONTAINS THE NUMBER
  OF RECORDS - IN THIS CASE 40*/

/*MACRO TO CREATE INDIVIDUAL DATASETS*/
%macro split_cars;
   /*LOOP FROM 1 TO SQLOBS (40)*/
   %do i=1 %to &sqlobs;
      /*CREATE INDIVIDUAL DATASET*/
      data &&make&i..;
         /*WHERE THE MAKE IS EQUAL TO THE MAKE OF THE I VALUE IN THIS LOOP*/
         set cars(where=(make="&&make&i.."));
      run;
   %end;
%mend split_cars;
%split_cars;
```

How does && work?

- Assuming in the first iteration of loop so i=1.
- During compile stage, SAS will change two & into one.
- Whereas if there is 1 & it will resolve.
- Hence
  -&&make&i..
  -&make1.
  -Acura

Example 1

```
proc sql noprint;
   /*SELECT ALL UNQUIE MODELS*/
   select distinct make
   /*PUT THE NAMES OF THE MODELS INTO */
   into: make1-
   /*FROM THE DATASET IN QUESTION*/
   from cars;
quit;

/*THIS WILL BUILD A UNIQUE MACRO VARIABLE FOR EACH MAKE EG:-
  MAKE1=Acura
  MAKE2=Audi
  ETC
  ALONG WITH ANOTHER VARIABLE CALLED SQLOBS WHICH CONTAINS THE NUMBER
  OF RECORDS - IN THIS CASE 40*/

/*MACRO TO CREATE INDIVIDUAL DATASETS*/
%macro split_cars;
   /*LOOP FROM 1 TO SQLOBS (40)*/
   %do i=1 %to &sqlobs;
      /*CREATE INDIVIDUAL DATASET*/
      data &&make&i..;
         /*WHERE THE MAKE IS EQUAL TO THE MAKE OF THE I VALUE IN THIS LOOP*/
         set cars(where=(make="&&make&i.."));
      run;
   %end;
%mend split_cars;
%split_cars;
```

- Set and where statement to finish off the code and to being in relevant records

Example 1 (Reworked)

```
/*THIS TIME CREATE TWO MACRO VARIABLES - ONE FORMATTED TO BE SAS DATASET NAMING SAFE*/
proc sql noprint;
   select distinct make,
                   compress(make,,'KN')
   into: make1-
       : makec1-

   from cars;
quit;

/*GOOD PRACTICE TO STORE SQLOBS VALUE INTO ANOTHER VARIABLE
  IN CASE LOOP CODE OVERWRITES IT*/
%let make_count=&sqlobs.;

%put _global_;

/*MACRO TO CREATE INDIVIDUAL DATASETS*/
%macro split_cars;
   /*LOOP AROUND ONCE PER CAR TO CREATE A DATASET*/
   data %do i=1 %to &make_count.;
            &&makec&i..
        %end; ;
      /*READ IN WHOLE TABLE IN ONE GO*/
      set cars;

      /*LOOP AROUND ONCE PER CAR TO CREATE IF STATEMENT*/
      %do i=1 %to &make_count.;
         /*TO MAKE CODE EVEN BETTER ADD AN ELSE STATEMENT IF IT ISNT THE FIRST MAKE*/
         %if &i^=1 %then %do;
            else
         %end;
         /*ONE IF THEN OUTPUT STATEMENT IS CREATED FOR EACH MAKE*/
         if make="&&make&i" then output &&makec&i..;
      %end;
   run;
%mend split_cars;

%split_cars;
```

- Added new item to select and into.
- Create a copy of sqlobs.
- Do all the processing in 1 datastep rather than 40+.
- Added a loop for data statement.
- Added loop to create if-then-else code.

Example 2

```
libname example "C:\Example";
/*FILENAME PIPE TO READ IN ALL FILES WITH CSV EXTENSION*/
filename csv pipe 'dir /b C:\Example\*.csv';
/*ALLOW COMMAND TO EXIT ITSELF*/
options noxwait;
data csv;
   /*READ IN FILENAME REQUEST AS A PIPE*/
   infile csv length=reclen;
   length filename $64.;

   /*READ IN EACH PROCESS*/
   input filename $varying64. reclen;
run;

/*LOAD ALL CSV VALUES TO MACRO VARIABLES*/
proc sql noprint;
   select filename
   into: f1-:f500
   from csv;
quit;

%let file_count=&sqlobs.;

%macro import_all_csvs;
    %do i=1 %to &file_count.;

       /*READ IN CSV FILE*/
        proc import datafile="C:\Example\&&f&i.."
                    out=example.%sysfunc(compress("&&f&i..",,'KN'))
                    dbms=csv
                    replace;
             getnames=yes;
        run;

        /*IMPORTANT NOXWAIT IS SET HERE ELSE YOU COULD BE
          CLOSING HUNDREDS OF WINDOWS*/
        x "del C:\Example\&&f&i..";
    %end;
%mend import_all_csvs;
options mprint;
%import_all_csvs;
```

- Based on XCMD example but with multiple files.
- Read in a dynamic number of files.

Hash Tables

Hash Tables

- Lookup table in Memory.
  - Faster in-memory processing.
  - Outperforms traditional DATA step or Proc SQL techniques.
  - Merge tables without sorting.
  - Fast, flexible table-lookup.

Hash Tables

- What would you use them for?
  - Bring in demographic data based on Postcode
  - Find the latest version (for example a users latest credit card number)
  - Data that you want to join needs pre-summarisation – use relevant rows
  - Check Lists
  - Dynamic duplicates

Hash Tables – Inner Join (1:1 or Many:1)

```
data Transaction_Demographic;
   length age_desc $10
          Income_desc $20
          Homeowner_Desc $20;

   if _N_=1 then do;
      declare hash h(dataset:'work.demographic');
                   h.defineKey('household_key');
                   h.defineData('age_desc',
                                'income_desc',
                                'homeowner_desc');
                   h.defineDone();
   end;

   set transaction_data;

   if h.find()=0 then
      output;
run;
```

Hash Tables – Inner Join (1:1 or Many:1)

```
data Transaction_Demographic;
   length age_desc $10
          Income_desc $20
          Homeowner_Desc $20;

   if _N_=1 then do;
      declare hash h(dataset:'work.demographic');
                   h.defineKey('household_key');
                   h.defineData('age_desc',
                                'income_desc',
                                'homeowner_desc');
                   h.defineDone();
   end;

   set transaction_data;

   if h.find()=0 then
      output;
run;
```

- Start with a standard datastep.
- It is recommended the larger table (or the table with many of the same keys) is placed in the set statement.
- With this example the hash table must have unique keys.

Hash Tables – Inner Join (1:1 or Many:1)

```
data Transaction_Demographic;
   length age_desc $10
          Income_desc $20
          Homeowner_Desc $20;

   if _N_=1 then do;
      declare hash h(dataset:'work.demographic');
                   h.defineKey('household_key');
                   h.defineData('age_desc',
                                'income_desc',
                                'homeowner_desc');
                   h.defineDone();
   end;

   set transaction_data;

   if h.find()=0 then
      output;
run;
```

- Any variables that are being brought in from the left table need to be defined in a length statement.
- The key does not need a length statement as it is already defined in the set dataset.

Hash Tables – Inner Join (1:1 or Many:1)

```
data Transaction_Demographic;
   length age_desc $10
          Income_desc $20
          Homeowner_Desc $20;

   if _N_=1 then do;
      declare hash h(dataset:'work.demographic');
                   h.defineKey('household_key');
                   h.defineData('age_desc',
                                'income_desc',
                                'homeowner_desc');
                   h.defineDone();
   end;

   set transaction_data;

   if h.find()=0 then
      output;
run;
```

- The hash table needs to be loaded before any other code is run.
- Encased in _n_=1 if statement to ensure this happens.
- Without if statement it will load once per observation!

Hash Tables – Inner Join (1:1 or Many:1)

```
data Transaction_Demographic;
   length age_desc $10
          Income_desc $20
          Homeowner_Desc $20;

   if _N_=1 then do;
      declare hash h(dataset:'work.demographic');
                  h.defineKey('household_key');
                  h.defineData('age_desc',
                               'income_desc',
                               'homeowner_desc');
                  h.defineDone();
   end;

   set transaction_data;

   if h.find()=0 then
      output;
run;
```

- First we declare the hash table.
- We are naming it ‘h’.
- Within the brackets are the hash table options.
- In this case we are just defining the dataset.

Hash Tables – Inner Join (1:1 or Many:1)

```
data Transaction_Demographic;
   length age_desc $10
          Income_desc $20
          Homeowner_Desc $20;

   if _N_=1 then do;
      declare hash h(dataset:'work.demographic');
                   h.defineKey('household_key');
                   h.defineData('age_desc',
                                'income_desc',
                                'homeowner_desc');
                   h.defineDone();
   end;

   set transaction_data;

   if h.find()=0 then
      output;
run;
```

- Next we need to define the key.
- The key variable must be the same name and type in both transaction and demographic datasets.

Hash Tables – Inner Join (1:1 or Many:1)

```
data Transaction_Demographic;
   length age_desc $10
          Income_desc $20
          Homeowner_Desc $20;

   if _N_=1 then do;
      declare hash h(dataset:'work.demographic');
                   h.defineKey('household_key');
                   h.defineData('age_desc',
                                'income_desc',
                                'homeowner_desc');
                   h.defineDone();
   end;

   set transaction_data;

   if h.find()=0 then
      output;
run;
```

- Next we define the data to be brought in.
- As mentioned the metadata about these variables is not imported so has to be loaded in a length statement.
- This step is optional if only want to do a join on the key.

Hash Tables – Inner Join (1:1 or Many:1)

```
data Transaction_Demographic;
   length age_desc $10
          Income_desc $20
          Homeowner_Desc $20;

   if _N_=1 then do;
      declare hash h(dataset:'work.demographic');
                   h.defineKey('household_key');
                   h.defineData('age_desc',
                                'income_desc',
                                'homeowner_desc');
                   h.defineDone();
   end;

   set transaction_data;

   if h.find()=0 then
      output;
run;
```

- To finish the declare statement we need to define the hash table complete.
- The dataset is loaded into memory at this point.

Hash Tables – Inner Join (1:1 or Many:1)

```
data Transaction_Demographic;
   length age_desc $10
          Income_desc $20
          Homeowner_Desc $20;

   if _N_=1 then do;
      declare hash h(dataset:'work.demographic');
                   h.defineKey('household_key');
                   h.defineData('age_desc',
                                'income_desc',
                                'homeowner_desc');
                   h.defineDone();
   end;

   set transaction_data;

   if h.find()=0 then
      output;
run;
```

- Once the hash table is setup we can do the lookup.
- h.find() will look for the household_key of the current observation within the hash table.
- If the key is found it will bring in the defineData columns to that observation and the function also sends back a return code of 0.
- If the key is not found it will return a non-zero value.
- In this case we are outputting the observation if the key is found (inner join).

Hash Tables – 1:Many Join

```
data Transaction_Demographic(drop=rc);
   length age_desc $10
          Income_desc $20
          Homeowner_Desc $20;
   if _N_=1 then do;
      declare hash h(dataset:'work.demographic',
                     multidata:'YES');
                  h.defineKey('household_key');
                  h.defineData('age_desc',
                               'income_desc',
                               'homeowner_desc');
                  h.defineDone();
   end;

   set transaction_data;

   rc=h.find();
   do while(rc=0);
      output;
      rc=h.find_next();
   end;
run;
```

- If your hash table has to be the “Many” due to the size of the left table.
- Change the declare statement to inform hash table of duplicates
- Introduce a loop to output when a key is found, then find the next key.

Hash Tables – Additional

- Hash Table Iterators
  - No Key needed
  - Move through data one row at a time
  - Define the order of data
- Summarise data (sum a variable)
- Add values
- Remove values

Quad

Quad

1. Any of the 5 topics covered here today
2. Your challenges
3. ANSWERS / VA
4. Views
5. Indexes
6. Efficiencies
7. Dataset Sizes (compress and var lengths)
8. Config options
9. Arrays
10. SAS “V” tables
11. Administration

SAS® FORUM
UNITED KINGDOM 2015

sas | THE POWER TO KNOW.

www.SAS.com