Mocking Data To Support Performance Load Testing

Troy Martin Hughes

ABSTRACT

This text introduces the MOCKDATA macro, which creates sample data sets that can be used in load testing, stress testing, and other performance testing. Users are able to configure the MOCKDATA macro to alter the mock data set that is produced. Parameters include the number of observations, number of character variables, length of character variables, number of numeric variables, highest number saved as a numeric variable, and percentage of variables that are complete. MOCKDATA creates SAS data sets and/or text flat files, thus it is ideal for testing the relative performance of input/output (I/O) functionality of flat files as compared with SAS data sets. Used in coordination with the author's PINCHLOG macro, MOCKDATA is able to mathematically demonstrate the performance advantages (i.e., increased runtime) of certain functionally methods over others. Data mocking is critical when data are sensitive and cannot be used, or data cannot be transferred (but must be tested among various locations), or simply to ensure statistically equivalent data for all types of load and performance testing.

INTRODUCTION

Introduce mocking data…

Introduce load/stress testing (performance, not functional testing for failure)

Describe MOCKDATA at a high-level

Describe two methods of collecting performance metrics in repeated measures analysis, including the pros and cons of each (collecting data in memory vs writing data incrementally to a SAS data set)

Describe some findings at a high-level (such as the tremendous performance advantages of utilizing SAS data set reads/writes over flat file reads/writes).

MOCKDATA MACRO

The MOCKDATA macro creates a user-defined data set and/or flat file .

The complex macro logic can be summarized in Table 1.

Parameter	**Option 1**	**Option 2**	**Option 3**	**Option 4**	**Option 5**	**Option 6**
MAKEDATASET	Y	Y	Y	Y	N	N
MAKEFLATFILE	Y	Y	N	N	Y	Y
RANDOMIZE	Y	N	Y	N	Y	N
1. DATA Step						
Statement						
DATA &dsn;	X	X	X	X	X	
DATA _null_;						X
LENGTH I J Obs;	X	X	X	X	X	X
LENGTH rando;	X		X		X	
rando = rand('uniform')	X		X		X	
OUTPUT;	X	X	X	X	X	
FILE "&flatfile";		X				X

PUT &putstatement;		X				X
2. PROC SORT						
SORT &dsn;						
3. DATA Step						
FILE “&flatfile”;	X				X	
PUT &putstatement;	X				X	

Table 1. Logic Inside %MOCKDATA

The macro is included inside Appendix A.

METADATA METRICS

First, you need to decide what metrics are being created

If this data set isn’t being automated (like occurs within PINCHLOG), then you’ll need to create a base file, such as the following metrics_base.sas7bdat:

```
data metrics_base;
   length machine $20 err 3 cpucount 8 memsize 8 sortsize 8 obs 8 obsuni 8
      charvar 8 charlen 8 numvar 8 numlen 8 pctcomp 8
      dsnfilesize 8 flatfilesize 8;
   format machine $20. err 3. cpucount 8. memsize comma8. sortsize comma8. obs
comma15. obsuni comma15.
      charvar comma8. charlen comma8. numvar comma8. numlen 8. pctcomp 8.
      dsnfilesize comma17.5 flatfilesize comma17.5;
   label machine='Environment' err='SYSCC Code' cpucount='CPUCOUNT'
      memsize='MEMSIZE' sortsize='SORTSIZE' obs='Obs' obsuni='Unique Obs'
      charvar='No of Char Vars' charlen='Char Var Length'
      numvar='No of Numeric Vars' numlen='Numeric Var Length'
      pctcomp='% Complete' dsnfilesize='Data Set File Size (MB)' flatfilesize='Flat
File Size (MB)';
   if ^missing(machine);
run;
```

The metrics data set should always contain return codes that denote whether warnings or runtime errors were encountered. For example, if a data set was created, but then failed to be sorted (to randomize the data) because of an out-of-memory error, this might go unnoticed because the data set still would have been created. This could cause dramatically erroneous results were subsequent processes to measure a SORT, FREQ, or other procedure or process.

TESTMOCK MACRO

The second macro is the TESTMOCK macro that iterates through various combinations to, in this instance, test for size. This macro is listed in Appendix B.

A sample invocation follows:

```
%let dtgstart=%sysfunc(datetime());
proc printto &printtolog new;
run;
%testmock(dsnmetrics=ushris.size_metrics,
   obsstart=1, obsend=4, obsiter=1,
   charvarstart=1, charvarend=4, charvariter=1,
   charlenstart=1, charlenend=4, charleniter=1,
   numvarstart=1, numvarend=4, numvariter=1,
   numlenstart=3, numlenend=6, numleniter=1,
   pctcompstart=70, pctcompend=100, pctcompiter=10,
   flatfile=D:\sas\USHRIS_mainframe\flatfilex.txt, flatfilespace=0,
```

```
   metricsreport=&metricsreport);
proc printto;
run;
%put TESTMOCK completed in %sysevalf(%sysfunc(datetime())-&dtgstart);
```

This sample has 4 X 4 X 4 X 4 X 4 = 4,096 iterations that are completed. Running on 32 GB RAM and 8 CPUs on a desktop, this completes in 230.67 seconds. (NEED TO GET AVERAGES FOR THIS)

However, the macro could be made more efficient were it to reduce the interim DATA step and APPEND procedure that must execute inside each loop.

A different methodology would instead aggregate these metadata metrics in memory rather than in a data set, so that I/O resources are not wasted inside each loop. In this paradigm shift, the only remaining I/O functions become the data set and flat file creation inside the MOCKDATA macro.

The revised TESTMOCKINMEM macro is listed in Appendix C. It has an identical specification and invocation as TESTMOCK:

```
%let dtgstart=%sysfunc(datetime());
proc printto &printtolog new;
run;
%testmockinmem(dsnmetrics=ushris.size_metricsinmem,
   obsstart=1, obsend=4, obsiter=1,
   charvarstart=1, charvarend=4, charvariter=1,
   charlenstart=1, charlenend=4, charleniter=1,
   numvarstart=1, numvarend=4, numvariter=1,
   numlenstart=3, numlenend=6, numleniter=1,
   pctcompstart=70, pctcompend=100, pctcompiter=10,
   flatfile=D:\sas\USHRIS_mainframe\flatfilex.txt, flatfilespace=0,
   metricsreport=&metricsreport);
proc printto;
run;
%put TESTMOCKINMEM completed in %sysevalf(%sysfunc(datetime())-&dtgstart);
```

This method is demonstrably faster and completes the same 7,500 iterations in only 160.11 seconds!!!!! However, two caveats exist while, in some cases, compel against its use.

NEED TO SHOW THE SCALABILITY OF THIS ADVANTAGE OVER TIME!!!

First, because this method saves all data in memory, with sufficient iterations, it can become a memory hog. Thus, if you are load testing (or especially stress testing) high-memory operations, it may not be a good choice. For example, each iteration requires approximately XXXXXXXXX bytes of memory to be saved, and this can add up over time. Thus, if you are stress testing a procedure like SORT, you may want to use the interim DATA step and APPEND procedure.

Second, the benefit of building the metrics data set incrementally is that if you need to stop a test, you can do so at any point, and be confident that the metrics data set will contain all the metrics generated to that point. However, because the in-memory solution builds the metrics data set only after ALL iterations have completed, those iterations must be allowed to finish.

Consider the first scenario in which I'm running load testing on my work desktop over the weekend. If the cleaning crew bumps a cord and terminates power, or if the cybersecurity team restarts my machine as part of the installation of security patches, all that work goes to waste. I don't even know how far the program got until its efforts were stymied.

Consider a second scenario in which you kick off load testing on your desktop overnight. It's morning now, your program is still running, and you can tell from opening the log file that it has another 500,000 iterations to run (and won't be ready for some hours), but you need to look at the current data NOW. Thus, in this scenario, the data you have now are more valuable than having more data later. You kill the program and, if you've built the metrics data set incrementally, you can immediately view the data that have been aggregated. But, if you're using…

Two solutions exist, neither of which are shown here:

1. A single DATA step with a thousand SET data sets is faster than PROC APPEND, so this could be used instead
2. If the in-memory approach is used, and those macro variables are saved to global macro variables, these could be scooped up somehow after a manual termination. However, this still would not prevent the issue of when the machine is forcibly rebooted or loses power and those data are lost.

PINCHLOG

Show a couple examples of how PINCHLOG can be used with either of the above two methodologies (in-data and in-memory).

Mention that whenever possible, system resource utilization metrics should also be extracted. This effectively creates a multidimensional array, which is again why it's useful to store the data as stacked rather than packed format.

APPENDIX A. MOCKDATA MACRO

```
* finds the symbol table entry for A;
%macro findabyte();
%global findabyte;
%local a;
%do findabyte=1 %to 200;
   %let a=A%sysfunc(byte(&findabyte));
   %if %eval(%length(%superq(a))=1) %then %do;
      %end;
   %else %do;
      %if "%superq(a)"="AA" %then %return;
      %end;
   %end;
%mend;


%macro mockdata(obs= /* number of observations */,
   obsuni= /* number of unique observations */,
   charvar=0 /* number of character variables */,
   charlen= /* length of character variables */,
   numvar=0 /* number of numeric variables */,
   numlen= /* length of numeric variables (3 to 8) */,
   pctcomplete=100 /* percent of all fields that are complete */,
   idxcharcomplete=YES /* yes to always make the first charvar complete */,
   idxnumcomplete=YES /* yes to always make the first numvar complete */,
   makedataset=YES /* YES to create a data set */,
   dsn= /* data set name in WORK library in DSN format */,
   makeflatfile=NO /* YES to create a positional flat file */,
   flatfile= /* location, file name, and extension */,
   flatfilespace=1 /* spaces between each column */,
   flatfilemiss=SPACE /* print missing numbers as space, not PERIOD */,
   randomize=YES /* YES or NO to sort the data set randomly */);
%let syscc=0;
%global mockdataRC inputstatement putstatement dsnfilesize flatfilesize;
%let mockdataRC=99;
%let idxcharcomplete=%upcase(&idxcharcomplete);
%let idxnumcomplete=%upcase(&idxnumcomplete);
%let makedataset=%upcase(&makedataset);
%let makeflatfile=%upcase(&makeflatfile);
%let randomize=%upcase(&randomize);
%if &makeflatfile^=YES and &makedataset^=YES %then %return;
%if &charvar=0 and &numvar=0 %then %return;
%if %symexist(findabyte)=0 %then %findabyte;
%let inputstatement=;
%let putstatement=;
%let dsnfilesize=;
%let flatfilesize=;
%local i j space maxnum maxnumwidth numvar dsntemp rc fid;
%if %length(&numvar)>0 %then %do;
   %let maxnum=%sysevalf(32*(256**(&numlen-2))); * largest num (bytewise);
   %let maxnumwidth=%length(%sysfunc(putc(&maxnum,17.))); * scientific note;
   %end;
%else %let numvar=0;
* optionally create PUT and INPUT statements;
%if &makeflatfile=YES %then %do;
   %if %upcase(&flatfilemiss)=SPACE %then %do;
      options missing='';
      %end;
   %else %do;
      options missing.;
      %end;
```

```
   %let col=1;
   %if %eval(&charvar>0) %then %do i=1 %to &charvar;
      %let putstatement=&putstatement @&col char&i;
  %let inputstatement=&inputstatement char&i $ &col - %eval(&col+&charlen-1);
      %let col=%eval(&col+&charlen+&flatfilespace);
      %end;
   %if %eval(&numvar>0) %then %do i=1 %to &numvar;
      %let putstatement=&putstatement @&col num&i;
      %let inputstatement=&inputstatement
         num&i &col - %eval(&col+&maxnumwidth-1);
      %let col=%eval(&col+&maxnumwidth+&flatfilespace);
      %end;
   %end;
* create data set and/or flat file;
%if &makedataset^=YES and &makeflatfile=YES and &randomize^=YES %then %do;
data _null_;
   %end;
%else %do;
data &dsn (drop=i j obs obs2);
   %end;
   length i j obs 8;
%if &randomize=YES %then %do;
   length rando 8;
   %end;
%if &makeflatfile=YES and &randomize^=YES %then %do;
   file "&flatfile";
   %end;
%if %eval(&charvar>0) %then %do;
   array chars $&charlen char1-char&charvar;
   %end;
%if %eval(&numvar>0) %then %do;
   array nums &numlen num1-num&numvar;
   format num1-num&numvar 17.;
   %end;
%let j=%eval(&obsuni-%sysfunc(mod(&obs,&obsuni)));
   do obs=1 to &obsuni;
   %if %eval(&charvar>0) %then %do;
      * create character vars;
      do i=1 to dim(chars);
         chars{i}='';
         do j=1 to &charlen;
            chars{i}=cats(chars{i},byte(int(rand('uniform')*10)+&findabyte));
            end; * only generate letters A to J;
      %if %eval(&pctcomplete<100) and &idxcharcomplete=YES
            %then %do;
         if i ne 1 and (int(rand('uniform')*100)+1) >= &pctcomplete
            then chars{i}='';
            %end;
      %else %if %eval(&pctcomplete<100)%then %do;
         if (int(rand('uniform')*100)+1) >= &pctcomplete
            then chars{i}='';
         %end;
         end;
      %end;
   %if %eval(&numvar>0) %then %do;
      * create numeric vars;
      do i=1 to dim(nums);
      %if &pctcomplete=100 %then %do;
         nums{i}=int(rand('uniform')*&maxnum);
         %end;
      %else %if %eval(&pctcomplete<100)
            and &idxnumcomplete=YES %then %do;
         if (i=1 or (int(rand('uniform')*100)+1) < &pctcomplete)
```

```
            then nums{i}=int(rand('uniform')*&maxnum);
         else nums{i}=.;
         %end;
      %else %if %eval(&pctcomplete<100)%then %do;
         if (int(rand('uniform')*100)+1) < &pctcomplete
            then nums{i}=int(rand('uniform')*&maxnum);
         else nums{i}=.;
         %end;
         end;
      %end;
      do obs2=1 to ifn(obs<=&j,%sysfunc(floor(%sysevalf(&obs/&obsuni))),
         %sysevalf(%sysfunc(floor(%sysevalf(&obs/&obsuni)))+1));
      %if &randomize=YES %then %do;
         rando=rand('uniform');
         %end;
      %if &makedataset^=YES and &makeflatfile=YES and &randomize^=YES %then %do;
         %end;
      %else %do;
         output;
         %end;
      %if &makeflatfile=YES and &randomize^=YES %then %do;
         put &putstatement;
         %end;
         end;
      end;
run;
* obtain data set file size;
%if &makedataset^=YES and &makeflatfile=YES and &randomize^=YES %then %do;
   %end;
%else %do;
proc sql noprint;
   select filesize format=15.
   into :dsnfilesize
   from dictionary.tables
   where libname="WORK" and memname="%upcase(&dsn)";
   quit;
   %let dsnfilesize=%sysevalf(&dsnfilesize/1048576); * in MB;
   %end;
* optionally randomizes observation order;
%if &randomize=YES %then %do;
proc sort data=&dsn out=&dsn (drop=rando);
   by rando;
run;
   %end;
%if &makeflatfile=YES and &randomize=YES %then %do;
data _null_;
   set &dsn;
   file "&flatfile";
   put &putstatement;
run;
   %end;
* obtain flat file size;
%if &makeflatfile=YES %then %do;
   ***** file size fails on mainframe;
   %if &syssite=70021509 %then %let flatfilesize=0;
   %else %do;
      %let rc=%sysfunc(filename(fil,&flatfile));
      %let fid=%sysfunc(fopen(&fil));
      %let flatfilesize=%sysfunc(finfo(&fid,File Size (bytes)));
      %let rc=%sysfunc(fclose(&fid));
      %end;
   %let flatfilesize=%sysevalf(&flatfilesize/1048576); * in MB;
   %end;
```

```
%let mockdataRC=&syscc;
%mend;
```

APPENDIX B. TESTMOCK MACRO

```
%macro testmockinmem(dsnmetrics= /* metrics in LIB.DSN or DSN format */,
   obsstart= /* smallest number of obs */,
   obsend= /* largest number of obs */,
   obsiter= /* number of obs to increment */,
   charvarstart= /* smallest number of char vars */,
   charvarend= /* largest number of char vars */,
   charvariter= /* number of char vars to increment */,
   charlenstart= /* smallest length of char var */,
   charlenend= /* largest length of char var */,
   charleniter= /* length of char var to increment */,
   numvarstart= /* smallest number of numeric vars */,
   numvarend= /* largest number of numeric vars */,
   numvariter= /* number of numeric vars to increment */,
   numlenstart= /* smallest length of numeric var */,
   numlenend= /* largest length of numeric var */,
   numleniter= /* length of numeric var to increment */,
   pctcompstart= /* pct complete low number */,
   pctcompend= /* pct complete high number */,
   pctcompiter= /* pct complete to increment */,
   flatfile= /* primary flat file created */,
   flatfilespace= /* number of spaces between positional vars */,
   metricsreport= /* file name of HTML report with metrics */);
data &dsnmetrics;
   if 0 then set metrics_base;
run;
%local a b c d e f i procedure;
* get memory and convert from bytes to MB;
%let memsize=%sysfunc(int(%sysevalf(%sysfunc(getoption(xmrlmem))/1048576)));
%let sortsize=%sysfunc(getoption(sortsize));
%if %sysfunc(notdigit(&sortsize))>0 %then %let sortsize=;
%else %let sortsize=%sysfunc(int(%sysevalf(&sortsize/1048576)));
%let cpucount=%sysevalf(%sysfunc(getoption(cpucount)));
%let i=0;
%do a=&obsstart %to &obsend %by &obsiter;
   %do b=&charvarstart %to &charvarend %by &charvariter;
      %do c=&charlenstart %to &charlenend %by &charleniter;
         %do d=&numvarstart %to &numvarend %by &numvariter;
            %do e=&numlenstart %to &numlenend %by &numleniter;
               %do f=&pctcompstart %to &pctcompend %by &pctcompiter;
   %let i=%sysevalf(&i+1);
   %mockdata(obs=&a, obsuni=&a, charvar=&b, charlen=&c,
      numvar=&d, numlen=&e, pctcomplete=&f,
      makedataset=YES, dsn=test, makeflatfile=YES,
      flatfile=&flatfile, flatfilespace=&flatfilespace,
      idxcharcomplete=NO, idxnumcomplete=NO, randomize=NO);
   %let err&i=&mockdataRC;
   %let obs&i=&a;
   %let obsuni&i=&a;
   %let charvar&i=&b;
   %let charlen&i=&c;
   %let numvar&i=&d;
   %let numlen&i=&e;
   %let pctcomp&i=&f;
   %let dsnsize&i=&dsnfilesize;
   %let flatsize&i=&flatfilesize;
               %end;
            %end;
         %end;
      %end;
   %end;
```

```
data &dsnmetrics (drop=i);
   if 0 then set metrics_base;
   machine="&machine";
   err=&mockdataRC;
   cpucount=&cpucount;
   memsize=&memsize;
   sortsize=&sortsize;
   do i=1 to &i;
      obs=symget('obs' || strip(put(i,$8.)));
      obsuni=symget('obsuni' || strip(put(i,$8.)));
      charvar=symget('charvar' || strip(put(i,$8.)));
      charlen=symget('charlen' || strip(put(i,$8.)));
      numvar=symget('numvar' || strip(put(i,$8.)));
      numlen=symget('numlen' || strip(put(i,$8.)));
      pctcomp=symget('pctcomp' || strip(put(i,$8.)));
      dsnfilesize=symget('dsnsize' || strip(put(i,$8.)));
      flatfilesize=symget('flatsize' || strip(put(i,$8.)));
      output;
      end;
run;
%mend;
```

APPENDIX C. TESTMOCKINMEM MACRO

```
%macro testmockinmem(dsnmetrics= /* metrics in LIB.DSN or DSN format */,
   obsstart= /* smallest number of obs */,
   obsend= /* largest number of obs */,
   obsiter= /* number of obs to increment */,
   charvarstart= /* smallest number of char vars */,
   charvarend= /* largest number of char vars */,
   charvariter= /* number of char vars to increment */,
   charlenstart= /* smallest length of char var */,
   charlenend= /* largest length of char var */,
   charleniter= /* length of char var to increment */,
   numvarstart= /* smallest number of numeric vars */,
   numvarend= /* largest number of numeric vars */,
   numvariter= /* number of numeric vars to increment */,
   numlenstart= /* smallest length of numeric var */,
   numlenend= /* largest length of numeric var */,
   numleniter= /* length of numeric var to increment */,
   pctcompstart= /* pct complete low number */,
   pctcompend= /* pct complete high number */,
   pctcompiter= /* pct complete to increment */,
   flatfile= /* primary flat file created */,
   flatfilespace= /* number of spaces between positional vars */,
   metricsreport= /* file name of HTML report with metrics */);
%local a b c d e f i procedure;
* get memory and convert from bytes to MB;
%let memsize=%sysfunc(int(%sysevalf(%sysfunc(getoption(xmrlmem))/1048576)));
%let sortsize=%sysfunc(getoption(sortsize));
%if %sysfunc(notdigit(&sortsize))>0 %then %let sortsize=;
%else %let sortsize=%sysfunc(int(%sysevalf(&sortsize/1048576)));
%let cpucount=%sysevalf(%sysfunc(getoption(cpucount)));
%let i=0;
%do a=&obsstart %to &obsend %by &obsiter;
   %do b=&charvarstart %to &charvarend %by &charvariter;
      %do c=&charlenstart %to &charlenend %by &charleniter;
         %do d=&numvarstart %to &numvarend %by &numvariter;
            %do e=&numlenstart %to &numlenend %by &numleniter;
               %do f=&pctcompstart %to &pctcompend %by &pctcompiter;
   %let i=%sysevalf(&i+1);
   %mockdata(obs=&a, obsuni=&a, charvar=&b, charlen=&c,
      numvar=&d, numlen=&e, pctcomplete=&f,
      makedataset=YES, dsn=test, makeflatfile=YES,
      flatfile=&flatfile, flatfilespace=&flatfilespace,
      idxcharcomplete=NO, idxnumcomplete=NO, randomize=NO);
   %let err&i=&mockdataRC;
   %let obs&i=&a;
   %let obsuni&i=&a;
   %let charvar&i=&b;
   %let charlen&i=&c;
   %let numvar&i=&d;
   %let numlen&i=&e;
   %let pctcomp&i=&f;
   %let dsnsize&i=&dsnfilesize;
   %let flatsize&i=&flatfilesize;
               %end;
            %end;
         %end;
      %end;
   %end;
data &dsnmetrics (drop=i);
   if 0 then set metrics_base;
   machine="&machine";
```

```
   err=&mockdataRC;
   cpucount=&cpucount;
   memsize=&memsize;
   sortsize=&sortsize;
   do i=1 to &i;
      obs=input(symget('obs' || strip(put(i,8.))),8.);
      obsuni=input(symget('obsuni' || strip(put(i,8.))),8.);
      charvar=input(symget('charvar' || strip(put(i,8.))),8.);
      charlen=input(symget('charlen' || strip(put(i,8.))),8.);
      numvar=input(symget('numvar' || strip(put(i,8.))),8.);
      numlen=input(symget('numlen' || strip(put(i,8.))),8.);
      pctcomp=input(symget('pctcomp' || strip(put(i,8.))),8.);
      dsnfilesize=input(symget('dsnsize' || strip(put(i,8.))),8.);
      flatfilesize=input(symget('flatsize' || strip(put(i,8.))),8.)/1048576;;
      output;
      end;
run;
%mend;


%let dtgstart=%sysfunc(datetime());
proc printto &printtolog new;
run;
%testmockinmem(dsnmetrics=ushris.size_metricsinmem,
   obsstart=1, obsend=1000, obsiter=1,
   charvarstart=1, charvarend=50, charvariter=1,
   charlenstart=1, charlenend=50, charleniter=1,
   numvarstart=0, numvarend=0, numvariter=1,
   numlenstart=0, numlenend=0, numleniter=1,
   pctcompstart=100, pctcompend=100, pctcompiter=10,
   flatfile=D:\sas\USHRIS_mainframe\flatfile.txt, flatfilespace=0,
   metricsreport=&metricsreport);
proc printto;
run;
%put TESTMOCKINMEM completed in %sysevalf(%sysfunc(datetime())-&dtgstart);
```