PharmaSUG 2021 – Paper RW-198

Better to Be Mocked Than Half-Cocked: Data Mocking Methods to Support Functional and Performance Testing of SAS® Software

Troy Martin Hughes

ABSTRACT

Data mocking refers to the practice of manufacturing data that can be used in software functional and performance testing, including both load testing and stress testing. Mocked data are not production or "real" data, in that they do not abstract some real-world construct, but are considered to be sufficiently similar (to production data) to demonstrate how software would function and perform in a production environment. Data mocking is commonly employed during software development and testing phases and is especially useful where production data may be sensitive or where it may be infeasible to import production data into a non-production environment (such as where production data contain sensitive PII or PHI, company secrets, or federal government super-secrets). This text introduces the MOCKDATA SAS® macro, which creates mock data sets and/or text files for which SAS practitioners can vary (through parameterization) the number of observations, number of unique observations, randomization of observation order, number of character variables, length of character variables, number of numeric variables, highest numeric value, percentage of variables that have data, and whether character and/or numeric index variables (which cannot be missing) exist. An example implements MOCKDATA to compare the input/output (I/O) processing performance of SAS data sets and flat files, demonstrating the clear performance advantages of processing SAS data sets in lieu of text files.

INTRODUCTION

Data mocking benefits environments and processes for which production data is costly, difficult, or infeasible to use during software development and testing. For example, if a development team needs to demonstrate that newly released software will exhibit acceptable performance in the future when the quantity of data is expected to have sextupled, the development team might mock data to create a sufficiently large data set to demonstrate the acceptable scaling of software during load testing. In other cases, sufficient production data may exist but may not be available in non-production environments (e.g., development, test regions). For example, a production environment might contain sensitive personally identifiable information (PII), protected health information (PHI), or federally classified data that cannot be transferred to non-production environments—and in these cases, data mocking can create representative data for development and testing phases of the software development life cycle (SDLC).

Data mocking is also useful where functional of performance testing requires that large data sets be processed across disparate environments. For example, researchers across the ten University of California (UC) campuses might want to separately test the performance of a single SAS program in their respective environments. Emailing the 46 KB SAS program file among researchers would not pose a logistic challenge; however, distribution of a 100 GB SAS data set could be difficult. Data mocking—and the MOCKDATA macro in particular—can facilitate this pursuit by enabling each researcher to create an equivalent data set by supplying MOCKDATA with identical parameters. MOCKDATA similarly enables SAS practitioners to include code that produces mock data sets inside white papers and other publications rather than being forced to rely only on SASHELP.Cars and other hackneyed data sets that ship with the SAS application.

This text introduces the MOCKDATA macro, which creates data sets and/or positional flat files dynamically, allowing multiple dimensions of files to be altered. The LOADTEST macro (see Appendix C) iteratively calls MOCKDATA to support load testing and can be used to incrementally alter observation count, variable type and count, variable length, and other dimensions. LOADTEST relies on the PINCHLOG macro to capture FULLSTIMER performance metrics so the impact of MOCKDATA parameters on operation runtime can be individually and collectively evaluated. One sample LOADTEST invocation is included, which demonstrates the clear performance advantages of both reading and writing SAS data sets as compared to reading and writing positional flat files. Data mocking importantly mitigates the risk of walking into your production environment half-cocked, and for this reason is a common best practice.

MOCKDATA MACRO

The MOCKDATA macro definition follows:

```
%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=0 /* spaces between each column */,
   flatfilemiss= /* print missing numbers as space, not PERIOD */,
   randomize=YES /* YES or NO to sort the data set randomly */);
```

MOCKDATA defines the following parameters:

- **OBS** – The number of observations in the resultant data set and/or flat file.
- **OBSUNI** – The approximate number of unique observations in the resultant data set and/or flat file. OBSUNI must be equal to or less than OBS. When OBSUNI equals OBS, all observations may be unique, but this is not guaranteed because the RAND function could produce observations that are identical. As the complexity of the data set being generated increases (e.g., greater number of observations, greater number of character or numeric variables, greater length of character or numeric variables), the likelihood of unintended identical observations will decrease.
- **CHARVAR** – The number of character variables created.
- **CHARLEN** – The length of character variables created.
- **NUMVAR** – The number of numeric variables created.
- **NUMLEN** – The length of the numeric variables created, which is used to calculate the range of possible integers that can be assigned to a numeric variable by the formula MAXNUM = 32 x $256^{(NUMLEN - 2)}$. For example, when NUMLEN is 3, all numeric variables can range from between 0 and 8,192, and when NUMLEN is 4, all numeric variables can range from between 0 to 2,097,152.
- **PCTCOMPLETE** – The average percentage (from 1 to 100) of variables that have data. For example, if PCTCOMPLETE is 90, then each character and numeric variable will individually have a 90 percent chance of having a value.
- **IDXCHARCOMPLETE** – YES to require that the first character variable (i.e., Char1) has no missing values, irrespective of the PCTCOMPLETE argument.
- **IDXNUMCOMPLETE** – YES to require that the first numeric variable (i.e., NUM1) has no missing values, irrespective of the PCTCOMPLETE argument.
- **MAKEDATASET** – YES to create an output data set.
- **DSN** – data set name when MAKEDATASET equals YES.
- **MAKEFLATFILE** – YES to create an output positional flat file.
- **FLATFILE** – flat file name when MAKEFLATFILE equals YES.
- **FLATFILESPACE** – when MAKEFLATFILE equals YES, this parameter specifies the number of spaces (0 or greater) that will separate each character or numeric variable.

- **FLATFILEMISS** – the SPACE value specifies that missing numeric values (that occur when the MAKEFLATFILE argument is YES and PCTCOMPLETE argument is less than 100) are denoted by spaces rather than the default period (by setting the MISSING SAS system option to ‘’).
- **RANDOMIZE** – YES to sort the resultant data set and/or flat file by the first character variable (i.e., Char1) or, if the CHARVAR argument is 0, by the first numeric variable (i.e, NUM1). When the RANDOMIZE argument is YES, the IDXCHARCOMPLETE argument should be YES to ensure that the sorted variable is not missing (if character variables are being created), and the IDXNUMCOMPLETE argument should be YES to ensure that the sorted variable is not missing (if no character variables are being created and only numeric variables are being created).

Based on whether MOCKDATA is creating a data set, is creating a positional flat file, and is being randomized, various statements will be executed dynamically within the macro. Table 1 demonstrates statements that execute dynamically (within MOCKDATA) based on the values of the MAKEDATASET, MAKEFLATFILE, and RANDOMIZE parameters. For example, if all three arguments equal YES then “DATA &dsn;” is executed but not “DATA _null_;”.

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;	X		X		X	
3. DATA Step						
FILE “&flatfile”;	X				X	
PUT &putstatement;	X				X	

Table 1. Statements Executed Dynamically by the MOCKDATA Macro

The following sample invocation of MOCKDATA creates a SAS data set (Mydata) that has 10,000 observations comprising five 10-character variables (and no numeric variables):

```
%mockdata(obs=10000,
   obsuni= 1000,
   charvar=5,
   charlen=10,
   numvar=0,
   pctcomplete=100,
   idxcharcomplete=YES,
```

```
makedataset=YES,
dsn=mydata,
makeflatfile=NO,
randomize=YES);
```

Only 1,000 unique observations are created, so the DATA step dynamically builds one observation then outputs it ten times. Rinse and repeat 1,000 times to create a data set with 10,000 observations. The data set is subsequently ordered randomly by the Rando variable (created with the RAND function). The data are demonstrated in Table 2.

Char1	Char2	Char3	Char4	Char5
BIDEAJJBBE	DBHIFGDFHF	EHCIAFDIHB	FJJHAEGGFE	GFCHGIIGDH
FBGACIGEFE	JIBBCEBJFA	EFDBDGADEA	BFHHFJBABF	GCICEDEIFF
EAHCIIDBFC	DFGJGDGBIA	EAFFBIABEA	FBHJCCIJCF	DBJFJGACAC
EHEBJCJJHF	BCJEJFJDBJ	ADEAHGHGAD	BFJBBACBHD	IJDABHDGAF
BIDEAJJBBE	DBHIFGDFHF	EHCIAFDIHB	FJJHAEGGFE	GFCHGIIGDH

Table 2. First Five Observations of Mydata Data Set

Note that the first and fifth observations are identical, representing that the observation was created ten times (because 1,000 unique observations were specified within the total 10,000 observations). The FREQ procedure can be used to demonstrate this distribution, with each Char1 value occurring ten times within the Mydata data set:

```
proc freq data=mydata order=freq;
   tables char1;
run;
```

As the number of observations increases or the length of variables decreases, the likelihood—however miniscule—of "collisions" increases, in which two variables intended to be unique will be duplicated by random chance. Randomization of observations is only required for testing certain operations, such as performance testing the speed of sorting data (e.g., SORT procedure) or assessing the frequency of values (e.g., FREQ procedure). Thus, in many cases in which only the number of observations (or file size) must be scaled to demonstrate load testing or stress testing, one observation can be created and duplicated for each observation that is created.

PERFORMANCE TESTING AND THE PINCHLOG MACRO

The MOCKDATA macro facilitates performance testing by creating data sets (and/or flat files) that can be used to measure runtime performance and resource utilization (e.g., memory, I/O, CPU cycles). In these examples, the number of observations is varied to demonstrate the scalability of various operations as the quantity of data increases. The PINCHLOG macro (demonstrated in Appendix A) is used to collect FULLSTIMER performance metrics automatically from log files and is introduced in the author's text *Pinching Off Your SAS® Log: Adapting from Loquacious to Laconic Logs To Facilitate Near-Real Time Log Parsing, Performance Analysis, and Dynamic, Data-Driven Design and Optimization* (Hughes, 2017).

The following code snippet from the LOADTEST macro (see Appendix B) demonstrates invocation of the MOCKDATA macro to create a sample data set, a DATA step that reads the sample data set, and invocation of the PINCHLOG macro to capture the performance metrics from that DATA step:

```
* create the data set and/or flat file;
%mockdata(obs=&a, obsuni=1, charvar=&b, charlen=&c,
   numvar=0, numlen=0, pctcomplete=&d, idxcharcomplete=no
   makedataset=yes, dsn=&dsn, makeflatfile=yes,
   flatfile=&flatfile, flatfilespace=&flatfilespace,
   flatfilemiss=SPACE, randomize=NO);

* 1. Input Data Set - Output _NULL_;
```

```
%let syscc=0;
%let procedure=Input Data Set - Output _NULL_;
proc printto log="&logtemp" new;
run;
data _null_;
   set &dsn;
run;
proc printto log="&printtolog";
run;
%let newsyscc=&syscc;
%pinchlog(logfile=&logtemp, dsnmetrics=&dsnmetrics,
   othervars=(&pinchother));
```

PINCHLOG is described fully in the referenced text, and creates a metrics data set that includes runtime, CPU time, memory, OS memory, and other FULLSTIMER performance metrics. When invoked repetitively within incremental load testing, as occurs in the LOADTEST macro, PINCHLOG not only creates but also incrementally builds the metrics data, which can be analyzed thereafter to evaluate the impact of observation count (and other attributes) on software performance.

LOADTEST MACRO

LOADTEST (see Appendix C) evaluates and compares the relative performance of basic data I/O operations. Two input operations are contrasted. Data are ingested from SAS data sets and from positional flat files, with the former operation completing in less time than the latter. Four output operations are contrasted. These include not creating a set (using the _NULL_ option), creating a data set, creating a flat file, and creating both a data set and flat file. The _NULL_ option is the fastest operation because no data set is created. Creating a SAS data set is second fastest, followed by creating a positional flat file. The slowest output operation creates both a flat file and data set. Thus, as demonstrated in both Figure 1 and Table 3, “Input Data Set – Ouptut _NULL_” executes the fastest and “Input Flat File – Output Data Set and Flat File” executes the slowest.

Before the LOADTEST macro can be executed, the &PINCHOTHER macro variable must be created, which is utilized by the PINCHLOG macro calls within LOADTEST. With &PINCHOTHER created, LOADTEST can be invoked to create mock data sets that increment from between 100,000 and 10 million observations:

```
%let loc=D:\sas\mockdata\;       * USER MUST CHANGE LOCATION;

%include "&loc.mockdata.sas";
%include "&loc.pinchlog.sas";
libname mockery "&loc";

%macro delifexist;
%if %symexist(flatfilesize) %then %symdel flatfilesize;
%if %symexist(dsnfilesize) %then %symdel dsnfilesize;
%if %symexist(memory) %then %symdel memory;
%if %symexist(osmemory) %then %symdel osmemory;
%mend;

%delifexist;

%global pinchother;
%let pinchother=%str(var=procedure, val=&procedure, len=$50, form=$50.,
   lab=Test Procedure /
var=merr, val=&mockdataRC, len=8, form=8.,
   lab=MOCKDATA Error Code /
var=err, val=&newsyscc, len=8, form=8., lab=Pinch Error Code /
var=makedataset, val=&makedataset, len=$3, form=$3.,
   lab=Make Data Set /
var=makeflatfile, val=&makeflatfile, len=$3, form=$3.,
   lab=Make Flat File /
var=obs, val=&a, len=8, form=comma15., lab=Obs /
```

```
var=obsuni, val=1, len=8, form=comma15., lab=Unique Obs /
var=charvar, val=&b, len=8, form=8., lab=Character Variables /
var=charlen, val=&c, len=8, form=8., lab=Character Length /
var=pctcomp, val=&d, len=8, form=8., lab=Percent Complete /
var=dsnfilesize, val=&dsnfilesize, len=8, form=comma15.5,
   lab=DSN Size in MB /
var=flatfilesize, val=&flatfilesize, len=8, form=comma15.5,
   lab=Flat File Size in MB /
var=flatfilespace, val=&flatfilespace, len=3, form=8.,
   lab=Flat File Space /
var=cpucount, val=&cpucount, len=8, form=8., lab=CPUCOUNT /
var=memsize, val=&memsize, len=8, form=comma8.,
   lab=MEMSIZE in MB /
var=sortsize, val=&sortsize, len=8, form=comma8.,
   lab=SORTSIZE in MB);

%let printtolog=&loc.log.txt;
%let logtemp=&loc.log_temp.txt;

options fullstimer cpucount=actual; *maximizes # of CPUs;
proc printto log="&printtolog" new;
run;
%loadtest(dsn=mockery.mydata,
   dsnmetrics=mockery.metrics,
   printtolog=&printtolog, logtemp=&logtemp,
   obsstart=100000, obsend=10000000, obsiter=100000,
   charvarstart=100, charvarend=100, charvariter=100,
   charlenstart=10, charlenend=10, charleniter=10,
   pctcompstart=100, pctcompend=100, pctcompiter=10,
   flatfile=&loc.myflat.txt, flatfilebak=&loc.myflatback.txt, flatfilespace=0);
proc printto;
run;
```

LOADTEST iterates 100 times and creates eight observations in the metrics data set each observation, thus producing 800 total observations.

RESULTS

Figure 1 demonstrates a subset of the performance testing, showing runtime in seconds from 100,000 observations to one million observations. Note the advantage of reading and writing from SAS data sets (as opposed to flat files), with “Input Data Set – Output _NULL_” and “Input Data Set – Output Data Set” completing the fastest and second fastest, respectively, for all observations counts.

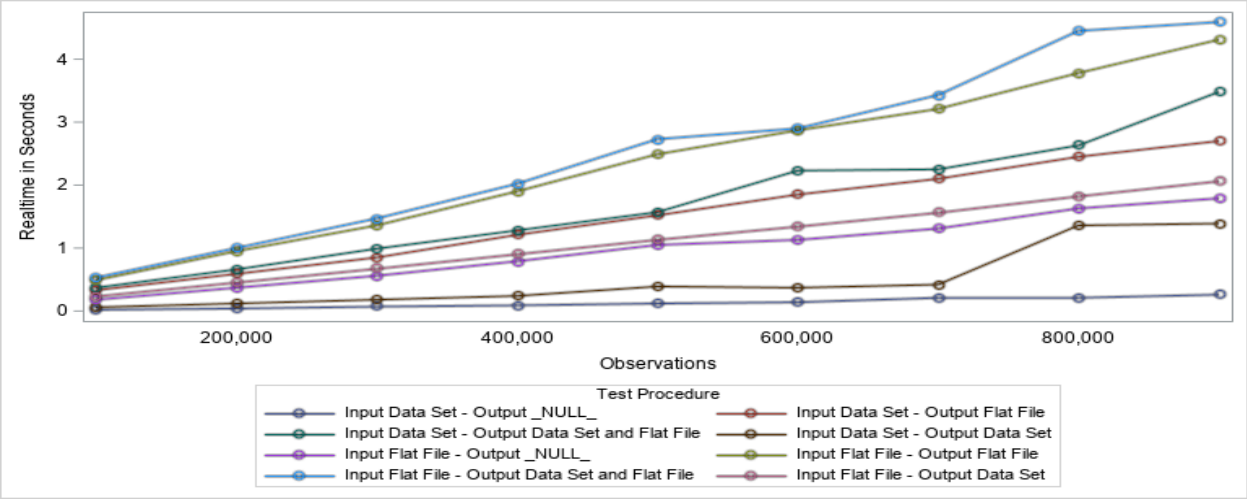

Figure 1. Performance Advantages of Using SAS Data Sets over Flat Files

For example, at one million observations, “Input Data Set – Output _NULL_” completed more than five times faster than “Input Flat File – Output _NULL_.” Similarly, ingesting and writing a SAS data set completes nearly five times faster than ingesting and writing an equivalent flat file. These performance advantages are demonstrated in Table 3.

Method	Runtime (in Seconds)
Input Data Set - Output _NULL_	0.340
Input Data Set - Output Data Set	0.950
Input Flat File - Output _NULL_	1.930
Input Data Set - Output Flat File	3.030
Input Flat File - Output Data Set	3.140
Input Data Set - Output Data Set and Flat File	4.140
Input Flat File - Output Flat File	4.530
Input Flat File – Output Data Set and Flat File	4.820

Table 3. Decreased Runtime of Data Sets over Flat Files at One Million Observations

CONCLUSION

The MOCKDATA macro is introduced, which creates mock data sets that can be used to facilitate functional and performance testing of SAS software. This text demonstrates load testing that shows the clear advantage of maintaining data within SAS data sets, where possible, in lieu of positional flat files—due to the I/O overhead that flat file reading and writing causes. The PINCHLOG macro is also demonstrated, which facilitates collection of FULLSTIMER performance metrics and aggregation into a metrics data set.

REFERENCES

Hughes, T. M. (2017). Pinching Off Your SAS® Log: Adapting from Loquacious to Laconic Logs To Facilitate Near-Real Time Log Parsing, Performance Analysis, and Dynamic, Data-Driven Design and Optimization. *Western Users of SAS Software (WUSS).* Long Beach, CA.

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Name: Troy Martin Hughes
E-mail: troymartinhughes@gmail.com

APPENDIX A. MOCKDATA MACRO

```
%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 LIB.DSN or DSN format */,
   makeflatfile=NO /* YES to create a positional flat file */,
   flatfile= /* location, file name, and extension */,
   flatfilespace=0 /* spaces between each column */,
   flatfilemiss= /* 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;
%let bytestart=65;
%let inputstatement=;
%let putstatement=;
%let dsnfilesize=;
%let flatfilesize=;
%local i j space maxnum maxnumwidth numvar dsntemp rc fid libname memname;
%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)+&bytestart));
            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;
   %let dsn=%upcase(&dsn);
   %if %length(%scan(&dsn,2,%str(.)))>0 %then %do;
      %let libname=%scan(&dsn,1,%str(.));
      %let memname=%scan(&dsn,2,%str(.));
      %end;
   %else %do;
      %let libname=WORK;
      %let memname=&dsn;
      %end;
   proc sql noprint;
      select filesize format=15.
      into :dsnfilesize
      from dictionary.tables
      where libname="&libname" and memname="&memname";
      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;
   %let rc=%sysfunc(filename(fil,&flatfile));
   %let fid=%sysfunc(fopen(&fil));
   %let flatfilesize=%sysfunc(finfo(&fid,File Size (bytes)));
   %let rc=%sysfunc(fclose(&fid));
   %let flatfilesize=%sysevalf(&flatfilesize/1048576); * in MB;
   %end;
%let mockdataRC=&syscc;
%mend;
```

APPENDIX B. PINCHLOG MACRO

```
%macro pinchlog(logfile= /* path, file name, and extension */,
    dsnmetrics= /* optional metrics data set in LIB.DSN or DSN format */,
    othervars= /* optional tokenized list of user-defined variables */);
* all times converted from HH:MM:SS.ss to SSSS.xx format;
%let syscc=0;
%global pinchlogRC pinchlogchildRC realtime usercputime systemcputime
    memory osmemory stepcount switchcount pagefaults pagereclaims
    volcontextswitches involcontextswitches blockinops blockoutops;
%let pinchlogRC=99;
%let pinchlogchildRC=0;
%let realtime=.;
%let usercputime=.;
%let systemcputime=.;
%let memory=.;
%let osmemory=.;
%let stepcount=.;
%let switchcount=.;
%let pagefaults=.;
%let pagereclaims=.;
%let volcontextswitches=.;
%let involcontextswitches=.;
%let blockinops=.;
%let blockoutops=.;
data _null_;
    length tab $500;
    infile "&logfile" truncover;
    input tab $500.;
    if strip(tab)=:'WARNING' or strip(tab)=:'ERROR' then do;
           call symput('pinchlogchildRC','4');
           return;
           end;
    if _n_>=7 then do; * skips the metrics produced by PRINTTO itself;
           if lowcase(substr(tab,1,9))='real time' then do;
                  if count(scan(substr(tab,10),1,' '),':')=0 then
                         call symput('realtime',scan(substr(tab,10),1,' '));
                  else if count(scan(substr(tab,10),1,' '),':')=1 then
                         call symput('realtime',
                         put(((input(strip(scan(substr(tab,10),
                         1,':')),8.0) * 60) +
                         input(strip(scan(substr(tab,10),2,':')),8.2)),8.2));
                  else if count(scan(substr(tab,10),1,' '),':')=2 then
                         call symput('realtime',
                         put(((input(strip(scan(substr(tab,10),
                         1,':')),8.0) * 3600) +
                         (input(strip(scan(substr(tab,10),2,':')),8.0) * 60) +
                         input(strip(scan(substr(tab,10),3,':')),8.2)),8.2));
                  end;
           else if lowcase(substr(tab,1,13))='user cpu time' then do;
                  if count(scan(substr(tab,14),1,' '),':')=0 then
                         call symput('usercputime',scan(substr(tab,14),1,' '));
                  else if count(scan(substr(tab,14),1,' '),':')=1 then
                         call symput('usercputime',
                         put(((input(strip(scan(substr(tab,14),
                         1,':')),8.0) * 60) +
                         input(strip(scan(substr(tab,14),2,':')),8.2)),8.2));
                  else if count(scan(substr(tab,14),1,' '),':')=2 then
                         call symput('usercputime',
                         put(((input(strip(scan(substr(tab,14),
                         1,':')),8.0) * 3600) +
                         (input(strip(scan(substr(tab,14),2,':')),8.0) * 60) +
                         input(strip(scan(substr(tab,14),3,':')),8.2)),8.2));
```

```
                end;
        else if lowcase(substr(tab,1,15))='system cpu time' then do;
                if count(scan(substr(tab,16),1,' '),':')=0 then
                        call symput('systemcputime',scan(substr(tab,16),1,' '));
                else if count(scan(substr(tab,16),1,' '),':')=1 then
                        call symput('systemcputime',
                        put(((input(strip(scan(substr(tab,16),
                        1,':')),8.0) * 60) +
                        input(strip(scan(substr(tab,16),2,':')),8.2)),8.2));
                else if count(scan(substr(tab,16),1,' '),':')=2 then
                        call symput('systemcputime',
                        put(((input(strip(scan(substr(tab,16),
                        1,':')),8.0) * 3600) +
                        (input(strip(scan(substr(tab,16),2,':')),8.0) * 60) +
                        input(strip(scan(substr(tab,16),3,':')),8.2)),8.2));
                end;
        * convert KB to MB;
        else if lowcase(substr(tab,1,6))='memory' then
                call symput('memory',
                put(input(scan(substr(tab,7),1,' k'),8.3)/1024,8.3));
        else if lowcase(substr(tab,1,9))='os memory' then
                call symput('osmemory',
                put(input(scan(substr(tab,10),1,' k'),8.3)/1024,8.3));
        else if lowcase(substr(tab,1,10))='step count' then do;
                call symput('stepcount',scan(substr(tab,11),1,' '));
                call symput('switchcount',scan(substr(tab,11),4,' '));
                end;
        else if lowcase(substr(tab,1,11))='page faults' then
                call symput('pagefaults',scan(substr(tab,12),1,' '));
        else if lowcase(substr(tab,1,13))='page reclaims' then
                call symput('pagereclaims',scan(substr(tab,14),1,' '));
        else if lowcase(substr(tab,1,26))='voluntary context switches' then
                call symput('volcontextswitches',scan(substr(tab,27), 1,' '));
        else if lowcase(substr(tab,1,28))='involuntary context switches' then
                call symput('involcontextswitches',scan(substr(tab,29),1,' '));
        else if lowcase(substr(tab,1,24))='block input operations' then
                call symput('blockinops',scan(substr(tab,25),1,' '));
        else if lowcase(substr(tab,1,25))='block output operations' then
                call symput('blockoutops',scan(substr(tab,26),1,' '));
        end;
run;
* optionally initialize user-defined variables;
* at least VAR, VAL, and FORM sub-parameters are required for each variable;
%if %length(othervars)>0 %then %do;
    %local otherval otherlen otherform otherlab var val len form lab i j;
    %let otherval=;
    %let otherlen=;
    %let otherform=;
    %let otherlab=;
    %let othervars=%sysfunc(compress(%bquote(&othervars),%nrstr(%))%nrstr(%(),));
    %let i=1;
    %do %while(%length(%scan(%bquote(&othervars),&i,/))>1);
        %let var=;
        %let val=;
        %let len=;
        %let form=;
        %let lab=;
        %let varstring=%scan(%bquote(&othervars),&i,/);
        %let j=1;
        %do %while(%length(%scan(%bquote(&varstring),&j,%str(,)))>1);
            %let valstring=%scan(%bquote(&varstring),&j,%str(,));
            %if %lowcase(%scan(&valstring,1,=))=var
                %then %let var=%scan(&valstring,2,=);
```

```
                %if %lowcase(%scan(&valstring,1,=))=val
                       %then %let val=%scan(&valstring,2,=);
                %if %lowcase(%scan(&valstring,1,=))=len
                       %then %let len=%scan(&valstring,2,=);
                %if %lowcase(%scan(&valstring,1,=))=form
                       %then %let form=%scan(&valstring,2,=);
                %if %lowcase(%scan(&valstring,1,=))=lab
                       %then %let lab=%scan(&valstring,2,=);
                %let j=%eval(&j+1);
                %end;
          %if %length(&var)>0 and %length(&len)>0 and %length(&val)>0 %then %do;
                %let otherlen=&otherlen &var &len;
                %if %substr(&len,1,1)=$ %then
                       %let otherval=&otherval &var="&val"%str(;);
                %else %let otherval=&otherval &var=&val%str(;);
                %if %length(&form)>0 %then %let otherform=&otherform &var &form;
                %if %length(&lab)>0 %then %let otherlab=&otherlab &var="&lab";
                %end;
          %let i=%eval(&i+1);
          %end;
    %end;
* optionally create/modify metrics data set;
%if %length(&dsnmetrics)>0 %then %do;
    %if %sysfunc(exist(&dsnmetrics))=0 %then %do;
          data &dsnmetrics;
                length realtime 8 usercputime 8 systemcputime 8 memory 8 osmemory
8
                       stepcount 8 switchcount 8 pagefaults 8 pagereclaims 8
                       volcontextswitches 8 involcontextswitches 8 blockinops 8;
                format realtime 8.3 usercputime 8.3 systemcputime 8.3 memory 8.3
                       osmemory 8.3 stepcount 8. switchcount 8. pagefaults 8.
                       pagereclaims 8. volcontextswitches 8. involcontextswitches
8.
                       blockinops 8.;
                label realtime='Real Time' usercputime='User CPU Time'
                       systemcputime='System CPU Time' memory='Memory in MB'
                       osmemory='OS Memory in MB' stepcount='Step Count'
                       switchcount='Switch Count' pagefaults='Page Faults'
                       pagereclaims='Page Reclaims' volcontextswitches=
                       'Voluntary Context Switches' involcontextswitches=
                       'Involuntary Context Switches'
                       blockinops='Block Input Operations'
                       blockoutops='Block Output Operations';
                %if %length(&otherval)>0 %then %do;
                       length &otherlen;;
                       %if %length(&otherform)>0 %then %do;
                              format &otherform;;
                              %end;
                       %if %length(&otherlab)>0 %then %do;
                              label &otherlab;;
                              %end;
                       %end;
                if not missing(realtime);
          run;
          %end;
    data pinchtemp;
          if 0 then set &dsnmetrics;
          realtime=&realtime;
          usercputime=&usercputime;
          systemcputime=&systemcputime;
          memory=&memory;
          osmemory=&osmemory;
          stepcount=&stepcount;
```

```
            switchcount=&switchcount;
            pagefaults=&pagefaults;
            pagereclaims=&pagereclaims;
            volcontextswitches=&volcontextswitches;
            involcontextswitches=&involcontextswitches;
            blockinops=&blockinops;
            %if %length(&otherval)>0 %then %do;
                  &otherval;
                  %end;
            output;
      run;
      proc append base=&dsnmetrics data=pinchtemp;
      run;
      %end;
%let pinchlogRC=&syscc;
%mend;
```

APPENDIX C. LOADTEST MACRO

```
%macro loadtest(dsn= /* mock data data set */,
   dsnmetrics= /* metrics in LIB.DSN or DSN format */,
   printtolog= /* main log file */,
   logtemp= /* name of temporary log */,
   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 */,
   pctcompstart= /* pct complete low number */,
   pctcompend= /* pct complete high number */,
   pctcompiter= /* pct complete to increment */,
   flatfile= /* primary flat file created */,
   flatfilebak= /* secondary flat file created */,
   flatfilespace= /* number of spaces between positional vars */);
%local a b c d e i procedure newsyscc;
* 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=&pctcompstart %to &pctcompend %by &pctcompiter;

      * create the data set and/or flat file;
      %mockdata(obs=&a, obsuni=1, charvar=&b, charlen=&c,
         numvar=0, numlen=0, pctcomplete=&d, idxcharcomplete=no
         makedataset=yes, dsn=&dsn, makeflatfile=yes,
         flatfile=&flatfile, flatfilespace=&flatfilespace,
         flatfilemiss=SPACE, randomize=NO);

      * 1. Input Data Set - Output _NULL_;

      %let syscc=0;
      %let procedure=Input Data Set - Output _NULL_;
      proc printto log="&logtemp" new;
      run;
      data _null_;
         set &dsn;
      run;
      proc printto log="&printtolog";
      run;
      %let newsyscc=&syscc;
      %pinchlog(logfile=&logtemp, dsnmetrics=&dsnmetrics,
         othervars=(&pinchother));

      * 2. Input Data Set - Output Flat File;

      %let syscc=0;
      %let procedure=Input Data Set - Output Flat File;
      proc printto log="&logtemp" new;
      run;
      filename fbak "&flatfilebak";
```

```
data _null_;
   set &dsn;
   file fbak;
   put &putstatement;
run;
proc printto log="&printtolog";
run;
%let newsyscc=&syscc;
%pinchlog(logfile=&logtemp, dsnmetrics=&dsnmetrics,
   othervars=(&pinchother));

* 3. Input Data Set - Output Data Set and Flat File;

%let syscc=0;
%let procedure=Input Data Set - Output Data Set and Flat File;
proc printto log="&logtemp" new;
run;
filename fbak "&flatfilebak";
data test2;
   set &dsn;
   file fbak;
   put &putstatement;
run;
proc printto log="&printtolog";
run;
%let newsyscc=&syscc;
%pinchlog(logfile=&logtemp, dsnmetrics=&dsnmetrics,
   othervars=(&pinchother));

* 4. Input Data Set - Output Data Set;

%let syscc=0;
%let procedure=Input Data Set - Output Data Set;
proc printto log="&logtemp" new;
run;
data test2;
   set &dsn;
run;
proc printto log="&printtolog";
run;
%let newsyscc=&syscc;
%pinchlog(logfile=&logtemp, dsnmetrics=&dsnmetrics,
   othervars=(&pinchother));

* 5. Input Flat File - Output _NULL_;

%let syscc=0;
%let procedure=Input Flat File - Output _NULL_;
proc printto log="&logtemp" new;
run;
filename f "&flatfile";
data _null_;
   infile f truncover end=eof;
   input &inputstatement;
run;
proc printto log="&printtolog";
run;
%let newsyscc=&syscc;
%pinchlog(logfile=&logtemp, dsnmetrics=&dsnmetrics,
   othervars=(&pinchother));

* 6. Input Flat File - Output Flat File;
```

```
      %let syscc=0;
      %let procedure=Input Flat File - Output Flat File;
      proc printto log="&logtemp" new;
      run;
      filename f "&flatfile";
      filename fbak "&flatfilebak";
      data _null_;
         infile f truncover end=eof;
         input &inputstatement;
         file fbak;
         put &putstatement;
      run;
      proc printto log="&printtolog";
      run;
      %let newsyscc=&syscc;
      %pinchlog(logfile=&logtemp, dsnmetrics=&dsnmetrics,
         othervars=(&pinchother));

      * 7. Input Flat File - Output Data Set and Flat File;

      %let syscc=0;
      %let procedure=Input Flat File - Output Data Set and Flat File;
      proc printto log="&logtemp" new;
      run;
      filename f "&flatfile";
      filename fbak "&flatfilebak";
      data test2;
         infile f truncover end=eof;
         input &inputstatement;
         file fbak;
         put &putstatement;
      run;
      proc printto log="&printtolog";
      run;
      %let newsyscc=&syscc;
      %pinchlog(logfile=&logtemp, dsnmetrics=&dsnmetrics,
         othervars=(&pinchother));

      * 8. Input Flat File - Output Data Set;

      %let syscc=0;
      %let procedure=Input Flat File - Output Data Set;
      proc printto log="&logtemp" new;
      run;
      filename f "&flatfile";
      data test2;
         infile f truncover end=eof;
         input &inputstatement;
      run;
      proc printto log="&printtolog";
      run;
      %let newsyscc=&syscc;
      %pinchlog(logfile=&logtemp, dsnmetrics=&dsnmetrics,
         othervars=(&pinchother));

      %end;
         %end;
      %end;
   %end;
%mend;
```