Paper 2622-2018

Come On, Baby, Light my SAS® Viya®: Programming for CAS

David Shannon, Amadeus Software

ABSTRACT

This paper is for anyone who writes SAS® 9 programs and wants to learn how to take advantage of SAS Viya's Cloud Analytics Service (CAS) engine. By the end of this paper SAS 9 programmers will be familiar with CAS sessions, libraries and the scope and reuse of those sessions. A programming pattern for the CAS data lifecycle will be presented and, finally, a set of benchmark performance metrics are presented that illustrate the potential performance gained from CAS, compared to SAS 9.

INTRODUCTION

This paper was written at the time when SAS® Viya™ was entering mainstream use, and I wanted to learn how to make SAS programs execute in CAS and do so efficiently. This paper focusses on SAS Studio, the Programming environment for SAS Viya.

The paper begins by confirming that existing programs run on SAS Viya platforms without needing to modify anything. The paper continues with CAS sessions and libraries (caslibs) and how introducing these into programs exploits the performance enhancements of CAS.

With the component parts of CAS knowledge understood, the paper describes the programming pattern for analysing and processing data in CAS.

Quantifying the performance lift from in-memory processing is subject to many factors. This paper concludes with a set of tests which can be repeated in your environment. Findings are presented from my own testing.

RUNNING SAS 9 PROGRAMS IN SAS VIYA

When integrating SAS Viya into production processes, your impact assessment will be concerned with the modifications and dependencies required to run existing programs on SAS Viya.

Compatibility is key to minimise impact and aid adoption of the platform. Therefore, SAS Institute has ensured your SAS 9 programs will run as is. Only the prerequisites for your programs, i.e. external resources such as data, databases etc. need to be accessible for programs to run.

This is possible because the SAS Workspace Server sits alongside the Cloud Analytic Service of SAS Viya.

You will wish to make use of the CAS engine to benefit from the SAS Viya platform. For this, we need to start a CAS session and tell our programs to make use of CAS libraries. We do this in the sections which follow.

CAS, SESSIONS AND LIBRARIES

Consider CAS a logical server where processing is performed in memory on one or more machines. Those machines can be in the cloud or on-premise. CAS is designed to exploit the scalability of cluster computing and solve analytical problems with very large volumes of data and so is more suited to cloud than SAS 9.

When a user signs into SAS Studio (the client), they do not yet have a CAS session running. The CAS statement invokes a session, from where CAS libraries (caslibs) and traditional resources are created and used. The client can start many CAS sessions.

For completeness, it is worth noting that the client does not have to be SAS Studio but could be any SAS 9 client when signed into an intelligence platform, such as Enterprise Guide. In fact, any language which

can call SAS action sets through SAS Viya APIs can be used. Action sets are an intermediate layer between your scripting language and the executable code under the covers in SAS. For example, programmers using Python can invoke CAS sessions from Jupyter, or operational devices can make use of CAS sessions via the REST API.

Overall, a CAS session provides user identification, fault tolerance, efficiency and resource tracking (reference 1).

CAS SESSIONS

We must start a CAS session from our SAS session to do some work in memory. The session encapsulates what we do without affecting servers or other users.

SAS Studio provides code snippets to start CAS sessions. Assuming you signed into SAS Studio on a SAS Viya server, then the only syntax required to launch a CAS session is:

```
cas;
```

This creates a session with a default name (CASAUTO), on the current server with the default port and username you signed into SAS Studio with. This, and other attributes of the CAS session are shown in the log:

```
NOTE: The session CASAUTO connected successfully to Cloud Analytic Services
THELINUX using port 5570. The UUID is
       2cdc1d05-2468-904c-8269-2bef1eca4e46. The user is dshannon and the
active caslib is CASUSER(dshannon).
NOTE: The SAS option SESSREF was updated with the value CASAUTO.
NOTE: The SAS macro _SESSREF_ was updated with the value CASAUTO.
NOTE: The session is using 0 workers.
```

Figure 1: SAS Log for default CAS statement

Each CAS session has a unique identifier, or UUID. This is printed to the log and is one method of connecting to a CAS session from a different client. It is better practice to specify a CAS statement with a session name, and other appropriate options. Consider:

```
cas speedyanalytics
    sessopts=(caslib=casuser timeout=3600 metrics=true)
    uuidmac=speedyanalytics_uuid;
```

The CAS statement above achieves the following:

- Starts CAS session with the name speedyanalytics
- Sets the default caslib - used for holding data in memory - to be casuser,
- Raises the default timeout for an inactive CAS session to 60 minutes (the default is 60 seconds). This is useful when disconnecting from sessions that you will return to later.
- The metrics=true option causes expanded log information, detailing every action called and performance statistics. This is extensive with SAS procedures which call several action sets under the covers, hence I recommend using this option conservatively.
- A macro variable called speedyanalytics_uuid is populated with the UUID of the CAS session. The log is shown below:

```
NOTE: The session SPEEDYANALYTICS connected successfully to Cloud Analytic
Services THELINUX using port 5570. The UUID is
       fa1ceb75-9b3d-a443-84f0-b21be88c585e. The user is dshannon and the
active caslib is CASUSER(dshannon).
NOTE: The SAS option SESSREF was updated with the value SPEEDYANALYTICS.
NOTE: The SAS macro _SESSREF_ was updated with the value SPEEDYANALYTICS.
NOTE: The session is using 0 workers.
NOTE: 'CASUSER(dshannon)' is now the active caslib.
NOTE: Action 'sessionProp.setSessOpt' used (Total process time):
NOTE:       real time               0.005298 seconds
NOTE:       cpu time                0.005507 seconds (103.94%)
NOTE:       total nodes             1 (8 cores)
NOTE:       total memory            23.28G
NOTE:       memory                  331.22K (0.00%)
NOTE: The CAS statement request to update one or more session options for
session SPEEDYANALYTICS completed.
```

Figure 2 SAS Log for CAS statement with customized session options

You may wish to consider making a CAS statement part of your autoexec. So far, we have started two CAS sessions, but we are yet to do anything of worth with them.

The following are a list of statements which I have found useful when working with CAS sessions:

1. What CAS sessions do I have running?

```
cas _all_ list;
```

2. Get the version and license specifics from the CAS server hosting my session:

```
cas speedyanalytics listabout;
```

3. I want to sign out of SAS Studio for now, so I will disconnect from my CAS session, but return to it later…

```
cas speedyanalytics disconnect;
```

4. …later in the same or different SAS Studio session, I want to reconnect to the CAS session I started earlier using the UUID I previous grabbed from the macro variable or SAS log:

```
cas uuid="&speedyanalytics_uuid";
```

5. At the end of my program(s), shutdown all my CAS sessions to release resources on the server:

```
cas _all_ terminate;
```

When writing reusable programs, you should build in checks and balances to ensure your program is behaving as intended. To aid this with CAS statements, the macro variable CASSTMTERR is populated with 0, 1 or 2 representing success, error and warning, respectively. The standard SYSWARNINGTEXT and SYSERRORTEXT macro variables can be used to extract the appropriate messages.

There are various other features of the CAS statement. The reader is encouraged to study reference 2.

CAS LIBRARIES

A CAS library (caslib) is the way programs access data in memory.

A caslib does not just store data in memory, it also handles metadata about those tables. Specifically, the connection details to access permanent file(s) or database and the access controls associated with the data.

When a CAS session is started, the CASUSER library is made available and set as the active library. This is your personal library, similarly to the SASUSER library in a SAS session. To view and access the contents of a CAS library we need to assign a SAS library with the CAS engine option:

```
libname inmem cas caslib=casuser;
```

The statement above creates a SAS library which we can use in our programs to reference data held in memory and use the performance of CAS to process data. Now we wish to load data into a CAS table, i.e. into memory for analytics. There are different methods, each achieving the same result. Consider the following three techniques:

1. The following data step takes a sample data set, calculates a new measure and stores the output data set in memory:

```
data inmem.class;
  set sashelp.class;
  bmi=(weight*703)/(height**2);
run;
```

The efficiency of this technique should be considered. If a table is intended to be loaded into CAS, why not calculate new measures once in memory? Method 3, below, presents the most efficient method of loading data into memory, so use method 1 sparingly. However, method 1 could be helpful when a measure is derived, but columns used in the expression are dropped rather than loaded into memory.

2. Proc COPY can be used to take an existing SAS data set(s) and copy these into a caslib. The SELECT statement allows explicit control over which tables to copied in one line of syntax. Proc COPY overwrites tables if they already exist:

```
proc copy in=sashelp out=inmem;
  select classfit electric junkmail;
run;
```

3. The final option uses Proc CASUTIL which, amongst several other uses, loads tables into caslibs. In each of the examples presented, the source tables are SAS data sets. The CASUTIL procedure is not restricted to data sets. In this example two data sets are loaded from the SASHELP library into memory, then the contents of the caslib listed. Note that the REPLACE keyword is required if the intention is to overwrite tables already in the target caslib:

```
proc casutil session=speedyanalytics;
  load data=sashelp.cars replace;
  load data=sashelp.snacks replace;
  list tables;
run;
```

The output produced by the LIST statement is shown below.

The CASUTIL Procedure

Caslib Information	
Library	CASUSER(dshannon)
Source Type	PATH
Description	Personal File System Caslib
Path	/home/dshannon/casuser/
Session local	No
Active	Yes
Personal	Yes
Hidden	No
Transient	Yes

The CASUTIL Procedure

Table Information for Caslib CASUSER(dshannon)

Table Name	Label	Number of Rows	Number of Columns	Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
CLASS_SI		19	8	0	utf-8	01 March 2018 23:52:33	01 March 2018 23:52:33	No	No	No	No
CLASS		19	6	0	utf-8	02 March 2018 00:31:41	02 March 2018 00:31:41	No	No	No	No
CLASSFIT	Predicted Weights with Confidence Limits	19	10	0	utf-8	02 March 2018 00:31:41	02 March 2018 00:31:41	No	No	No	No
ELECTRIC	Electric Power Generation and Revenue	48	16	0	utf-8	02 March 2018 00:31:41	02 March 2018 00:31:41	No	No	No	No
JUNKMAIL	Classifying Email as Junk or Not	4601	59	0	utf-8	02 March 2018 00:31:41	02 March 2018 00:31:41	No	No	No	No
CARS	2004 Car Data	428	15	0	utf-8	02 March 2018 00:46:37	02 March 2018 00:46:37	No	No	No	No
SNACKS	Daily snack food sales	35770	6	0	utf-8	02 March 2018 00:46:37	02 March 2018 00:46:37	No	No	No	No

Figure 3 Output from Proc CASUTIL LIST statement

A common use of SAS programming is to modify the structure or contents of a table in preparation for analytics. For example, the following data step manipulates a table already in memory. Note this example has a farfetched use of a varchar variable; this is only to alert the reader that CAS tables bring different data types other than numeric and character:

```
data inmem.classsi;
  set inmem.class;
  length namev varchar(8);
  namev=strip(name);
  weightsi = (weight * 0.4536);
  heightsi = (height * 0.0254);
  bmisi    = (weightsi/(heightsi**2));
  drop name;
run;
```

I wish to preserve the newly modified table “classsi” for future use. Therefore, I save it to the source disk location associated with the caslib. Proc CASUTIL does this with the SAVE statement:

```
proc casutil;
  save casdata="classsi" casout="classsi";
run;
```

This creates a SAS High Performance Data file (.sashdat) on disk. This is the default file type for SAS Viya (I believe it is a memory mapped structure) and data are loaded into caslibs extremely efficiently. On disk, the file appears as follows:

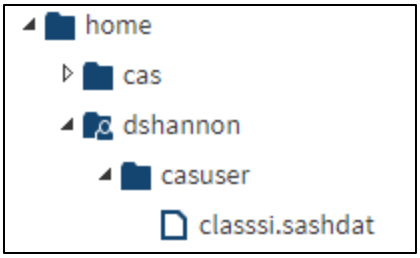

Figure 4 File system view of tables saved from CAS to file store

In a future CAS session, the preserved `classsi` data can be reloaded into memory as follows:

```
proc casutil;
  load casdata="classsi" casout="classsi";
run;
```

Note, the names when in memory and on disk do not have to match. However, I would suggest there be a logical reason for these. For example, a table holding “sales” represents the latest three months data, whereas on disk, there may be many historical copies of the file representing sales at different points in time.

There are always two scopes for CAS libraries:

1. Session: Data is accessible to your CAS session only. You can conduct analysis securely and without concern for another process updating a table mid-flight through your analysis.
2. Global: All CAS sessions can access tables in the caslib. Sharing data is efficient when tables are read by several users. Global caslibs are automatically available when a CAS session starts. Public and Formats are examples of default global caslibs. Global caslibs are created by authorised users, e.g. the operating system “CAS” account, or other users who are granted the permission.

   Access to tables in the public caslib can be achieved with a SAS libname similarly to:

   ```
   libname public cas caslib=public;
   ```

Whether scope is *session* or *global*, access to a table in the caslib is subject to access controls.

3. Personal: There may be a third scope of caslib, depending on decisions made when your software was deployed. As with global caslibs these are automatically available when a CAS session starts, but they are personal to users.

Caslibs become a versatile and powerful tool when combined with data connectors. This allows external data sources to be loaded and saved to memory for analytics. Data connectors are available for, and not least: Amazon Redshift, DB2, Hadoop, Microsoft SQL Server, Oracle and SAP HANA. The reader is advised to research the appropriate data connector for their environment.

Further capabilities of caslibs are not covered within this paper. Specifically, formats and grouping of tables by variables for partitioned processing. These topics should be researched accordingly.

PROGRAMMING PATTERN FOR CAS

So far in this paper we have started a CAS session and loaded data into memory. Figure 5 presents a pattern to apply in SAS programs which puts CAS to work.

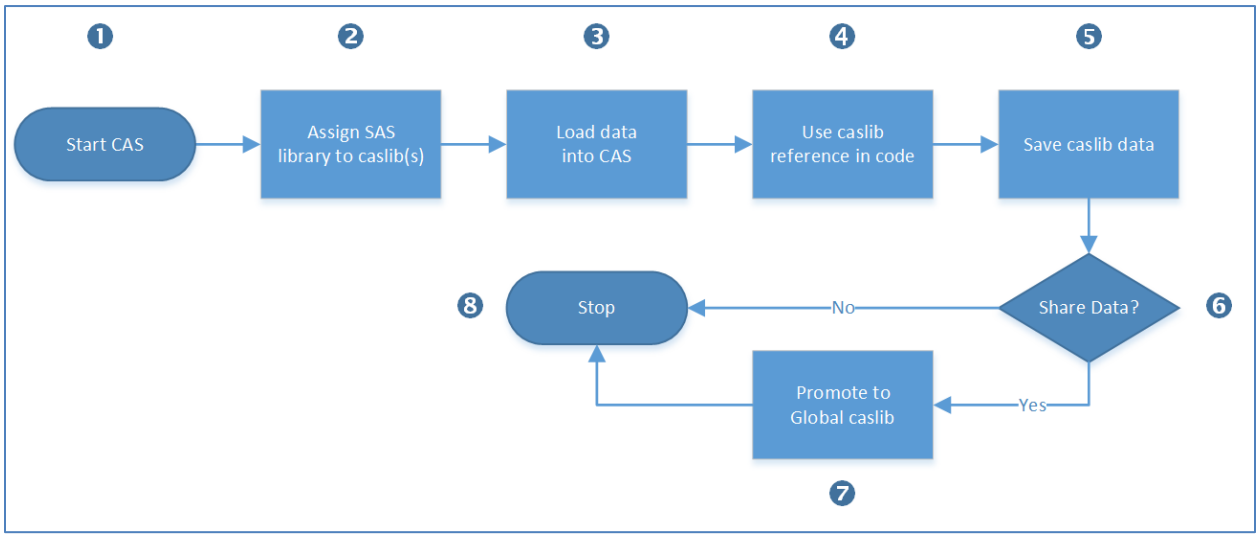

Figure 5 Programming for CAS

The steps are described as follows:

1. Start CAS: Using the CAS statement discussed earlier in this paper, consider appropriate session options, such as extending default timeouts if you expect sessions to be idle for extended periods.
2. Assign SAS library to caslib(s): Using libname statements with the CAS engine, connect your SAS session to CAS libraries. Consider the librefs as they should be unique to avoid hiding global caslibs. Also consider who else will be running your program. If it will be run in batch can the libraries be accessed by the account hosting the batch session?
3. Load data into CAS: Use Proc CASUTIL to load data from saved data stores, SAS data sets or another ODBC or flat file format. Consider using other SAS programming statements to load data into CAS when specific reason requires it.
4. Use caslib reference in code: Whether you have pre-written programs or are writing from scratch, we can add the libref used to access the caslib into our programs.

   To make use of the CAS session, the proc or data step must both read and write to a caslib. If not, processing defaults to a standard SAS session. Typically, by replacing where the WORK library was used (or implied) with the caslib reference, we will get a marked performance improvement.

   In circumstances where your tables are very large they and will be split over many nodes for parallel processing. You may wish to prevent this behaviour when your data step has inter-row dependencies in its expressions or calculations. Specifically, if your program works with LAG, DIF functions, using the automatic iteration counter _n_ or automatic first. and last. variables. These could yield results you would not expect because tables are partitioning across many compute nodes. Therefore, use the SINGLE=YES data step option to force single threaded processing:

```
data inmem.failure2 / single=yes;
set  inmem.failure;
  total+_n_;
run;
```

Any Proc SORT steps can be commented out when running in CAS as sorting is performed in memory. Even when Proc SORT is used for deduplication; more efficient methods are presented for CAS programming (reference 3).

A paper by Phil Weiss (reference 3) provides comprehensive and detailed technical reference.

5. Save caslib data: If tables in caslibs are updated, or new tables created, they should be saved to the caslib storage area (be that file system, data lake, etc.) if that copy of that tables is required again in the future.
6. Share data: Should the table(s) created by your program be shared with other users? If yes, review step 7:
7. Promote data to global caslibs: For example, data which is prepared for use in Visual Analytics, Visual Statistics, etc. should be promoted into caslibs where shared reports and models can read it. Promoting tables has not been discussed so far in this paper, hence the following example demonstrates promoting a table to the Public caslib:

```
proc casutil outcaslib="public”;
   promote casdata="classsi;
run;
```

You may also use the PROMOTE= dataset option.

8. Stop: If your program is the last in your SAS session, terminate your CAS session. If more programs follow and reuse the same CAS session, consider the size of your tables in relation to the resources available, saving and dropping tables from memory to release resources.

PERFORMANCE ADVANTAGE

To understand the performance benefit gained from CAS, an experiment was run with comparable functionality between SAS 9.4M5 and SAS Viya 3.3. A set of basic programming tasks using a randomly generated 5GB data set was used. Programs for these tests are provided in Appendices A and B for SAS 9 and SAS Viya, respectively.

Each step was repeated three times. The mean results are presented in in the table below:

Step	SAS 9.4		SAS Viya 3.3		Viya Benefit	
	Real Time (s)	CPU Time (s)	Real Time (s)	CPU Time (s)	Real Time	CPU Time
1. data _null_	16.12	16.11	14.78	14.78	6%	6%
2. data (Read)	23.64	23.39	12.58	0.01	47%	100%
3. data (R+W)	13.00	12.99	12.90	2.14	6%	84%
4. summarize	10.38	10.75	2.75	0.00	74%	100%
5. transpose	22.77	22.31	4.53	0.34	80%	98%

Table 1 Performance Test Results

Test 1: Designed to verify that CPU performance is comparable between environments and tests (reference 4). Equal results indicate that the environment is performing consistency for each of SAS Viya and SAS 9.4.

Test 2: Designed to test the repetitive reads expected of tables stored in memory. On average, read times in SAS Viya are half that of SAS 9.

Test 3: Designed to test the data preparation tasks performed on tables held in memory. The elapsed time is not significantly different between SAS Viya and SAS 9 in my tests. This is thought to be an anomaly of my test setup.

Test 4: Designed to test basic analytical aggregation. Proc SUMMARY is used in SAS 9.4, whilst Proc MDSUMMARY is used in SAS Viya. Proc MDSUMMARY performs far more efficiently than Proc SUMMARY.

Test 5: Designed to test a transpose task. A significant benefit was observed with CAS.

Tests were performed on a single machine configured with SAS 9.4M5 Foundation and SAS Viya 3.3 Programming Environment, specification as follows:

- Processor: Intel® i7-7700T, 2.9GHz, 8 cores with hyperthreading
- RAM: 23GB allocated
- Operating System: Oracle Linux Server 7.4
- Chassis: VM (Microsoft Hyper-V)
- Storage: Crucial MX300 525GB SATA Internal SSD.

This device is lower than an enterprise class specification, but quite fit for a single user.

EXCEPTIONS

The log tells us clearly when CAS is processing the steps we have programmed. The notes highlighted below are an example of CAS processing, but processing in a single thread:

```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step has no input data set and will run in a single thread.
NOTE: The table datawrite in caslib CASUSER(dshannon) has 1000000000
observations and 2 variables.
NOTE: DATA statement used (Total process time):
      real time           1:45.17
      cpu time            0.00 seconds
```

Figure 6 SAS log with CAS processing on a single thread

Optimally, the log below shows CAS processing in multiple threads. In massively parallel computing environments, i.e. multiple machines, further performance benefit will be observed:

```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
NOTE: There were 1000000000 observations read from the table DATAWRITE in
caslib CASUSER(dshannon).
NOTE: DATA statement used (Total process time):
      real time           14.79 seconds
      cpu time            0.01 seconds
```

Figure 7 SAS log with CAS processing over multiple threads

There could be situations where SAS determines that CAS is not to be used. In those situations, SAS processes as normal (without any user intervention) and the log omits statements regarding cloud analytic services.

CONCLUSIONS

The objective of this paper is to enable the reader to understand CAS, caslibs and to make productive use of their performance benefits.

We have seen that existing SAS 9 programs can run “as is” on SAS Viya. This is important in environments where modifying validated programs is expensive. Options for controlling the multithreading of data were introduced and the reader recommended to explore the ability to process data in groups for efficiency. The benefits from CAS are achieved by loading data in a CAS library (caslib) and referring to the caslib in our programs.

Caslibs can be used to access stored data from a variety of systems via data connectors. Tables stored in caslibs may have other data types such as varchar for efficiency.

A pattern for programming with CAS was presented. This integrates the steps for loading and unloading data from memory. Sharing data via global caslibs was discussed.

The potential performance gain from CAS was explored, with a relatively simple experiment. Tests for read, write and analytics were conducted with a single 5GB table. Performance is of course dependent on many factors and this test was conducted in a controlled, artificial environment. Should this test be representative for SAS Viya, results indicate approximately a halving of elapsed time in jobs on a single server.

REFERENCES

1. SAS Institute Inc. 2017. SAS® Cloud Analytic Services 3.3: Fundamentals. Cary, NC: SAS Institute Inc.
2. SAS Institute Inc. 2017. SAS® Cloud Analytic Services 3.3: User’s Guide. Cary, NC: SAS Institute Inc.
3. Weiss P., 2018, Getting Your SAS® 9 Code to Run MultiThreaded in SAS® Viya® 3.3, SAS Institute.
4. Hughes T., 2016, SAS Data Analytical Development: Dimensions of Software Quality. Wiley.

CONTACT INFORMATION

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

David Shannon
Amadeus Software
+44 (0)1993 848010
david.shannon@amadeus.co.uk
www.amadeus.co.uk

APPENDIX A: PERFORMANCE TESTING SAS 9

The following listing is the program used to represent a set of data and proc steps to evaluate performance in SAS 9.4M5:

```
libname pfc "/home/dshannon/pfc";

* Create a sample data set;
/*
   For ~500MB %let obs=13200
   For ~1GB   %let obs=23600
   For ~5GB   %let obs=132000
   For ~10GB  %let obs=263000
*/
%let obs=132000;
data pfc.canary(drop=a b i j x );
    array characters{150} $255;
    array numbers{150};
    array chars{255} $1 _temporary_;
    a=rank('a');
    b=rank('z');
    do z=1 to &obs.;
      do i=1 to 150;
        do j=1 to 255;
          x=round(a+(b-a)*ranuni(0),1);
          chars{j}=byte(x);
        end;
        characters{i}=cats(of chars{*});
        numbers{i}=round(100*uniform(0),1);
      end;
      output pfc.canary;
    end;
run;

* Test 1: data _null_;
data _null_;
  do i = 1 to 1000000000; * 1 billion rows;
    n1=rand('uniform');
    output;
  end;
run;

* Test 2: prepare for data read;
data work.dataread;
  do i = 1 to 1000000000; * 1 billion rows;
    n1=rand('uniform');
    output;
  end;
run;

*data (read);
data _null_;
set work.dataread;
run;

proc datasets lib=work kill nodetails nolist;
```

```
run;quit;

* Test 3: data step (read+write);
data pfc.canary2;
set  pfc.canary;
run;

proc delete data=pfc.canary2;
run;

* Test 4: proc summary;
proc summary data=pfc.canary;
  class characters1 characters2 characters3 ;
  var numbers1-numbers10;
  output out=pfc.canary_summary;
run;

* Test 5: proc transpose;
proc transpose data=pfc.canary out=pfc.canary_transposed;
  id z;
run;
```

APPENDIX B: PERFORMANCE TESTING SAS VIYA

The following listing is the program used to represent a set of data and proc steps to evaluate performance in SAS Viya:

```
* Start CAS session and assign libraries;
cas benchmark;
libname inmem cas;
libname pfc "/home/userid/pfc";

* Create a sample data set;
/*
   For ~500MB %let obs=13200
   For ~1GB   %let obs=23600
   For ~5GB   %let obs=132000
   For ~10GB  %let obs=263000
*/

%let obs=132000;
data pfc.canary(drop=a b i j x ) ;
    array characters{150} $255;
    array numbers{150};
    array chars{255} $1 _temporary_;
    a=rank('a');
    b=rank('z');
    do z=1 to &obs.;
      do i=1 to 150;
        do j=1 to 255;
          x=round(a+(b-a)*ranuni(0),1);
          chars{j}=byte(x);
        end;
        characters{i}=cats(of chars{*});
        numbers{i}=round(100*uniform(0),1);
      end;
      output pfc.canary;
    end;
run;
*/

* Test 1: data _null_;
data _null_;
  do i = 1 to 1000000000;
    n1=rand('uniform');
    output;
  end;
run;

* Test 2: prepare for data read;
data work.dataread;
  do i = 1 to 1000000000; * 1 billion rows;
    n1=rand('uniform');
    output;
  end;
run;

proc casutil outcaslib=casuser;
```

```
  load data=work.dataread casdata="dataread" replace;
run;

*data (read);
data _null_;
set inmem.dataread;
run;

proc casutil session="benchmark";
  droptable casdata="dataread";
run;

* Test 3: Data (read+write);
proc casutil outcaslib=casuser;
  load data=pfc.canary casdata="canary" replace;
run;

* Test 4: proc mdsummary;
proc mdsummary data=inmem.canary;
  groupby characters1 characters2 characters3;
  var numbers1-numbers10;
  output out=inmem.canary_summary ;
run;

* Test 5: proc transpose;
proc transpose data=inmem.canary out=inmem.canary_transposed;
  id z;
run;

* Terminate the CAS session;
cas benchmark terminate;
```