Paper SAS 5332-2020
Coding in SAS® Viya®
Charu Shankar, SAS Institute Inc.

ABSTRACT
This hands-on workshop is for users who want to take advantage of the boost in processing speed for Base SAS programs executing in SAS Viya. This paper covers using the power of SAS Cloud Analytic Services (CAS) to access, manage, and manipulate in-memory tables. The purpose of this paper is to support users in their ability to get started with coding in SAS Viya as they transition from coding in Base SAS programs. Users learn to perform three simple yet important tasks to get comfortable with the language of SAS Viya: connect to the CAS LIBNAME engine for data transfer between SAS and CAS; load data to a caslib and process data in CAS; and modify SAS programs to run in SAS Viya.

INTRODUCTION
SAS Viya is a cloud-enabled, in-memory analytics engine that provides quick, accurate, and reliable analytical insights. The latest enhancement of the SAS Platform, SAS Viya is an open, cloud-enabled, analytic run-time environment with a number of supporting services. One of those supporting services is SAS Cloud Analytic Services, or CAS. CAS provides a powerful in-memory engine that delivers blazing speed to accurately process your big data. It uses scalable, high-performance, multi-threaded algorithms to rapidly perform analytical processing on in-memory data of any size. Data used in this paper can be downloaded from this Github Repository https://github.com/CharuSAS/CodingInSASViya.

Terminology

**File** is used to refer to the source data that is in a caslib’s data source. For a caslib that uses a path-based data source, this is natural. For a caslib that uses a database as a data source, the tables in the database are referred to as files.

**Table** is used to refer to in-memory data. After a file (using the preceding definition) is loaded into the server, it is referred to as a table.
**CAS** - in SAS Viya, SAS Cloud Analytic Services (CAS) is the star of the show, providing lightning fast analytics of in-memory data for SAS Visual Analytics and other software offerings.

**SPRE** - Foundation SAS, the long-time workhorse of SAS analytics is also offered, referred to as the SAS Programming Runtime Environment (SPRE). SPRE provides a user interface and data processing environment for executing classic SAS program code. It offers the Foundation SAS software that we're all familiar with, including Base SAS, SAS/ACCESS engines, and more, as well as the SAS Studio web application.

SAS Viya is not a replacement for SAS®9. You can still leverage your SAS programming knowledge and make modifications to existing SAS code to enable it to run in SAS Viya.

**SAS Programming Interfaces: SAS Studio Login**

SAS Studio is the SAS language code editor in SAS Viya. With the latest release of SAS Viya, you can also use SAS Enterprise Guide 7.15 and SAS 9.4M5 to submit code to CAS.

Launch Google Chrome. Click the **SAS Viya** tab below the address bar and select **SAS Studio**. Sign in to SAS Studio with the following credentials:

<table>
<thead>
<tr>
<th>User ID</th>
<th>Password</th>
</tr>
</thead>
<tbody>
<tr>
<td>lynn</td>
<td>Student1</td>
</tr>
</tbody>
</table>
1. CONNECT TO CAS

After a CAS session starts, you can write code and submit CAS-enabled procedures.

1.1 CONNECT TO CAS SERVER

Here is the code to connect to the CAS server:

```sas
 cas mySession sessopts=(caslib=casuser timeout=1800 locale="en_US");
```

Display 1: Log showing That the CAS Session MYSESSION Started Successfully

1.2 ACCESSING CASLIBS

A caslib is a container for both the files in the caslib’s data source and the in-memory tables that you load from the data source.

Here is the code to list caslibs:

```sas
 caslib _all_ list;
```

Display 2: Log Verifying That Casuser Is the Active Caslib
Here is the code to list the files in `casuser`, which is the default caslib or active caslib:

```sas
proc casutil;
    list files;
quit;
```

On the Results tab, observe that the `sales.xlsx` file is listed.

Here is the code to create a new caslib:

```sas
caslib mycas path="/workshop/casfiles";
```

NOTE: 'MYCAS' is now the active caslib.
NOTE: Cloud Analytic Services added the caslib 'MYCAS'.
NOTE: Action to ADD caslib MYCAS completed for session MYSESSION.

**Display 3: Log to Verify That MYCAS Is the Active Caslib**

In the navigation pane, select Libraries ➔ My Libraries. The `mycas` library is not visible. By default, the caslibs do not show up in the Libraries tree in SAS.

Here is the code to assign a new caslib:

```sas
caslib _all_ assign;
```

In the navigation pane, select Libraries ➔ My Libraries. The `mycas` library should be visible. Using the CASLIB statement with the ASSIGN option and the _ALL_ keyword assigns SAS library references for existing caslibs for visibility in the Libraries tree in SAS Studio. Because CAS processes only in-memory tables, tables are loaded into memory before they are used in CAS.

### 1.3 Changing an Active Caslib

There is only one active caslib at a time in a CAS session. The active caslib is where data is processed by default.

When Lynn starts a session, her personal caslib, `casuser(Lynn)`, is defined by default and is the active caslib. She can use it to access the files in the directory /home/lynn/casuser.
If Lynn wants to access data in another directory, she can use the CASLIB statement, and the newly defined caslib becomes the active caslib.

If Lynn wants to access data in another directory, she can use the CASLIB= option in the CAS statement, and the referenced caslib becomes the active caslib.

When Lynn clears the active caslib, her default caslib becomes active again.

2. LOAD DATA TO A CASLIB AND PROCESS DATA IN CAS

In the first section, we started a CAS session and accessed a caslib. Because CAS can process data only in its in-memory space, the next step is to load your file into memory. Subsequently, data updates and analysis can begin.

Source data files mapped to a caslib are referred to as server-side files. These files can be rapidly loaded into the caslib’s in-memory space for processing. After a file is loaded into memory, it is referred to as a table. These CAS tables are in-memory copies of the associated CAS file. The source data files remain on disk and unchanged.
2.1 SESSION SCOPE VERSUS GLOBAL SCOPE

In-memory tables can have either session scope or global scope.

By default, in-memory tables have session scope. A session-scope table is accessible only in the CAS session where it was created. It's only visible to the user who created it. Session-scope tables are useful for ad hoc data access and analysis because they don't require access control checks or locking for concurrent access.

A session-scope in-memory table exists only for the duration of the session. When the CAS session ends, the table is dropped.

To share data across your sessions or with other users, create a global-scope table, also called a promoted table. You can promote a table when you load a file into memory or promote an in-memory session table. After a session-scope table is promoted, it's visible across CAS sessions.

Unlike session-scope tables, global-scope tables are not dropped from memory when a CAS session ends. The table is still available to other sessions and will be available in the next CAS session that the user starts.

To understand the concepts of session versus global scope, we will load the client-side SAS data set, mysas.employees, into Lynn’s personal caslib. Lynn creates an in-memory session-scope table, myemployees. This is a table that others will need to access. When Lynn is happy with the contents of the session-scope table, she uses the PROMOTE statement to create a global-scope table in the public caslib so that other users can use the in-memory table.

Here is the code to load data, create a session scope table, and promote it:

```plaintext
proc casutil;
  load data=sashelp.cars outcaslib=casuser
casout="MyCars" replace;
  load data=mysas.employees outcaslib="casuser"
casout="MyEmployees" promote ;
quit;
proc casutil;
  list tables incaslib="casuser";
quit;
```

The log shows that CAS processed the code, and the results show specific metadata about the myemployees and mycars tables.
Display 4: Metadata about the myemployees and mycars Tables

The mycars table is session scope (**Promoted Table=No**) and is dropped from memory when the CAS session ends.

The myemployees table is global scope (**Promoted Table=YES**) and is **not** dropped from memory when the CAS session ends.

### 2.2 LOADING CLIENT-SIDE DATA INTO CAS

Files of any type that are not mapped to the caslib are called *client-side* files. You load files into the in-memory space in CAS using your client software. To load a client-side SAS data set that resides in a SAS library, we use the CASUTIL procedure LOAD statement with the **DATA= option.**

We proceed to load the client-side SAS data set, mysas.employees (mysas is a local SAS library), into a CAS table in Lynn’s personal caslib and investigate the contents of the in-memory table.

Here is the code to load **mysas.employees** into a CAS table in Lynn’s personal caslib:

```sas
proc casutil;
   load data=mysas.employees casout="Employees" replace;
quit;
```

In the navigation pane, in the SAS Studio tree, expand **Libraries ⇒ My Libraries ⇒ CASUSER** to verify that **employees** was loaded into the casuser caslib.
Here is the code to verify that employees was loaded into the casuser caslib:

```sas
proc casutil;
   list tables incaslib="casuser";
quit;
```

**Display 5: Log Indicating That CAS Processed the Code**

The employees table is session scope (**Promoted Table=No**) and is dropped from memory when the CAS session ends.

```sas
proc casutil;
   contents casdata="employees" incaslib="casuser";
quit;
proc contents data=casuser.employees varnum;
run;
```

Let’s compare the results. SAS Viya has additional data types. The data type DOUBLE maps to a SAS numeric data type. SAS Viya also supports CHAR, which is the SAS character fixed-width data type, and the VARCHAR data type, which is a variable-length character field.
Display 7: Partial PROC CASUTIL Results

Display 8: Partial PROC CONTENTS Results
2.3 LOADING SERVER-SIDE FILES INTO CAS AND PROMOTING TABLES

What if you need to load a server-side file that's stored in the caslib’s data source? You use the CASUTIL procedure LOAD statement with the CASDATA= option.

![Diagram of Loading Data into Memory in CAS](image)

The CASUTIL procedure always uses the active caslib. As a best practice, always specify the caslib explicitly with the INCASLIB= and OUTCASLIB= options.

By default, the in-memory table will have the same name as the original file, but you can use the CASOUT= option to specify a different name.

Let’s consider the sales.xlsx file that exists in the data source of Lynn’s personal caslib. Lynn creates an in-memory session-scope table, salesxlsx, and investigates the table’s descriptor portion. This is a table that others will need to access. When Lynn is happy with the contents of the session-scope table, she uses the PROMOTE statement to create a global-scope table in the public caslib so that other users can use the in-memory table.

Here is the code to create an in-memory session-scope table:

```sas
proc casutil;
   load casdata="sales.xlsx" incaslib="casuser"
       outcaslib=casuser
       casout="salesxlsx" replace;
   contents casdata="salesxlsx" incaslib="casuser";
run;
```
Display 9: Output from PROC CASUTIL
Examine the results to see the descriptor portion of the `salesxlsx` in-memory table. This is a session-scope table as noted by the value of `No` for Promoted Table. Also note that when the `salesxlsx` table was stored in CAS, the character values were converted to the VARCHAR data type.

Here is the code to create `salesxlsx` as a global-scope table in the `public` caslib:

```plaintext
proc casutil;
    promote casdata="salesxlsx" incaslib="casuser"
        outcaslib="public" casout="salesxlsx";

    list tables incaslib="public";
quit;
```

Display 10: Results Showing the In-Memory Tables in the Public Caslib
3. MODIFYING SAS PROGRAMS TO RUN IN SAS VIYA

3.1 LOADING SERVER-SIDE FILES INTO CAS AND PROMOTING TABLES

When a DATA step is executed in Base SAS, it runs in a single thread on the SAS Workspace Server. Processing data in a single thread reads data sequentially, one row at a time.

SAS Viya enables data to be divided and processed simultaneously on multiple threads. When a DATA step executes in CAS, each thread executes the program statements on its data and returns the results to the controller.

The threads might receive different amounts of data, and might complete their processing and return the results in a seemingly random order. SAS Viya reassembles the results. We’ll look at examples where the parallel processing is transparent to the user. The only difference you’ll see is faster execution. We’ll also look at situations where you, as the programmer, need to take additional action to summarize the results from the threads.

Let’s compare Base SAS execution with that of SAS Viya.

Here is the code to run Base SAS running a simple program in Base SAS using a single thread:

```sas
data _null_;      
  put "Processed on " _threadid_=' _nthreads_=';  
run;
```
The _THREADID_ value is 1, which indicates that the DATA step processed in SAS is a single thread. The value of _NTHREADS_ is 1, which indicates that there is one thread available in the Base SAS session for processing the code.

```sas
73   data _null_;  
74       put "Processed on " _threadid_= _nthreads_=;  
75       run;  
```

Processed on _THREADID_=1 _NTHREADS_=1

Display 11: Log Indicating Single-Thread Processing in the Base SAS Session

Here is the code to run a DATA Step in the CAS session:

```sas
data _null_/sessref="MySession";  
    put "Processed on " _threadid_= _nthreads_=;  
run;
```

The first note confirms that the DATA step was executed in CAS.

One big difference is the fact that _THREADID_ is equal to different values for each row in the log. The threads operate independently. Therefore, the log messages were generated by each thread at slightly different times. The values represent the thread on which the DATA step was executed in the CAS session. There are 16 threads available (_NTHREADS_=16). In this execution of the code, thread 3 completed the execution first, and then thread 8, and so on.

If you run the program multiple times, you might get a different order each time that the program runs. This is exactly what we want to happen when a program is executed in multiple threads. Otherwise, the performance gains by threading are lost if the DATA step were to somehow synchronize the output to the log.

Display 12: Log Indicating Multi-Thread Processing in the CAS Session
3.2 MODIFYING DATA STEP CODE TO RUN IN SAS VIYA: NEW VARIABLES

Sometimes to get the DATA step to process in CAS, it's as simple as modifying the library reference in the DATA statement and the SET statement to use a caslib. When both the output and input tables are CAS tables, the DATA step processes in CAS.

Let's look at a DATA step that creates a variable conditionally using a SELECT statement. We will modify the Base SAS DATA step to run in multi-threaded environment in CAS. Before the program can run in CAS, ensure that the SAS table mysas.customers is loaded into CAS memory as global-scope table mycustomers in Lynn's casuser caslib.

```sas
    data casuser.Departments;
       set casuser.mycustomers end=eof;
       select(Continent);
       when ('Africa', 'Asia') Department="General Shoes";
       when ("Oceania") Department="Men's Shoes";
       when ("North America", "Europe") Department="Women's Shoes";
       otherwise Department='Unknown';
       end;
       keep City Continent Department;
       if eof then put _threadid_= _N_=;
    run;
```

Display 13: Log Showing Program runs in a Single Thread

Here is the code to modify a DATA step to run in CAS:

```sas
    data casuser.Departments;
       set casuser.mycustomers end=eof;
       select (Continent);
       when ('Africa', 'Asia') Department="General Shoes";
       when ("Oceania") Department="Men's Shoes";
       when ("North America", "Europe") Department="Women's Shoes";
       otherwise Department='Unknown';
       end;
       keep City Continent Department;

       if eof then
          put _threadid_= _N_=;
    run;
```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.

<table>
<thead>
<tr>
<th>THREADID</th>
<th>N</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>60000</td>
</tr>
<tr>
<td>12</td>
<td>59000</td>
</tr>
<tr>
<td>1</td>
<td>60000</td>
</tr>
<tr>
<td>7</td>
<td>60000</td>
</tr>
<tr>
<td>15</td>
<td>59000</td>
</tr>
<tr>
<td>9</td>
<td>59000</td>
</tr>
<tr>
<td>10</td>
<td>59000</td>
</tr>
<tr>
<td>13</td>
<td>59000</td>
</tr>
<tr>
<td>16</td>
<td>58669</td>
</tr>
<tr>
<td>2</td>
<td>60000</td>
</tr>
<tr>
<td>14</td>
<td>59000</td>
</tr>
<tr>
<td>11</td>
<td>59000</td>
</tr>
<tr>
<td>4</td>
<td>60000</td>
</tr>
<tr>
<td>5</td>
<td>60000</td>
</tr>
<tr>
<td>8</td>
<td>60000</td>
</tr>
</tbody>
</table>

NOTE: There were 951,669 observations read from the table MYCUSTOMERS in caslib CASUSER(lynn).
NOTE: The table Departments in caslib CASUSER(lynn) has 951,669 observations and 3 variables.
NOTE: DATA statement used (Total process time):
  real time 0.19 seconds
  cpu time 0.01 seconds

Display 14: Log Showing That the Program Runs in Multiple Threads

The data was distributed across the 16 threads in the CAS session. The results were returned as each thread completed its processing. Thread three completed first after processing 60,000 rows, and then thread 12, and so on. If you were to add up all the values of _N_, the sum would equal 951,669, which is the total number of rows that were read from the casuser.mycustomers table. Also note that the first row value for City listed in the casuser.Departments table is not the same as the first row returned in work.departments.
3.3 MODIFYING DATA STEP CODE TO RUN IN SAS VIYA: BY STATEMENT

If you are using the DATA step to process in groups or merge data based on the value of one or more variables, then you would have to first sort the data and then use the BY statement and FIRST. and LAST. processing to identify the first and last row in each group. Sorting can be a very resource intensive, especially with very large data sets, and when the DATA step is processed in Base SAS, the rows are processed sequentially in a single thread.

The default when data is loaded into CAS is to distribute the input data based on the original order among the different threads or multiple machines. The DATA step is executed among the different threads or on multiple machines.

When a BY statement is added to the DATA step, the rows are group based on the first BY variable and then distributed across multiple threads or machines. And because the data is distributed based on the value of the BY variable, PROC SORT is no longer necessary. The DATA step with the BY statement executes on each thread. Results are returned as each thread finishes processing. Thread 3 processes the DATA step and returns the results first. The order might be different each time that the program executes.

Here is the code to create a table with the total cost for each continent:

```sas
proc sort data=mysas.customers out=customers;
   by Continent;
run;

data work.CityTotals;
   set customers;
   by Continent;
      if first.Continent then TotalCost=0;
      TotalCost+Cost;
      if last.Continent then output;
      keep Continent TotalCost;
      format TotalCost dollar15.2;
run;
```
Here is the code to run in CAS in multiple threads:

```plaintext
data casuser.CityTotals;
    set casuser.mycustomers;
    by Continent;

    if first.Continent then TotalCost=0;
    TotalCost+Cost;

    if last.Continent then output;
    keep Continent TotalCost;
    format TotalCost dollar15.2;
run;
```

<table>
<thead>
<tr>
<th>Continent</th>
<th>TotalCost</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Europe</td>
<td>$50,600,011.32</td>
</tr>
<tr>
<td>2 Africa</td>
<td>$87,096.20</td>
</tr>
<tr>
<td>3 Asia</td>
<td>$125,983.80</td>
</tr>
<tr>
<td>4 North America</td>
<td>$18,518,294.90</td>
</tr>
<tr>
<td>5 Oceania</td>
<td>$4,666,493.04</td>
</tr>
</tbody>
</table>

**Display 15: Output Data for Work.citytotals Showing Values Ordered by Continent**

**Display 16: Same Results for TotalCost but Rows Not Returned in Sorted Order by Continent**
CONCLUSION
This paper attempted to showcase the power of SAS Viya and CAS from assigning libraries, to moving data and manipulating it. Performance benefits were highlighted so that readers weighing options can perhaps begin to consider SAS Viya for their daily data work.

ACKNOWLEDGMENTS
The author is grateful to the many SAS users that have entered her life. Charu is grateful to the SAS Global Forum User Committee for the opportunity to present this paper. She would also like to express her gratitude to her manager, Stephen Keelan, without whose support and permission this paper would not be possible.

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

Charu Shankar
SAS Institute Canada, Inc.
Charu.shankar@sas.com
https://blogs.sas.com/content/author/charushankar/

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.
Other brand and product names are trademarks of their respective companies.

REFERENCES
CAS Concepts
https://go.documentation.sas.com/?docsetId=calserverscas&docsetTarget=n05000viyaservers000000admin.htm&docsetVersion=3.4&locale=en

An Introduction to SAS Viya 3.4 Programming
https://go.documentation.sas.com/api/docsets/pgmdiff/3.4/content/pgmdiff.pdf

Differences in the SAS®9 and SAS Viya 3.1 Platforms
https://go.documentation.sas.com/api/docsets/whatsdiff/3.1/content/whatsdiff.pdf?locale=en#nameddest=n0evbd1ha0clqv1sbz5yag06xi6

SPRE (SAS Programing Runtime Environment)

SAS Cloud Analytic Services 3.1: Language Reference
https://go.documentation.sas.com/api/docsets/casref/3.1/content/casref.pdf?locale=en#nameddest=p05ccny5glgwan19mikixi8z1jk