PharmaSUG 2018 - Paper AD-23

Let's Start Something New! A Beginner's Guide to Programming in the SAS® Cloud Analytic Services Environment

Kevin Russell and Gerry Nelson, SAS Institute Inc.

ABSTRACT

Starting something new (whether it is a home project or a work project) is not always the easiest thing to do. However, more often than not, starting something new can be very rewarding. So while you might be a little hesitant to start programming with SAS® Cloud Analytic Services (CAS), you will likely find it extremely beneficial! CAS is a server that provides a cloud-based, run-time environment for analytics and data management. The server can run as a distributed server on multiple machines with multithreaded processing, which means that it can perform extremely high-performance analytics against very large tables.

The purpose of this paper is to help you take that first step toward programming with CAS. This paper provides a complete example of programming in the CAS environment, from showing you how to establish a CAS session to running one of the many high-performance procedures that are available in CAS. The discussion also covers other details that you need in order to start programming in CAS, including everything from loading files into CAS to manipulating the data with the SAS® DATA step. The paper also discusses the CAS and CASUTIL procedures. PROC CAS (the procedural interface to the CAS server) uses the new CASL language that enables you to perform actions on the CAS server.

So get ready to take that first step and start something new!

INTRODUCTION

CAS is the cloud-based, run-time environment for data management and analytics that is available through SAS Viya. CAS is a server that processes data in memory and enables you to quickly produce accurate and consistent results no matter how complex your analytical workload is.

CAS can run either on a single machine or on multiple machines as a distributed server. In either case, the servers can be on-site, be cloud-based, or use a combination of locations. The distributed server includes a controller (Session Controller in Figure 1) and one or more worker nodes (Session Worker in Figure 1), as shown in Figure 1 below. The controller node communicates with client applications and controls the processing being performed by the worker nodes. The worker nodes can consist of one or more machines. Each of the worker nodes performs analysis on the rows of data that are in-memory on that node.

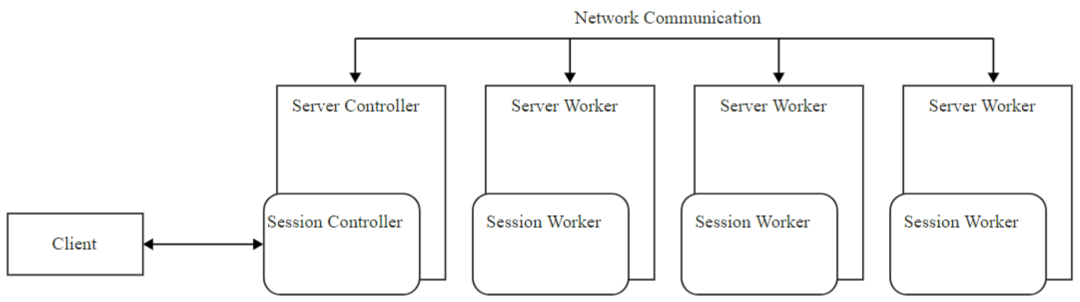


Figure 1. Distributed Server Architecture (from SAS® Cloud Analytic Services 3.3: Fundamentals)

Benefits of CAS:

- Much quicker processing for analytics: CAS can be set up as a distributed server that consists of multiple machines. The ability to distribute the workload across multiple machines can provide extremely quick processing for all your large-scale analytics needs.
- Fault-tolerant processing: With CAS, there is no need to worry if one or more of the nodes becomes unresponsive because the distributed servers are fault tolerant. If the controller node is no longer able to communicate with a worker node, another worker node takes over the analysis using a copy of the data.
- Scalable: The distributed server configuration scales horizontally, which means that more worker nodes can be added to distribute the processing and improve processing time.
- Standardized, open code base: Some programmers might be more familiar with a programming language other than SAS. To make CAS more open, CAS has a standardized code base that supports code not only from SAS, but also from Python, Java, Lua, and even R.

This paper provides a complete example of programming in CAS using the following steps:

- Establish a CAS session
- Allocate a CAS library (caslib)
- Read data into CAS
- Manipulate data using the DATA step
- Use PROC CAS to execute various actions
- Analyze the data

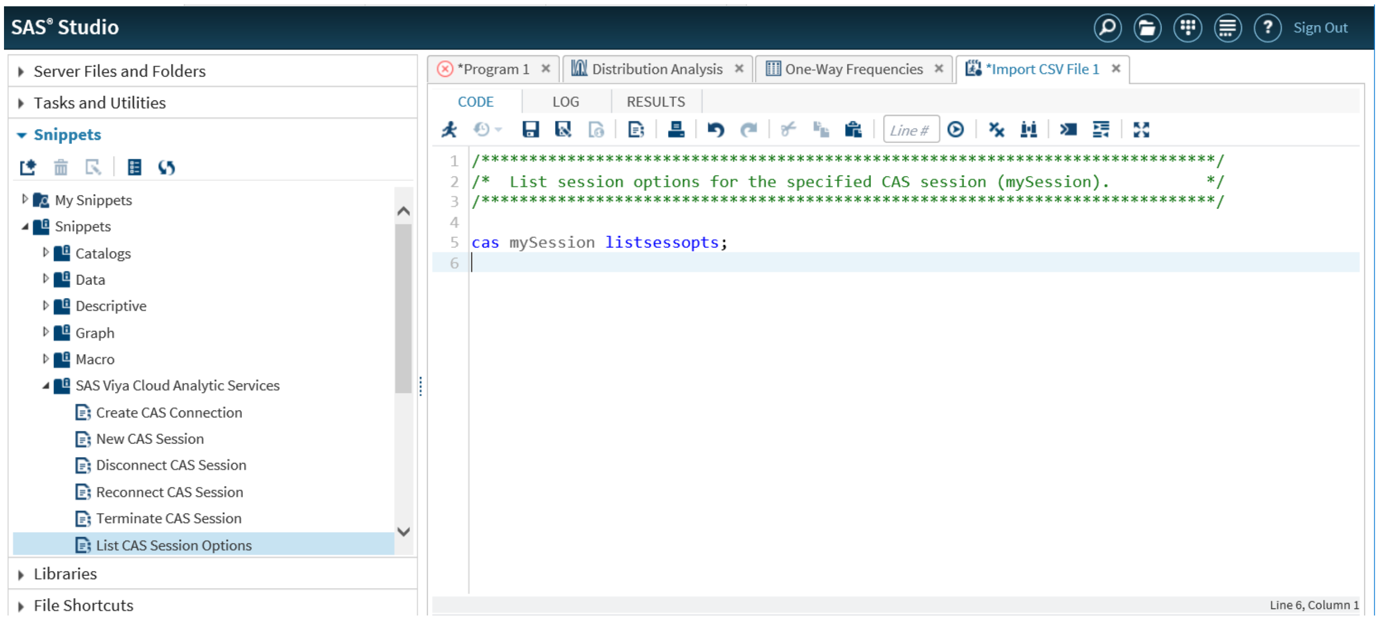
USE SAS® STUDIO

You can use many interfaces to write and submit code to CAS—both delivered by SAS and available elsewhere. For SAS programmers, SAS Studio is the recommended interface because it provides many user-friendly features to enhance your programming environment. SAS Studio is delivered as part of all SAS deployments so it should be available to you.

SAS Studio is a web interface to the SAS system, which means that you can use it on multiple devices. SAS Studio gives you access to all of your SAS resources so that it is easy to use data that you have collected, files that you rely on, and programs that you have written. In addition, these resources are stored on the server so that you can easily access them from whatever device you are using. To make code processing more efficient, SAS Studio uses the SAS server, which might be local or remote (so it can be scalable depending on the size of your data). When the code has finished processing, you view the results in SAS Studio.

SAS Studio contains programming tools that enable you to work more efficiently. You can perform common tasks such as reading CSV files by using the predefined code in one of the many code snippets that are available. SAS Studio also provides tasks that enable you to generate code based on input provided by you.

SAS Studio contains too many benefits of to list in this paper. For an in-depth discussion of SAS Studio, see SAS® Studio, which also contains links to helpful videos. Display 1 below shows the SAS Studio interface and the **Snippets** menu:



```
/*****************************************************************************/
/*  List session options for the specified CAS session (mySession).          */
/*****************************************************************************/

cas mySession listsessopts;
```

Display 1. Code Generated by the List CAS Session Options Snippet

ESTABLISH A CAS SESSION

You must always take that first step when trying something new. The first step that you need to take when you start programming in CAS is to establish a CAS session. The client communicates with the server using a CAS session. When you start a new session, the server starts the session controller, which in turn communicates with the session worker nodes.

CAS sessions manage various aspects of your environment. An important task that the session manages is authenticating your credentials. After you start the session, you do not have to re-enter your credentials to access other server resources.

CAS sessions also increase your efficiency by using the following management strategies:

- resource allocation (The resources that you create in a session are available only within that session.)
- concurrent processing, where several actions run at the same time in different sessions

In addition, if you want to monitor how the session is using your system's resources, you can use the METRICS=TRUE session option. This option provides you with metrics about items such as memory usage. Finally, the session provides fault isolation, which ensures that issues that you encounter do not affect other clients or the server.

The syntax for the CAS statement that starts the session is very straightforward, as you can see here:

```
cas session-name <option(s)>;
```

There are currently 28 options that you can use in the CAS statement. For a complete list of options, see the SAS® 9.4 and SAS® Viya® 3.3 Programming Documentation CAS User's Guide.

This paper, however, shows only the basic CAS statement needed to start a CAS session:

```
cas casauto;
```

This statement starts a CAS session named CASAUTO. Because the statement did not specify any options, the session has default session properties. If the session starts successfully, you see a message similar to the following in the log:

```
NOTE: The session CASAUTO connected successfully to Cloud Analytic Services
############.sas.com using port 5570. The UUID is ##########. The user is xxxxxx
and the active caslib is CASUSER(xxxxxx).
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 4 workers.
```

Note: CASAUTO is a name that I chose for this paper but you can choose your own name.

That's it—you've taken your first step with starting to program in CAS!

ALLOCATE A CAS LIBRARY (CASLIB)

CAS pulls data from data sources and uses caslibs to hold the data in-memory. Caslibs are the holding spaces for your data such as database files, files in a directory, and existing in-memory tables. In a CAS session, you use caslibs in all your interactions with data.

Also, caslibs contain access controls that determine who is authorized to use the information in the caslib. There are three types of caslibs: personal, pre-defined, and manually added.

- *Personal caslibs* are optional, and they are restricted so that only your user credentials can access them. If you want a personal caslib, you need to select the option when you configure the server. This type of caslib has a global scope, which is explained later in this section.
- *Pre-defined caslibs* are global in scope and are defined and maintained an administrator. This type of caslib is normally used as a common data source for programmers who use the server.
- *Manually added caslibs* are added by an administrator and are used for data access that is not normally available to programmers using the server.

Caslibs have either a session scope or a global scope. A *session-scope caslib* is available only in the session that added it. Session-scope caslibs are intended for when you need to only access data and do not need to share the data. Conversely, when you need to share data, you use caslibs with a *global scope*, which are accessible from any active session on the server. Access to the data is determined by the access controls established by the administrator.

Within a CAS session, the default location for server-side data access is the active caslib. If you have a personal caslib when you start a session, then it is your active caslib and is used for all CAS operations. You can switch the active caslib to another caslib by using the SESSOPTS= option in the CAS statement. This example shows how to make CASTestLib the active caslib.

```
cas casauto sessopts=(caslib='CASTestLib');
```

Another way to change the active caslib is to use the CASLIB statement. The following CASLIB statement illustrates how to make this change:

```
caslib global_lib path='/opt/mydata' type=path sessref=casauto global;
```

Another function that the CASLIB statement can perform is to show the settings for one or more caslibs. The LIST option for the CASLIB statement writes out the settings for the specified caslib. The following syntax writes out the settings for the Casuser caslib.

```
caslib casuser list;
```

In the log, you see output similar to the following that shows that CASUSER is the active caslib and that it is a personal caslib:

```
77          caslib casuser list;
 NOTE: Session = CASAUTO Name = CASUSER(xxxxxx)
          Type = PATH
          Description = Personal File System Caslib
          Path = /r/xxxxx.unx.sas.com/vol/vol710/u71/xxxxxx/casuser/
          Definition =
          Subdirs = Yes
          Local = No
          Active = Yes
          Personal = Yes
 NOTE: Action to LIST caslib CASUSER completed for session CASAUTO.
```

Note that in my environment, Casuser is the default active caslib that I use throughout the paper.

The final, but very important, aspect of a caslib that you need to know is that the name of your active caslib is not a libref. To read and write data to and from the caslib using traditional SAS PROCs and the DATA step, you must associate a libref with the caslib using the LIBNAME statement with the CAS engine. The following LIBNAME statement associates the Casuser caslib with the mycas libref.

```
libname mycas cas caslib=casuser;
```

In sample code later in the paper, I use the mycas libref to point to the Casuser caslib.

READ DATA INTO CAS

Now that you have started SAS Studio, established the CAS session, and assigned your active caslib, you can start reading in, manipulating, and analyzing data. The first step is to read in the data. The CAS environment provides great flexibility when it comes to reading in data, offering multiple ways of accomplishing this task. Not only are there several ways to read in the data, but there are also numerous file types that can be read in. The file types that can be read into CAS include SAS data sets, delimited files, Microsoft Excel files, and tables from various databases including Hive, Oracle, and Teradata.

This paper demonstrates three ways to read in data from the SAS client to the CAS server. The first example shows how to import an existing SAS data set using a DATA step. The second example describes how to use the CASUTIL procedure to read in an Excel file. The final example illustrates how to read in a CSV file using PROC CAS. Note that each of these tools can be used to read in multiple data types.

DATA STEP

SAS programmers are most familiar with using a DATA step to read existing SAS data sets, and they can also use this method in CAS. To read the data set into a caslib, you must use a libref that points to an existing caslib in the DATA statement. Here is the LIBNAME statement from earlier in the paper that shows how to create a libref that points to a caslib:

```
/* This statement associates the mycas libref with the Casuser caslib */
libname mycas cas caslib=casuser;
```

After running the LIBNAME statement above, you can use the mycas libref to name the table that is created by the DATA step. Here is the DATA step code needed to import an existing SAS data set into a caslib:

```
data mycas.cars;
   set sashelp.cars;
run;
```

It is that simple. The CARS table now exists in the Casuser caslib.

PROC CASUTIL

Another tool that can load data into CAS is PROC CASUTIL, which has three primary functions:

- Loading data
- Providing table and file information

- Deleting tables and files

This section of the paper discusses the data loading functionality. In the following example, PROC CASUTIL creates the CARS2 table by reading in the cars.xls file (The sashelp.cars data set is included with SAS). If you are familiar with using the IMPORT procedure to read in an Excel file, you might notice that the syntax is somewhat similar:

```
 proc casutil;
    load file='/tmp/cars.xls' /* 1 */
    casout='cars2' outcaslib='casuser' /* 2 */
    importoptions=(filetype='xls' getnames=true); /* 3 */
quit;
```

1. The file= option points to the file to be read in.
2. The casout= option names the output table and the outCaslib= option designates the output caslib.
3. The importOptions= option enables you to specify the options used when importing the file. The fileType= option specifies the import file type, and the GETNAMES statement determines whether the first line of data is used to create the variable names for that file.

PROC CAS

PROC CAS enables you to interact with CAS by executing the CAS language (CASL). CASL interacts with the server by enabling you to run CAS actions, which are requests to perform tasks on the server. A typical action provides multiple input parameters that enable you to control the operation performed by the action. Many actions are available within PROC CAS and that number is continually increasing. PROC CAS provides actions that perform tasks ranging from session administration and table management to complex analytic reporting.

Because there are so many actions available, each action is part of an action set. An action set is simply a collection of actions grouped based on their functionality. Typically, when you specify an action, the syntax is the action set name followed by the action:

<action-set-name.>action-name

You only need to specify the action set name if the action that you are using is contained within multiple action sets. However, new actions are being added regularly to all action sets, so it is recommended that you include the action set name along with the action. For a list of action sets and the actions they contain, see SAS® Viya® 3.2 Programming / Actions and Action Sets by Name and Product.

The following PROC CAS code uses the addCaslib and loadTable actions to read in the cars.csv file.

```
 proc cas;
   session casauto;
   table.addcaslib  /* 1 */
     caslib='csvlib'
     datasource={srctype='path'}
     path='/tmp';
   run;
   table.loadtable /   /* 2 */
     path='cars.csv'
     casout={caslib='casuser',name='cars3'}
     importoptions={filetype='csv',getnames='true'};
quit;
```

1. The addCaslib action adds the Csvlib caslib, which contains the information about the location of the file being read in. The SrcType='path' parameter states that the file being read in is located in the directory listed on the path= parameter.
2. The loadTable action indicates the name of the file being read in and the name of the output table. The path= parameter names the input file. The caslib= parameter specifies the output caslib, and the name= parameter designates the output table. The fileType= parameter specifies

the input file type, and the getNames= parameter tells the loadTable action to use the first row of data for the variable names in the table.

So, now that you have read in your data, the next step is to manipulate it before you ultimately analyze it.

MANIPULATE DATA USING THE DATA STEP

If you are a long time SAS user, you will be happy to know that the reliable DATA step is available in CAS. Like the other steps that you can execute in CAS, the DATA step can run in parallel on a distributed server. When you run the DATA step in CAS, extremely large data sets are divided among all the threads on all the available machines. The DATA step code is also copied to each thread and executes against only the data located on that thread. Running the DATA step on multiple threads across many machines can greatly increase processing speeds.

The following figure from the "DATA Step Processing Modes" section of *SAS® Viya® 3.2 Programming / DATA Step Programming for CAS* illustrates how a DATA step runs in CAS:

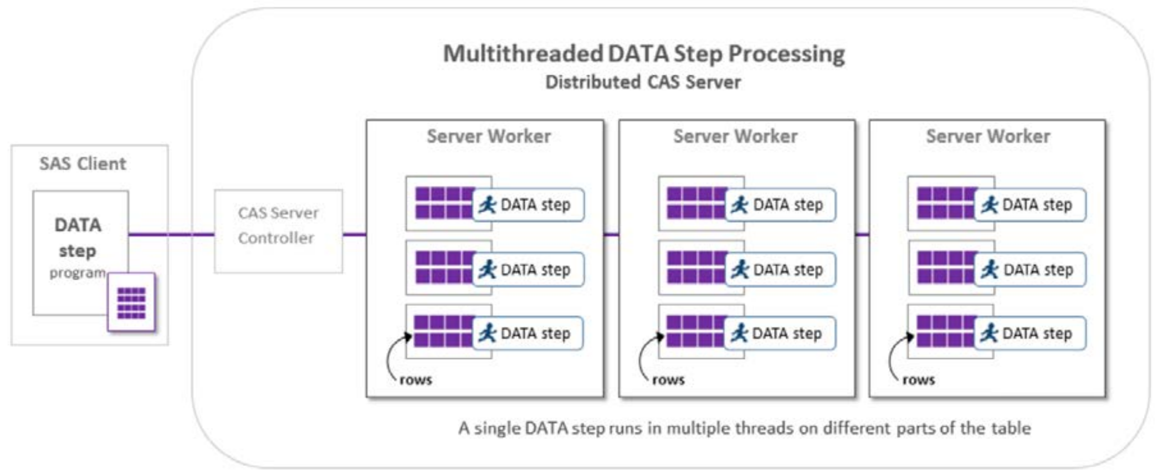


Figure 2. The DATA Step Executing in CAS

Each Server Worker (machine) in Figure 2 contains multiple threads that the data is divided amongst and that the DATA step executes on.

THE DATA STEP IN CAS

When a DATA step runs in CAS, the following note is written to the log:

```
NOTE: Running DATA step in Cloud Analytic Services.
```

This note simply indicates that the DATA step is running in CAS. Whether the DATA step runs in CAS is determined by where your input and output tables are stored. If your tables are stored in a single location (either in SAS or in CAS), then the DATA step runs in that location. The DATA step runs in SAS if your tables are stored in multiple locations.

For example, the following DATA step executes in CAS because both the input and output tables are stored in a caslib:

```
77    data mycas.bytest;
78      set mycas.cars;
79      by make;
80      if first.make then put make=;
81    run;
NOTE: Running DATA step in Cloud Analytic Services.
```

However, in this example from earlier in the paper, the DATA step runs in SAS because the input data set is from a SAS library:

```
data mycas.cars;
  set sashelp.cars;
run;
NOTE: There were 428 observations read from the data set SASHELP.CARS.
```

You can see that the note stating that the DATA step ran in CAS is absent.

The next example uses a much larger data set. This data set contains ATM transaction data that includes the transaction date, amount withdrawn or deposited, status, and the ending account balance. However, this data set does not contain the ATM transaction fee. The DATA step below adds the transaction fee based on the year of the transaction and whether the transaction was successful. The account balance is then adjusted to reflect the fee.

As the log below illustrates, the PUT statement is used to write out the value of _THREADID_. The _THREADID_ variable is an automatic variable created by a DATA step executed in CAS. The value of _THREADID_ is the thread number associated with the thread that the DATA step is running on. The test environment for this paper runs on a total of four threads. So, **4** is the maximum value that would be written out by the PUT statement.

```
data mycas.updated_transaction_history;
  set mycas.transaction_history;
  year=year(transaction_dt);
  fee=0;
  if transaction_status='SUCCESSFUL' then do;
    if year=2013 then fee=1;
    else if year=2014 then fee=2;
    else if year>2014 then fee=3;
  end;
  new_transaction_amt=transaction_amt+fee;
  new_act_balance=act_balance-new_transaction_amt;
  put _threadid_=;
run;
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
_THREADID_=2
_THREADID_=4
_THREADID_=1
_THREADID_=3
NOTE: Duplicate messages output by DATA step:
_THREADID_=2  (occurred 720000 times)
_THREADID_=4  (occurred 719059 times)
_THREADID_=1  (occurred 721000 times)
_THREADID_=3  (occurred 720000 times)
NOTE: There were 2880059 observations read from the table TRANSACTION_HISTORY in
caslib CASUSER(sasdemo).
NOTE: The table updated_transaction_history in caslib CASUSER(sasdemo) has 2880059
observations and 8 variables.
NOTE: DATA statement used (Total process time):
      real time           3.86 seconds
      cpu time            0.00 seconds
```

DATA STEP DIFFERENCES

Be aware that there are some components of the DATA step that either work differently in CAS or are not available at all. For example, BY-group processing has some important differences in a DATA step in CAS versus on the SAS client. CAS distributes a data set based on the values of the first BY variable. Each BY group is directed to a different machine in the distributed network. When the DATA step runs on multiple machines containing many threads, each thread runs independently of other threads. Operating on BY groups in independent threads speeds the processing of BY groups. However, this behavior means that ordering is not guaranteed between BY groups when running on the CAS server.

Below are partial logs from BY-group processing on the SAS client versus in CAS. You can see that the order of MAKE in each of the logs is different. The log from the SAS client shows the expected order whereas the log from the DATA step run in CAS reveals a different order. The order in which groups are processed in the CAS log depends on how the BY groups were distributed across the different machines.

Log from SAS Client Execution

```
83      data bytest;
84        set sashelp.cars;
85        by make;
86        if first.make then put make=;
87      run;

Make=Acura
Make=Audi
Make=BMW
Make=Buick
Make=Cadillac
Make=Chevrolet
Make=Chrysler
Make=Dodge
Make=Ford
```

Log from CAS Execution

```
77    data mycas.bytest;
78      set mycas.cars;
79      by make;
80      if first.make then put make=;
81    run;
NOTE: Running DATA step in
      Cloud Analytic Services.
NOTE: The DATA step will run
      in multiple threads.
Make=Hyundai
Make=Kia
Make=BMW
Make=Cadillac
Make=Audi
Make=Land Rover
Make=Honda
Make=Hummer
Make=MINI
```

Refer to SAS® Viya® 3.2 Programming / DATA Step Programming for CAS for further discussion about the components in the DATA step that either work differently or are not available in CAS.

So, even though CAS provides you with many new tools to use when programming, it is comforting to know that you can still use the reliable DATA step!

USE PROC CAS TO EXECUTE VARIOUS ACTIONS

As mentioned earlier, the PROC CAS enables you to interact with CAS by executing CASL. CASL interacts with the server by enabling you to run CAS actions, which are requests made by the programmer to perform tasks on the server. The vast number of CAS actions (a number that continues to grow) along with the diversity of these actions make PROC CAS a programming powerhouse.

Each action is part of an action set, which is a group of actions that have similar functionality. The action sets range in functionality from configuration actions to text-mining actions to regression actions. For example, the configuration action set contains actions for managing server options such as the getServOpt action, which displays the value of server options. The regression action set contains actions for fitting linear, generalized linear, and logistic models. These actions include the genmod, glm, and logistic actions. These are just a small sample of the actions that are available in PROC CAS. For the complete list, refer to the SAS® Viya® 3.2 Programming / SAS Viya System Programming Guide.

To complement the actions, PROC CAS also contains statements that are familiar to SAS programmers. These statements include the DO, DO UNTIL, DO WHILE, IF-THEN/ELSE, and GOTO statements. For a complete list of the statements available with PROC CAS, refer to SAS® 9.4 and SAS® Viya® 3.2 Programming documentation / CAS Procedure. The code example below demonstrates how you can use IF-THEN logic.

TABLE ACTION SET

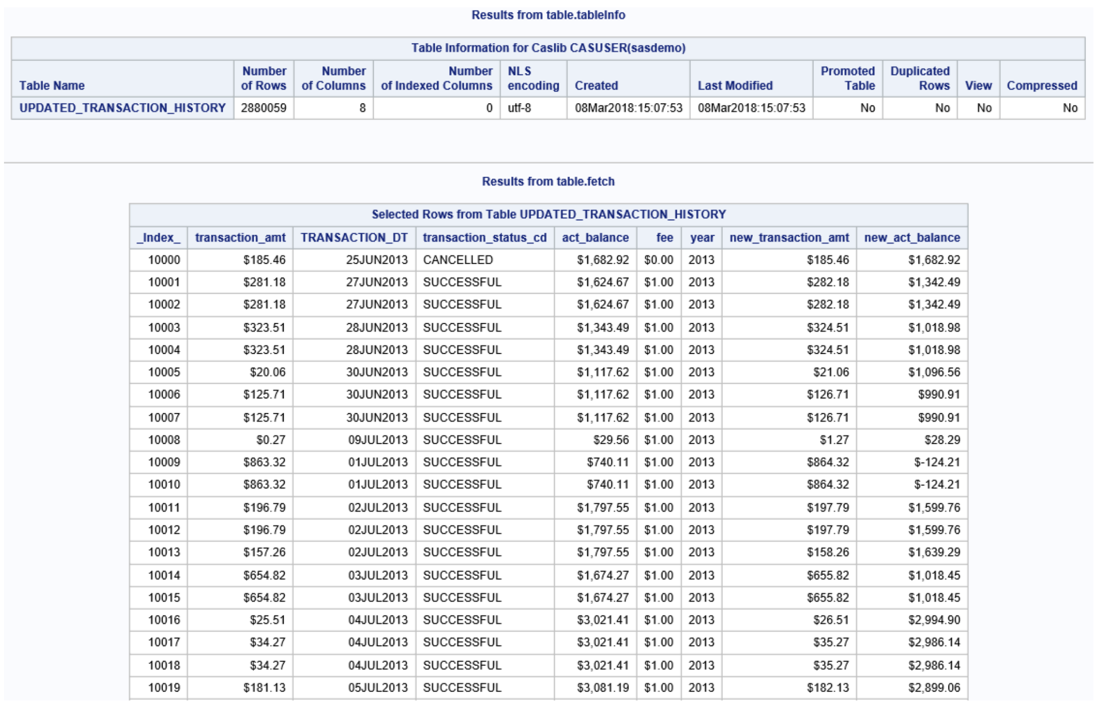
This section describes various actions from the table action set, which contains actions for accessing and managing data. This set includes actions that manage your caslibs and actions that enable you to manage your tables and files. The following examples use various table actions to manage the UPDATED_TRANSACTION_HISTORY table that was created in the DATA step example in the previous section.

The following PROC CAS code demonstrates how you can use CAS actions to work with tables. This example shows how to verify the existence of a table, review the table's attributes, and view the table itself. Because the table in this example is extremely large, I use the to= and from= parameters to subset the data that interests me.

```
proc cas;
  session casauto; /* 1 */
  table.tableexists result=r /  /* 2 */
    caslib='casuser'
    name='updated_transaction_histordsy';
  run;
  if (r.exists) then do; /* 3 */
    table.tableinfo /   /* 4 */
      caslib='casuser'
      name='updated_transaction_history';
  table.fetch /  /* 5 */
    table={caslib='casuser', name='updated_transaction_history'}
    from=10000    /* 6 */
    to=10250;
  end;
quit;
```

1. The SESSION statement specifies the session name that I am working with.
2. The tableExists action determines whether the UPDATED_TRANSACTION_HISTORY table exists. The result of this action is stored in the R.EXISTS variable.
3. The IF condition determines whether the value of R.EXISTS is a nonzero value. A nonzero value indicates that the table exists, and then the DO loop will execute.
4. The tableInfo action provides information about the table's attributes.
5. The fetch action retrieves rows from the UPDATED_TRANSACTION_HISTORY table.
6. The from= parameter indicates the first row to fetch and the to= parameter specifies the last row to fetch.

Output 1 below shows the results:



Results from table.tableInfo

Table Information for Caslib CASUSER(sasdemo)

Table Name	Number of Rows	Number of Columns	Number of Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Duplicated Rows	View	Compressed
UPDATED_TRANSACTION_HISTORY	2880059	8	0	utf-8	08Mar2018:15:07:53	08Mar2018:15:07:53	No	No	No	No

Results from table.fetch

Selected Rows from Table UPDATED_TRANSACTION_HISTORY

Index	transaction_amt	TRANSACTION_DT	transaction_status_cd	act_balance	fee	year	new_transaction_amt	new_act_balance
10000	$185.46	25JUN2013	CANCELLED	$1,682.92	$0.00	2013	$185.46	$1,682.92
10001	$281.18	27JUN2013	SUCCESSFUL	$1,624.67	$1.00	2013	$282.18	$1,342.49
10002	$281.18	27JUN2013	SUCCESSFUL	$1,624.67	$1.00	2013	$282.18	$1,342.49
10003	$323.51	28JUN2013	SUCCESSFUL	$1,343.49	$1.00	2013	$324.51	$1,018.98
10004	$323.51	28JUN2013	SUCCESSFUL	$1,343.49	$1.00	2013	$324.51	$1,018.98
10005	$20.06	30JUN2013	SUCCESSFUL	$1,117.62	$1.00	2013	$21.06	$1,096.56
10006	$125.71	30JUN2013	SUCCESSFUL	$1,117.62	$1.00	2013	$126.71	$990.91
10007	$125.71	30JUN2013	SUCCESSFUL	$1,117.62	$1.00	2013	$126.71	$990.91
10008	$0.27	09JUL2013	SUCCESSFUL	$29.56	$1.00	2013	$1.27	$28.29
10009	$863.32	01JUL2013	SUCCESSFUL	$740.11	$1.00	2013	$864.32	$-124.21
10010	$863.32	01JUL2013	SUCCESSFUL	$740.11	$1.00	2013	$864.32	$-124.21
10011	$196.79	02JUL2013	SUCCESSFUL	$1,797.55	$1.00	2013	$197.79	$1,599.76
10012	$196.79	02JUL2013	SUCCESSFUL	$1,797.55	$1.00	2013	$197.79	$1,599.76
10013	$157.26	02JUL2013	SUCCESSFUL	$1,797.55	$1.00	2013	$158.26	$1,639.29
10014	$654.82	03JUL2013	SUCCESSFUL	$1,674.27	$1.00	2013	$655.82	$1,018.45
10015	$654.82	03JUL2013	SUCCESSFUL	$1,674.27	$1.00	2013	$655.82	$1,018.45
10016	$25.51	04JUL2013	SUCCESSFUL	$3,021.41	$1.00	2013	$26.51	$2,994.90
10017	$34.27	04JUL2013	SUCCESSFUL	$3,021.41	$1.00	2013	$35.27	$2,986.14
10018	$34.27	04JUL2013	SUCCESSFUL	$3,021.41	$1.00	2013	$35.27	$2,986.14
10019	$181.13	05JUL2013	SUCCESSFUL	$3,081.19	$1.00	2013	$182.13	$2,899.06

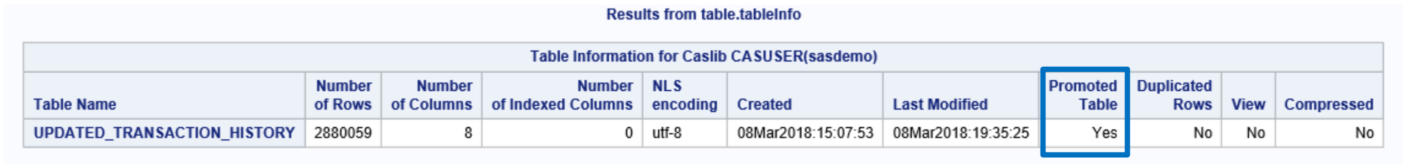
Output 1. Results of Using the Fetch Action

Because the table is ready, I use the following code to promote it. This action makes the table available to me in any CAS session that I have access to and to any user who has the appropriate authorization to access that caslib.

```
proc cas;
  session casauto;
  table.promote /  /* 1 */
    caslib='casuser'
    name='updated_transaction_history'
    targetlib='casuser';   /* 2 */
  run;
  table.tableinfo / /* 3 */
    caslib='casuser'
    name='updated_transaction_history';
quit;
```

1. The promote action makes the table available in a caslib.
2. The targetLib= parameter references the caslib where the promoted table will reside. This example promotes the table and stores it in the Casuser caslib.
3. The tableInfo action confirms that the table has been promoted.

The **Yes** in the **Promoted Table** column in Output 2 verifies that the table was successfully promoted:



Results from table.tableInfo

Table Information for Caslib CASUSER(sasdemo)

Table Name	Number of Rows	Number of Columns	Number of Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Duplicated Rows	View	Compressed
UPDATED_TRANSACTION_HISTORY	2880059	8	0	utf-8	08Mar2018:15:07:53	08Mar2018:19:35:25	Yes	No	No	No

Output 2. Results of Running the Promote Action

This section illustrated what a few CAS actions in a single action set can do. PROC CAS provides hundreds of actions to address multiple programming needs from administration and configuration to data management and analysis. The next section shows examples of how to analyze the data, including an example of using the freq action from the simple action set.

ANALYZE THE DATA

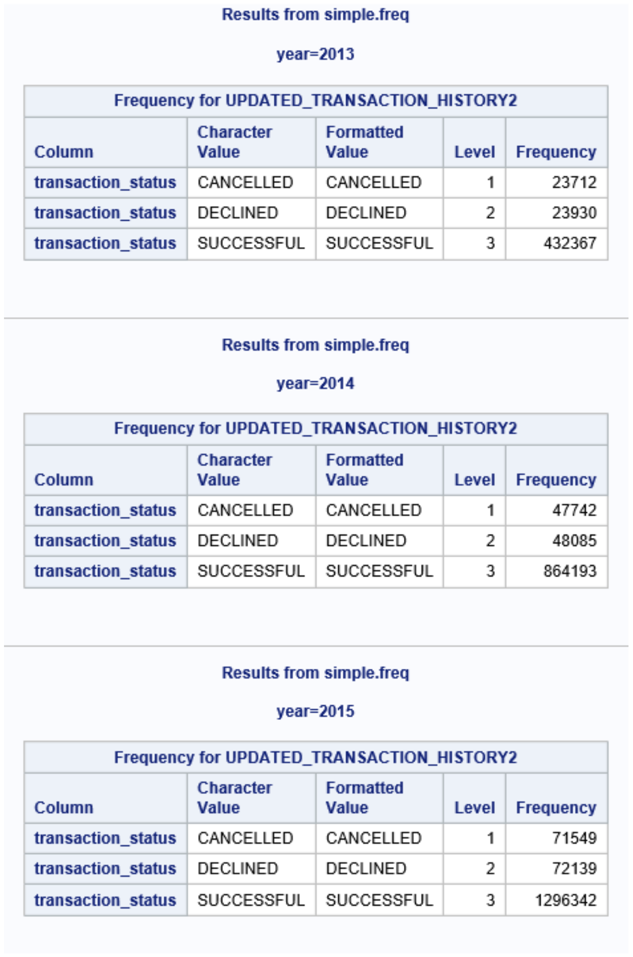
Now that the data has been read in and updated, it is time to perform some basic analysis. This section uses the freq action in PROC CAS to get a frequency count and PROC MDSUMMARY to sum a variable.

The freq action is part of the simple action set, which provides actions for performing basic analytic functions. Some of the actions included in this action set are the correlation, crossTab, freq, and regression actions. The freq action enables you to generate a frequency distribution. The following example uses the freq action to generate a frequency count for TRANSACTION_STATUS by year:

```
proc cas;
  session casauto;
  simple.freq /  /* 1 */
    inputs={'transaction_status'}   /* 2 */
    table={caslib='casuser', name='updated_transaction_history2',
  groupby={name='year'}};  /* 3 */
quit;
```

1. The freq action in the simple action set generates a frequency distribution.
2. The input= parameter contains the name of the variables that the frequency distribution is being generated for.
3. The groupBy= parameter lists the variable to group TRANSACTION_STATUS by.

Output 3 below shows the results:



Results from simple.freq

year=2013

Frequency for UPDATED_TRANSACTION_HISTORY2				
Column	Character Value	Formatted Value	Level	Frequency
transaction_status	CANCELLED	CANCELLED	1	23712
transaction_status	DECLINED	DECLINED	2	23930
transaction_status	SUCCESSFUL	SUCCESSFUL	3	432367

Results from simple.freq

year=2014

Frequency for UPDATED_TRANSACTION_HISTORY2				
Column	Character Value	Formatted Value	Level	Frequency
transaction_status	CANCELLED	CANCELLED	1	47742
transaction_status	DECLINED	DECLINED	2	48085
transaction_status	SUCCESSFUL	SUCCESSFUL	3	864193

Results from simple.freq

year=2015

Frequency for UPDATED_TRANSACTION_HISTORY2				
Column	Character Value	Formatted Value	Level	Frequency
transaction_status	CANCELLED	CANCELLED	1	71549
transaction_status	DECLINED	DECLINED	2	72139
transaction_status	SUCCESSFUL	SUCCESSFUL	3	1296342

Output 3. Results from Using the Freq Action

In addition to the actions available in PROC CAS, you can also use multiple procedures to perform analytics in SAS Viya and CAS. The SAS products that you have licensed and installed determines which procedures are available to you. SAS Viya contains procedures ranging from statistical procedures such as the CARDINALITY and REGSELECT procedures to data mining and machine learning procedures such as the FOREST and TEXTMINE procedures. This section uses PROC MDSUMMARY, which is a statistical procedure. PROC MDSUMMARY enables you to compute descriptive statistics such as the following on your data: MAX, MIN, and SUM. For a complete list of statistics PROC MDSUMMARY performs, see SAS® Viya® 3.2 Programming / CAS Language Reference.

The following example uses PROC MDSUMMARY to return the total number of ATM fees collected per year, per month as well as the total dollar amount collected per year, per month. Be aware that PROC MDSUMMARY does not produce a default report, but only an output table. To address this issue, this example uses the PRINT procedure to view the output table produced by PROC MDSUMMARY.

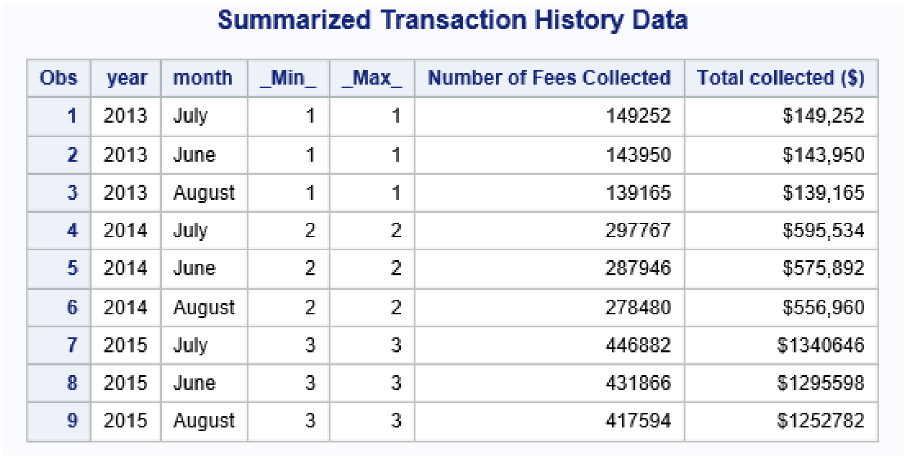
```
data mycas.updated_transaction_history2;
  set mycas.updated_transaction_history;
  month=put(transaction_dt,monname8.);  /* 1 */
run;

proc mdsummary data=mycas.updated_transaction_history2(where=(fee ne 0)); /* 2 */
  var fee;  /* 3 */
  groupby year month / out=mycas.summary_transaction_history; /* 4 */
run;

  proc print data=mycas.summary_transaction_history label; /* 5 */
    title 'Summarized Transaction History Data';
    var year month _min_ _max_ _nobs_ _sum_;
    label _sum_='Total collected ($)' _nobs_='Number of Fees Collected';
    format _sum_ dollar8.;
  run;
```

1. The MONTH function did not exist in the data, so I use the PUT function and the MONNAME format to create it.
2. The WHERE clause in the PROC MDSUMMARY statement reads only fees that are greater than **`0`**.
3. The VAR statement lists the analysis variable.
4. The GROUPBY statement is similar to the BY or CLASS statement. This example groups the data by YEAR and MONTH and uses the out= option to name the output table.
5. Finally, I use PROC PRINT to print the table produced by PROC MDSUMMARY.

Output 4 shows the results:



Summarized Transaction History Data

Obs	year	month	_Min_	_Max_	Number of Fees Collected	Total collected ($)
1	2013	July	1	1	149252	$149,252
2	2013	June	1	1	143950	$143,950
3	2013	August	1	1	139165	$139,165
4	2014	July	2	2	297767	$595,534
5	2014	June	2	2	287946	$575,892
6	2014	August	2	2	278480	$556,960
7	2015	July	3	3	446882	$1340646
8	2015	June	3	3	431866	$1295598
9	2015	August	3	3	417594	$1252782

Output 4. Results from PROC MDSUMMARY

The freq action and PROC MDSUMMARY are just two of the vast number of tools available to analyze your data. SAS Viya and the CAS environment provide all the tools you will need to conquer any of your analytics challenges!

CONCLUSION

Cloud Analytic Services (CAS) is at the heart of SAS Viya. CAS provides a cloud-enabled, distributed, and in-memory environment that enables you to perform super-fast analytics. This paper provided you with an overview of CAS and a basic introduction to how to program in the CAS environment.

This paper provided a complete programming example in CAS. We started with logging on to the CAS server and uploading data to a caslib. We then manipulated the data with the DATA step and checked the data using various table actions in PROC CAS. Finally, we analyzed the data using PROC CAS, the freq action, and PROC MDSUMMARY.

Whether you are a data scientist, a business analyst, an application programmer, or a domain expert, you can start something new by programming in the SAS CAS environment!

REFERENCES

Gass, Mark. 2018. "Cloud Analytic Services Actions: A Holistic View." *Proceedings of the SAS Global Forum 2018 Conference*. Cary, NC: SAS Institute Inc. Available at **support.sas.com/resources/papers/proceedings18/SAS1981-2018.pdf**.

Nelson, Gerry. "First steps in coding with SAS Viya and CAS." Cary, NC: SAS Institute Inc. 2017. Available at **communities.sas.com/t5/SAS-Communities-Library/First-steps-in-coding-with-SAS-Viya-and-CAS/ta-p/347318**.

SAS Institute Inc. 2017. "Caslibs." *SAS® Viya® 3.2 Programming / CAS Fundamentals*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.11&docsetId=casfun&docsetTarget=n1i11h5hggxv65n1m5i4nw9s5cli.htm&locale=en**.

SAS Institute Inc. 2017. "CASLIB Statement." *SAS® Viya® 3.2 Programming / CAS Language Reference*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.11&docsetId=casref&docsetTarget=n0lgusu0v43zxwn1kc5m6cvtnzey.htm&locale=en**.

SAS Institute Inc. 2017. "CAS Procedure: Overview." *SAS® Viya® 3.2 Programming / CAS Procedure*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.11&docsetId=proccas&docsetTarget=p09ouj759e7ysyn1b0ws8f0ho9e5.htm&locale=en**.

SAS Institute Inc. 2017. "CAS Procedure: PROC CAS Statement." *SAS® Viya® 3.2 Programming / CAS Procedure*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.11&docsetId=proccas&docsetTarget=p14hvzwg57dkg5n1fgrugckkgerl.htm&locale=en**.

SAS Institute Inc. 2017. "The DATA Step and CAS." *SAS® Viya® 3.2 Programming / DATA Step Programming for CAS*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.11&docsetId=casdspgm&docsetTarget=n17qunh9lpqavtn1t8mv40r0s5tu.htm&locale=en**.

SAS Institute Inc. 2017. "MDSUMMARY Procedure." *SAS® 9.4 and SAS® Viya® 3.2 Programming Documentation / CAS Language Reference*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.2&docsetId=casref&docsetTarget=n11ya1chrkoyykn17iewtol63xtx.htm&locale=en**.

SAS Institute Inc. 2017. "SAS Cloud Analytic Services 3.3: Fundamentals." *SAS® 9.4 and SAS® Viya® 3.3 Programming Documentation / CAS Fundamentals*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.3&docsetId=casfun&docsetTarget=titlepage.htm&locale=en**.

SAS Institute Inc. 2017. "SAS® Viya™." Cary, NC: SAS Institute Inc. Available at **www.sas.com/content/dam/SAS/en_us/doc/overviewbrochure/sas-viya-108233.pdf.**

SAS Institute Inc. 2017. "Sessions." *SAS® 9.4 and SAS® Viya® 3.3 Programming Documentation / CAS Fundamentals*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.3&docsetId=casfun&docsetTarget=p1wwz16igttqevn1sdej28ekf0f4.htm&locale=en**.

SAS Institute Inc. 2017. "Simple Analytics Action Set: Syntax." *SAS® 9.4 and SAS® Viya® 3.2 Programming Documentation / SAS Visual Analytics Programming Guide*. Cary, NC: SAS Institute Inc.

Available at **go.documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.2&docsetId=casanpg&docsetTarget=cas-simple-freq.htm&locale=en**.

SAS Institute Inc. 2018. "About SAS Studio." *SAS® Studio 3.71 / User's Guide*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=webeditorcdc&cdcVersion=3.71&docsetId=webeditorug&docsetTarget=n1lynjknlzrukln1cxencems3iec.htm&locale=en**.

SAS Institute Inc. 2018. "Examples." *SAS® 9.4 and SAS® Viya® 3.3 Programming Documentation / CAS User's Guide*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.3&docsetId=casref&docsetTarget=p13nxu1c3v5fghn18pyakf52of5e.htm&locale=en#p1fzayvgmpow0mn1ivop3pqakrx5**.

SAS Institute Inc. 2018. "SAS® Studio." Cary, NC: SAS Institute Inc. Available at **support.sas.com/software/products/sas-studio/index.html#s1=1**.

SAS Institute Inc. 2018. "SAS Studio." *SAS® 9.4 and SAS® Viya® 3.3 Programming Documentation / Introduction to SAS Viya Programming*. Cary, NC: SAS Institute Inc. Available at **go.documentation.sas.com/?cdcId=pgmsascdc&cdcVersion=9.4_3.3&docsetId=pgmdiff&docsetTarget=p04bhyhw5bqizfn1c0cj8bmu6v0u.htm&locale=en#n1v1abv75sf2scn18y90ksq02fei**.

Secosky, Jason. 2017. "DATA Step in SAS® Viya™: Essential New Features." *Proceedings of the SAS Global Forum 2017 Conference*. Cary, NC: SAS Institute Inc. Available at: **support.sas.com/resources/papers/proceedings17/SAS0118-2017.pdf**.

ACKNOWLEDGMENTS

The author acknowledges the following people who helped improve this paper: Samantha Biswas, Melissa Corn, Lisa Davenport, Jane Eslinger, Mark Gass, Linda Hamm, Michael McKiernan, Julie McKnight, Jerry Pendergrass, Amy Peters, Denise Poll, Ann Rockett, Rayna Rowell, and Jason Secosky.

CONTACT INFORMATION

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

Kevin Russell
SAS Institute Inc.
SAS Campus Drive
Cary, NC 28513
Email: support@sas.com
Web: support.sas.com