Paper SAS287-2017

Basic Ingredients for Blending the SAS® Client with SAS® Viya™

Denise Poll, SAS Institute Inc., Cary NC

ABSTRACT

Let's walk through an example of communicating from the SAS® client to SAS® Viya™. The demonstration focuses on how to use SAS® language to establish a session, transport and persist data, and receive results. Learn how to establish communication with SAS Viya. Explore topics such as: What is a session? How do I make requests? What does my SAS log tell me? Get a deeper understanding of data location on the client and the server side. Learn about applying existing user formats, how to get listings or reports, and how to query sessions, data and properties.

INTRODUCTION

Existing users of SAS language and procedures are presented with a new tool to assist in data management and analysis. That tool is SAS® Cloud Analytic Services (CAS), which provides a cloud-based run-time environment for data management and analytics with SAS. The goal of this paper is to provide information to allow SAS users to become acquainted with the terms used to describe the CAS server environment and to provide examples of new statements and procedures that interface with the CAS server.

For the examples in this paper, SAS® Studio is the SAS client used to submit example SAS language, statements and procedures. SAS Studio supports **client-side** processing. The CAS server supports **server-side** processing.

SAS CLIENT AND CAS SERVER—LET'S TALK

Communication between a SAS client and a CAS server requires that a connection be made and a server session created. To make a connection some information is required: CAS host name, port and user credentials that can successfully authenticate. The CAS server session processes that are created for your session retain your identity and are then used to determine your authorization to access data.

You can start more than one session (if you want) and other users can start sessions that run simultaneously. A CAS server can run hundreds, and even thousands of sessions simultaneously. Sessions execute independently of each other. Resources such as data or user formats can have global scope, meaning they are sharable with other sessions, or have session scope, meaning they are available to a single session only.

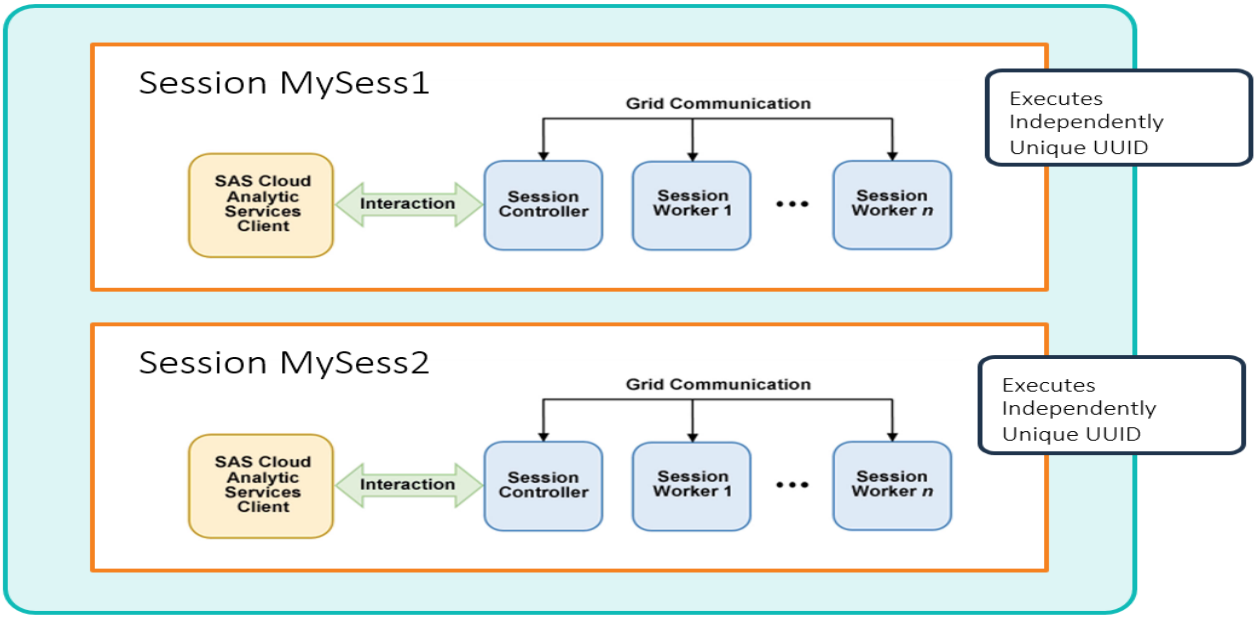

Figure 1. SAS Studio Connects to a CAS Server Session

THE CAS STATEMENT

The CAS statement is a new for SAS® Viya statement that is used to manage a CAS session. The statement supports starting, disconnecting and terminating sessions. In addition, the CAS statement is used to perform management tasks on your session, such as listing or changing session properties and managing session format libraries.

USING THE CAS STATEMENT TO CREATE A SESSION

The CAS statement is used to make a connection, create a session and identify the session with a name. A session reference name allows SAS language procedures and statements to explicitly identify a session.

In this example the CAS statement is used to connect to a CAS server and create a session:

```
cas mysess cashost="viyaserver02" casport=5570 ;
```

```
1?  cas mysess cashost="viyaserver02" casport=5570 ;

NOTE: The session MYSESS connected successfully to Cloud Analytic Services
      viyaserver02 using port 5570. The UUID is
      279c8906-bf2b-0442-9887-11f060b37c30. The user is myid and the active
      caslib is CASUSERHDFS(myid).
NOTE: The SAS option SESSREF was updated with the value MYSESS.
NOTE: The SAS macro _SESSREF_ was updated with the value MYSESS.
NOTE: The session is using 2 workers.
```

Output 1. CAS Statement to Create a Session

The SAS log shows a CAS statement and resulting log notes. Many procedures on SAS Viya run in a CAS session. Other clients such as Lua, Python, and Java also start CAS sessions.

- Session reference names help users keep track of sessions. Example names are MYSESS and CASAUTO. The most recently created session reference is automatically set in the SAS option named SESSREF and macro _SESSREF_. If you do not specify a session reference, the SESSREF system option identifies the session.
- UUIDs are used by the server to manage hundreds of concurrent sessions and maintain uniqueness.
- When a session is started the number of workers associated with the session is reflected in the log.

USING THE CAS STATEMENT TO DISCONNECT OR TERMINATE SESSION

A session can be terminated, using the CAS statement, when it is no longer needed. You can also use the CAS statement to disconnect from your session and reconnect to the session at a later time.

This example shows a CAS statement to disconnect from a session and to terminate a session:

```
cas mysess disconnect;
cas mysess terminate;
```

```
2? cas mysess disconnect;
NOTE: Request to DISCONNECT completed for session MYSESS.

3? cas mysess terminate;

NOTE: Deletion of the session MYSESS was successful.
NOTE: Request to TERMINATE completed for session MYSESS.
```

Output 2. CAS Statement Terminate

In addition to disconnecting from a session with the CAS statement, any sessions that you started are automatically disconnected when you terminate SAS. The sequence is as follows:

1. SAS terminates, any CAS sessions that you started from the SAS session are disconnected.
2. Each disconnected CAS session remains running. When no actions are being processed each session starts a time-out counter. The time-out counter is configurable.
3. When each session time-out expires (and you have not reconnected) the session terminates.

SAS OPTIONS AND THE CAS STATEMENT

SAS options allow administrators to set common values in a client-side configuration file or autoexec file. For example, CASPORT= and CASHOST= identify the CAS server port and host name. When connecting and creating a session if the port is not specified on the CAS statement the SAS option CASPORT= and CASHOST= values (if specified) are used to make the connection.

This example shows the SAS options associated with CAS server usage.

```
proc options group=CAS; run;
```

```
1? proc options group=CAS; run;

Group=CAS
 CASAUTHINFO=      Specifies an authinfo or netrc file that includes
                   authentication information.
 CASHOST=viyaserver02
                   The CAS server name associated with a CAS session.
 CASLIB=           Specify the default CASLIB name.
 CASNWORKERS=ALL   Specify the number of workers to use with a CAS session.
 CASPORT=5570      The port associated with a CAS session.
 CASSESSOPTS=      Identify CAS server session options.
 CASTIMEOUT=60     The CAS session timeout in seconds.
 SESSREF=MYSESS    Identify the name to associate with a generated CAS
                   session.
```

Output 3. OPTIONS Procedure

USING THE CAS STATEMENT TO MANAGE SESSION OPTIONS

Session options are customizations, similar to SAS options, applied to your session when the session is started or after the session is started. Session options are server-side options.

This code lists the session option names and values associated with the session MYSESS:

```
cas mysess listsessopts; run;
```

```
cas mySess LISTSESSOPTS;

NOTE: Name = caslib
      UsageType = Session
      Type = String
      Value = CASUSER(viyauser)
      Default Value =
      Group = Caslib
      Description = specifies the caslib name to set as the active caslib.
NOTE: Name = locale
      UsageType = Session
      Type = String
      Value = en_us
      Default Value = en_US
      Group = Localization
      Description = specifies the locale to use for sorting and formatting.
```

Output 4. Session Properties

The partial SAS log displays the value for the CASLIB and LOCALE session options.

SAS CLIENT AND CAS SERVER—LET'S DO SOME WORK

After a session is available **actions** can be submitted from the client to the server. An action is a task that is performed by the server at the request of the client. Actions execute sequentially within a session. When using SAS statements and procedures with a CAS server session, you are **normally not aware** of the action names and parameters being used. The statements and procedures generate action requests and handle the responses. I'm providing information on actions so you have a basic understanding of work flow between the client and server. Action requests can be used to perform the following tasks:

- Summarize data
- Create and use data
- Create and use user-define formats

CREATE AND SUBMIT AN ACTION REQUEST

The following CAS statement constructs an action **request**, sends the request to the server and receives a **response** from the session.

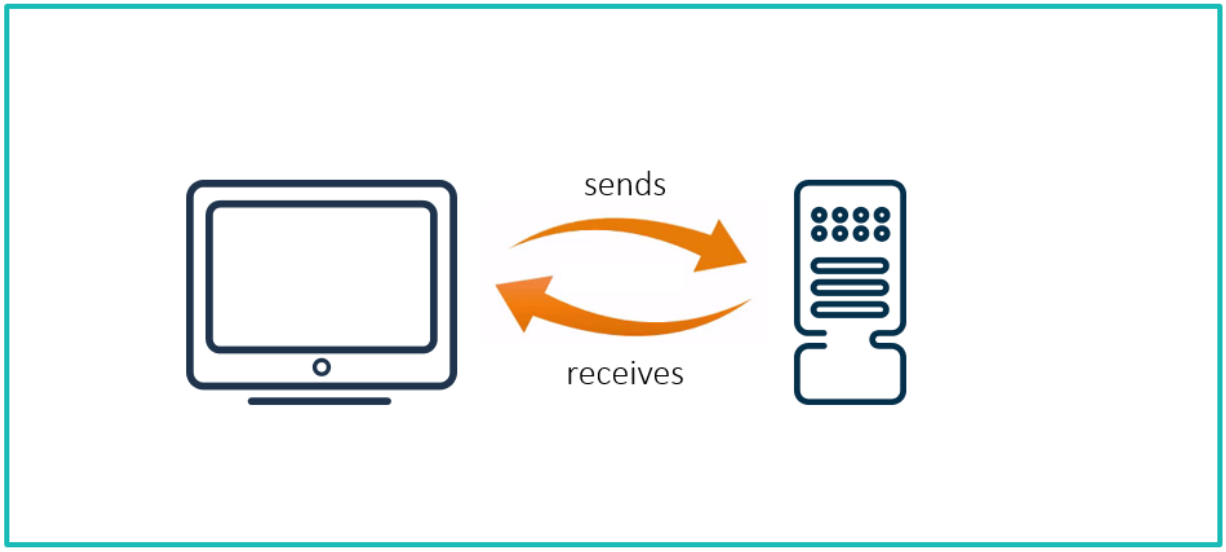

Figure 2. Actions Send Requests and Receive Responses

The action name and parameter syntax is common for all supported clients. This allows requests made from one client to be migrated for use with a different client.

In the following example the **client sends** a request to a session and the client receives a **response** from the session.

This action request sets the **server-side** session option locale. Actions run server-side. When locale is significant to the action, regional settings are applied.

```
cas mysess sessopts=(locale=fr_Fr);
```

```
2? cas mysess sessopts=(locale=fr_Fr);

NOTE: The CAS statement request to update one or more session options for
      session MYSESS completed.
```

Output 5. CAS Statement to Submit an Action

REFLECT ACTION HISTORY TO THE SAS LOG

The CAS statement can be used to list action history. Actions are named and have parameters. The LISTHISTORY option enables you to see the underlying actions that a SAS Viya procedure or statement runs in CAS. Be aware that only some statements trigger actions in CAS.

Here is an example to list action history:

```
cas mysess listhistory;
```

```
3? cas mysess listhistory;

NOTE: 7: action setsessopt / locale='fr_Fr'; /* (SUCCESS) */
```

Output 6. CAS Statement to List History

Notice that the action is named SETSESSOPT and that it has one parameter, LOCALE. Actions are named and have parameters. See the following documents for language elements that trigger actions:

- SAS Cloud Analytic Services: Language Reference
- SAS Visual Statistics procedures
- SAS Visual Data Mining and Machine Learning procedures

WHERE DOES MY DATA LIVE AND HOW DO I IDENTIFY DATA?

All data is available to CAS through caslibs and all operations in CAS that use data are performed using caslibs. A caslib provides access to files in a data source, such as a database or file system directory, and to in-memory **tables**. Access controls are associated with caslibs to manage access to data.

You can think of a caslib as a container. In figure 3, notice the container has two areas where data is referenced: a physical space that includes the **source** data or files, and an **in-memory** space that makes the data available for CAS action processing. The container also holds connection information for the source data and access controls governing who can access the caslib and what they can do with it.

After a file is loaded into the server, it is referred to as a **table**. The in-memory section contains **table data4,** which was loaded from the data source **file data4.sashdat.**

Actions operate on in-memory table data.

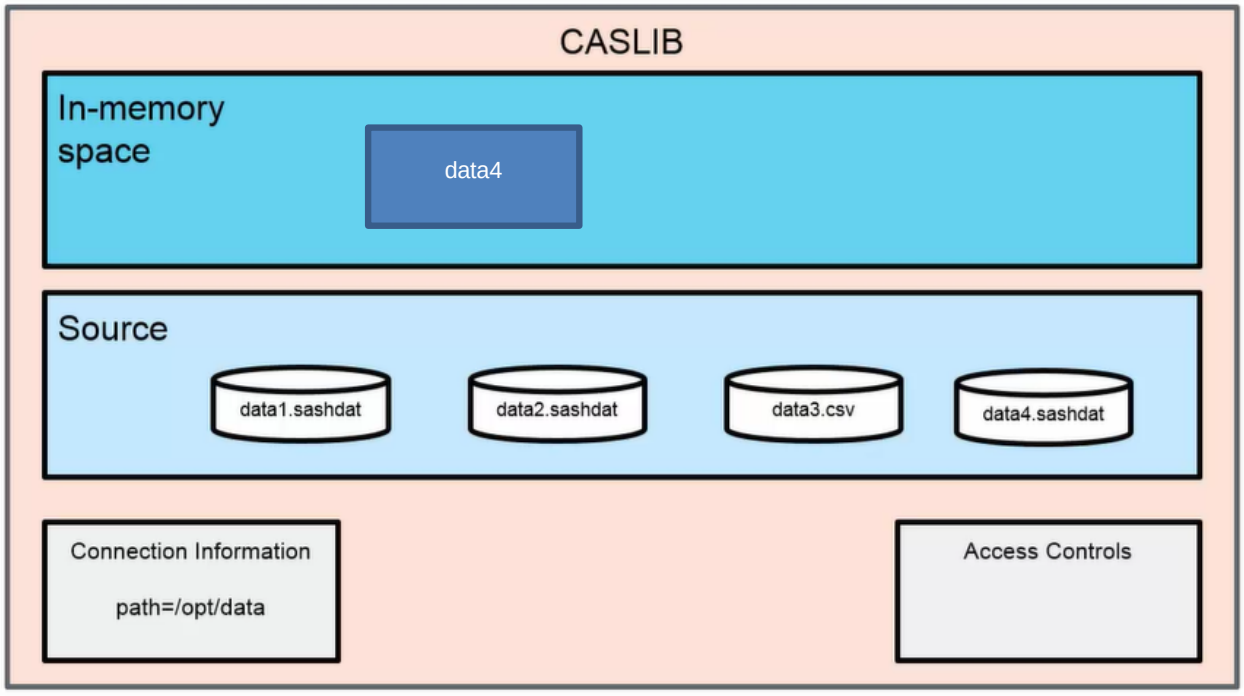

Figure 3. Orientation to CAS Server CASLIB Concepts

CASLIB SCOPE

Session-scoped caslibs can be used in a single session only. Access to the caslib and the tables associated with the caslib is limited to the session that adds the caslib.

Global-scoped caslibs can be used in more than one session. The ability to access data from more than one session is valuable. However, the risk of affecting data access for multiple users is a reason that administrators control access to caslibs. Access controls can be applied to a global caslib to control which users or groups are able to access data from the caslib.

A personal caslib is a special type of global caslib. Because it has global scope, it can be accessed by more than one session. Because it is personal, only the sessions of the caslib's owner can access data from it.

TERMS ASSOCIATED WITH DATA LIFECYCLE

Several terms are associated with the data lifecycle:

Load reads data from a file in a caslib's data source, a libref, or a client-side file and loads it into memory on SAS Cloud Analytic Services.

Save creates a permanent copy of an in-memory table. The in-memory table is saved to the data source that is associated with the caslib.

Drop removes a table from memory on SAS Cloud Analytic Services.

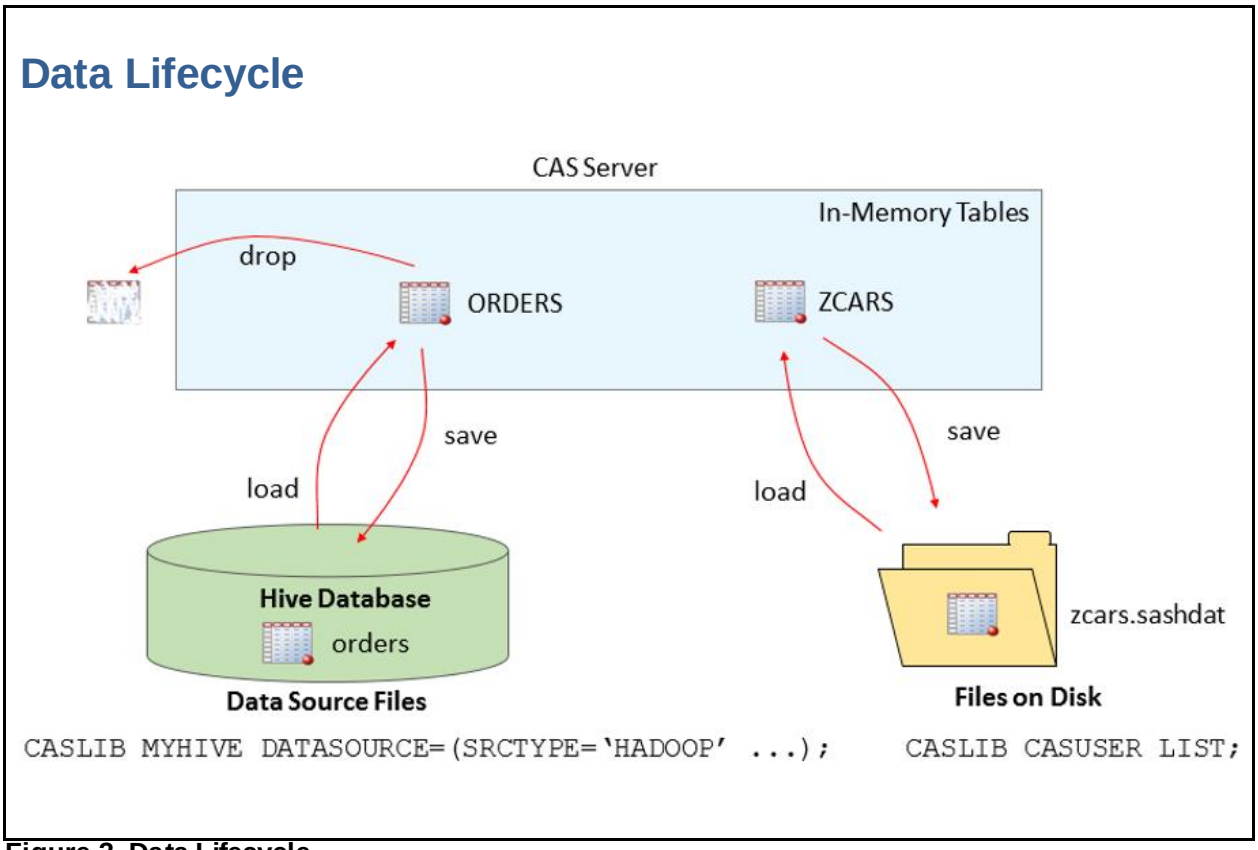

Figure 3. Data Lifecycle

THEY ARE A COUPLE—LIBNAME AND CASLIB STATEMENTS

The new statement CASLIB supports identifying a server-side data source. You can use the CASLIB statement to add a caslib or drop an existing caslib. Hadoop and Oracle are examples of data sources that can be identified using the CASLIB statement.

Here is an example to illustrate the CASLIB statement:

```
caslib mycaslib datasource=(srctype='path') path='/mylib' sessref=mysess;

caslib myhiveclib desc='hive caslib'
  datasource=(srctype='hadoop'
              datatransfermode='parallel'
              hadoopjarpath='/data/myjarpath/lib'
               hadoopconfigdir='data/myjarpath/conf');
```

```
1? caslib mycaslib datasource=(srctype='path') path='/mylib'
sessref=mysess;

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

2? caslib myhiveclib desc='hive caslib'
   datasource=(srctype='hadoop'
               datatransfermode='parallel'
               hadoopjarpath='/data/myjarpath/lib'
                hadoopconfigdir='data/myjarpath/conf');

NOTE: 'MYHIVECLIB' is now the active caslib.
NOTE: Cloud Analytic Services added the caslib 'MYHIVECLIB'.
NOTE: Action to ADD caslib MYHIVECLIB completed for session MYSESS.
```

Output 7. CASLIB Statement

Notice that that MYHIVECASLIB is now the active caslib. When an action parameter does not explicitly identify a caslib, the active caslib is used. By default when a CASLIB statement adds or drops a caslib, the active caslib for the session is updated. The caslib MYCASLIB references a path data source. The caslib MYHIVECLIB references a database. For databases connection information is required.

RELATIONSHIP BETWEEN THE STATEMENTS LIBREF AND CASLIB

SAS language statements and procedures support the use of a libref to identify a data source. The LIBNAME CAS engine supports data transfer between the SAS client and the CAS server. To simplify programming, you can bind the libref to a specific caslib by specifying the CASLIB= option in the LIBNAME statement.

In the DATA step below:

1. The LIBNAME statement identifies the CAS engine and caslib name.
2. DATALINES data is read to the libref MYCAS, which is associated with the CAS engine.
3. The DATA step loads the data to in-memory table CAKE in the CASUSER caslib because the libref MYCAS specifies the caslib associated with MYCAS as CASUSER.
4. The CASUTIL procedure (described in a later section), is used to list in-memory tables.

```
libname mycas cas Caslib=casuser sessref=mysess;

data mycas.cake;
 input lastName $ 1-12 Age 13-14 presentScore 16-17
       tasteScore 19-20 flavor $ 23-32 layers 34;
datalines;
Orlando     27 93 80   Vanilla    1
Ramey       32 84 72   Rum        2
Goldston    46 68 75   Vanilla    1
Roe         38 79 73   Vanilla    2
Larsen      23 77 84   Chocolate  4
Davis       51 86 91   Spice      3
Strickland  19 82 79   Chocolate  1
;

proc casutil sessref=mysess incaslib=casuser;
     list tables ;
```

```
 57    libname mycas cas Caslib=casuser sessref=mysess;
 NOTE: Libref MYCAS was successfully assigned as follows:
       Engine:        CAS
       Physical Name: 710a8ac6-2eb7-f04f-af6d-e7b80a24e2c8
 58
 59    data mycas.cake;
 60     input lastName $ 1-12 Age 13-14 presentScore 16-17
 61           tasteScore 19-20 flavor $ 23-32 layers 34;
 62
 63    datalines;

 NOTE: The data set MYCAS.CAKE has 7 observations and 6 variables.

58         proc casutil sessref=mysess incaslib=casuser;
 NOTE: The UUID '710a8ac6-2eb7-f04f-af6d-e7b80a24e2c8' is connected using
session MYSESS.
 58                    list tables incaslib=casuser;
 NOTE: Cloud Analytic Services processed the combined requests in 0.001000
seconds.
```

The CASUTIL Procedure

Table Information for Caslib CASUSER([redacted])

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
CAKE	7	6	utf-8	20Feb2017:19:46:24	20Feb2017:19:46:24	No	No	No	No

Output 8. SAS Log and PROC CASUTIL List

The following example uses the MDSUMMARY procedure with a server-side table for input:

1. PROC MDSUMMARY sends action requests to the CAS session MYSESS to compute summary statistics for the in-memory table cake.
2. The analysis is performed and the in-memory table FLAG is created.
3. The PRINT procedure brings table FLAG table back from the session and creates a report client-side.

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

proc mdsummary data=mycas.cake;
     var tastescore;
     Groupby flavor age /out=mycas.flag;

proc print data=mycas.flag ;
```

```
57     libname mycas cas caslib=casuser sessref=mysess;
 NOTE: Libref MYCAS was successfully assigned as follows:
       Engine:        CAS
       Physical Name: 710a8ac6-2eb7-f04f-af6d-e7b80a24e2c8
 58
 59     proc mdsummary data=mycas.cake;
 60          Groupby flavor age /out=mycas.flag;

NOTE: The Cloud Analytic Services server processed the request in 0.001
seconds.
NOTE: The data set MYCAS.FLAG has 7 observations and 19 variables.
```

Obs	flavor	flavor_f	Age	Age_f	_Column_	_Min_	_Max_
1	Chocolate	Chocolate	19	19	tasteScore	79	79
2	Chocolate	Chocolate	23	23	tasteScore	84	84
3	Rum	Rum	32	32	tasteScore	72	72

Output 9. PROC MDSUMMARY

SAS LOG NOTES FOR CLIENT-SIDE AND SERVER-SIDE EXECUTION

SAS Studio supports running DATA steps and procedures. Both the SAS Studio client and the CAS server session can run DATA step code. Where a DATA step runs depends on the location of the data and the DSACCEL option. The DSACCEL SAS option enables the DATA step to execute in CAS.

In the following example, the data libraries, WORK and SASHELP are client-side data sources.

This DATA step runs client-side:

```
data work.cars ;
    set sashelp.cars ;
run;
proc print;
```

```
20? data work.cars ;
    set sashelp.cars ;
run;
proc print;
 22?
NOTE: There were 428 observations read from the data set SASHELP.CARS.
NOTE: The data set WORK.CARS has 428 observations and 15 variables.
```

Output 10. PROC PRINT

Notice that WORK.CARS is where the data is created.

In the following example, all the data sources are server-side.

This DATA step runs server-side:

```
libname mycas cas caslib=casuser sessref=mysess;
data mycas.updatecars;
     set mycas.cars ;
     custom='shiny bumpers';
```

```
36? libname mycas CAS CASLIB=CASUSER sessref=mysess;

NOTE: Libref MYCAS was successfully assigned as follows:
      Engine:        CAS
      Physical Name: 63309d85-afe2-7f40-a975-a4865a87bf34

data mycas.updatecars;
     set mycas.cars ;
     custom='shiny bumpers';

NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
NOTE: There were 428 observations read from the table CARS in caslib
      CASUSER(myuserid).
NOTE: The table updatecars in caslib CASUSER(myusrid) has 428 observations
and 16 variables.
```

Output 11. PROC PRINT

Notice the log NOTE indicates that the data step ran server-side. In addition, the notes also indicate that table **updatecars** in caslib CASUSER was created.

DATA AND THE CAS SERVER

Let's focus on the data processing aspects of the CAS server. We know that the CAS server can manage and analyze vast amounts of data. Whenever data is being used choices have to be made regarding where data resides, how much data is needed for analysis, and the best methodology for accessing the data.

TRANSFERRING DATA TO A CAS SERVER SESSION

Let's understand data loading in order to minimize the continuous transfer of data. If you have data that is client accessible and you know that the data will be continuously used with server-side processing, it makes sense to transfer the data and save the data to a caslib.

The following figure depicts transferring data to a CAS server session.

1. The client reads the file and transfers the data serially, in chunks, to the session controller.
2. The session controller receives the data from the client and distributes rows to the worker nodes. The rows are distributed in round-robin fashion to the workers.

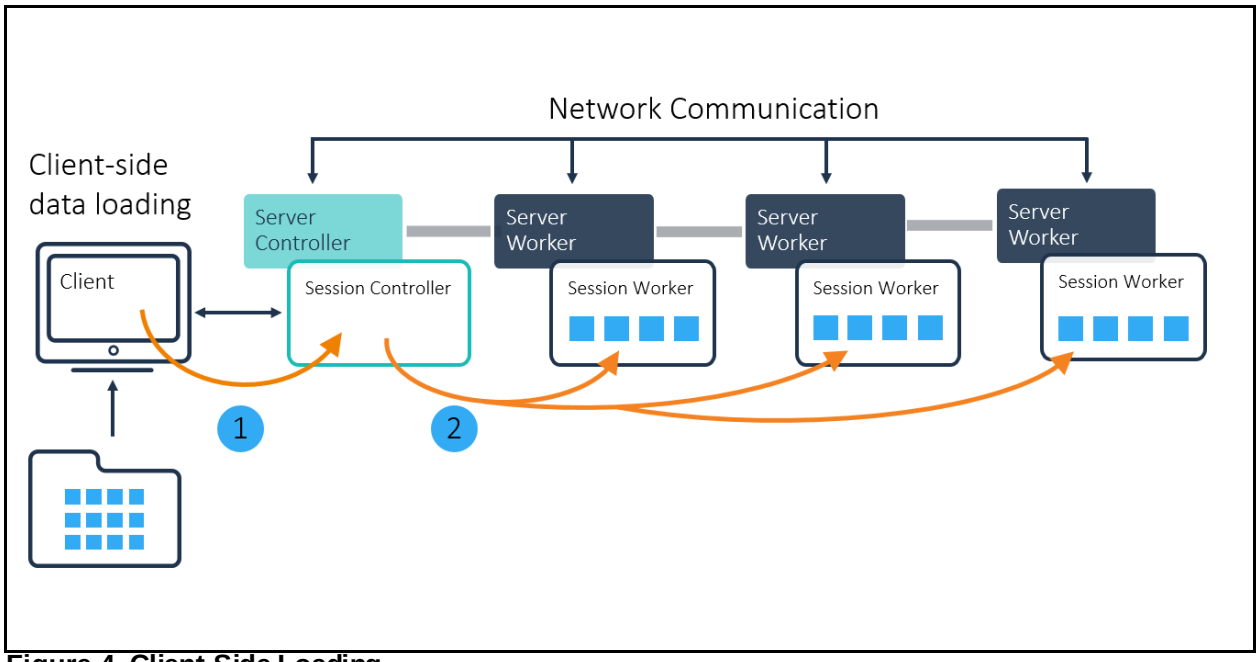

Figure 4. Client-Side Loading

In client-side loading, the first step becomes a bottleneck (network I/O) when the data volume becomes large.

SERIAL LOADING OF DATA DIRECTLY FROM A DATA SOURCE TO THE SERVER

The following figure depicts serial loading of data directly from a data source to the server.

1. The data resides server side.
2. The data is read server side and distributed to workers.

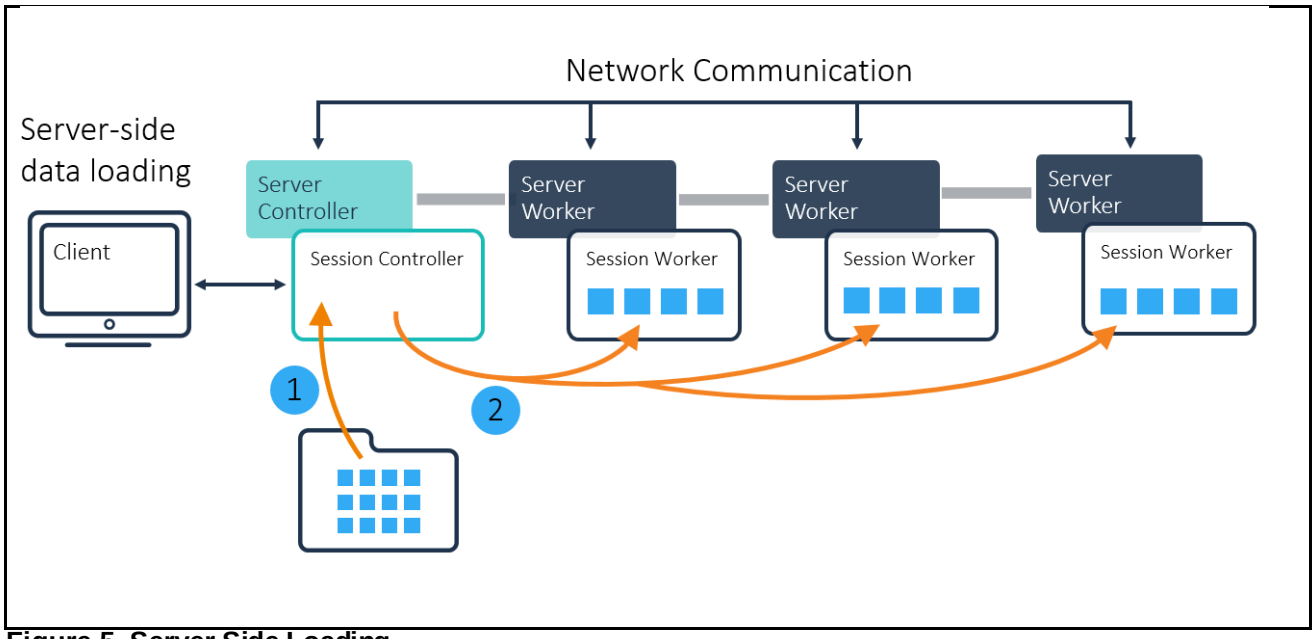

Figure 5. Server-Side Loading

Server-side loading avoids the network bandwidth bottleneck, and when data volumes get even bigger, then disk I/O becomes the next bottleneck.

PARALLEL LOADING OF DATA DIRECTLY FROM A DATA SOURCE TO THE SERVER

The following figure depicts parallel loading of data directly from a data source to the server. This is the fastest way to load data.

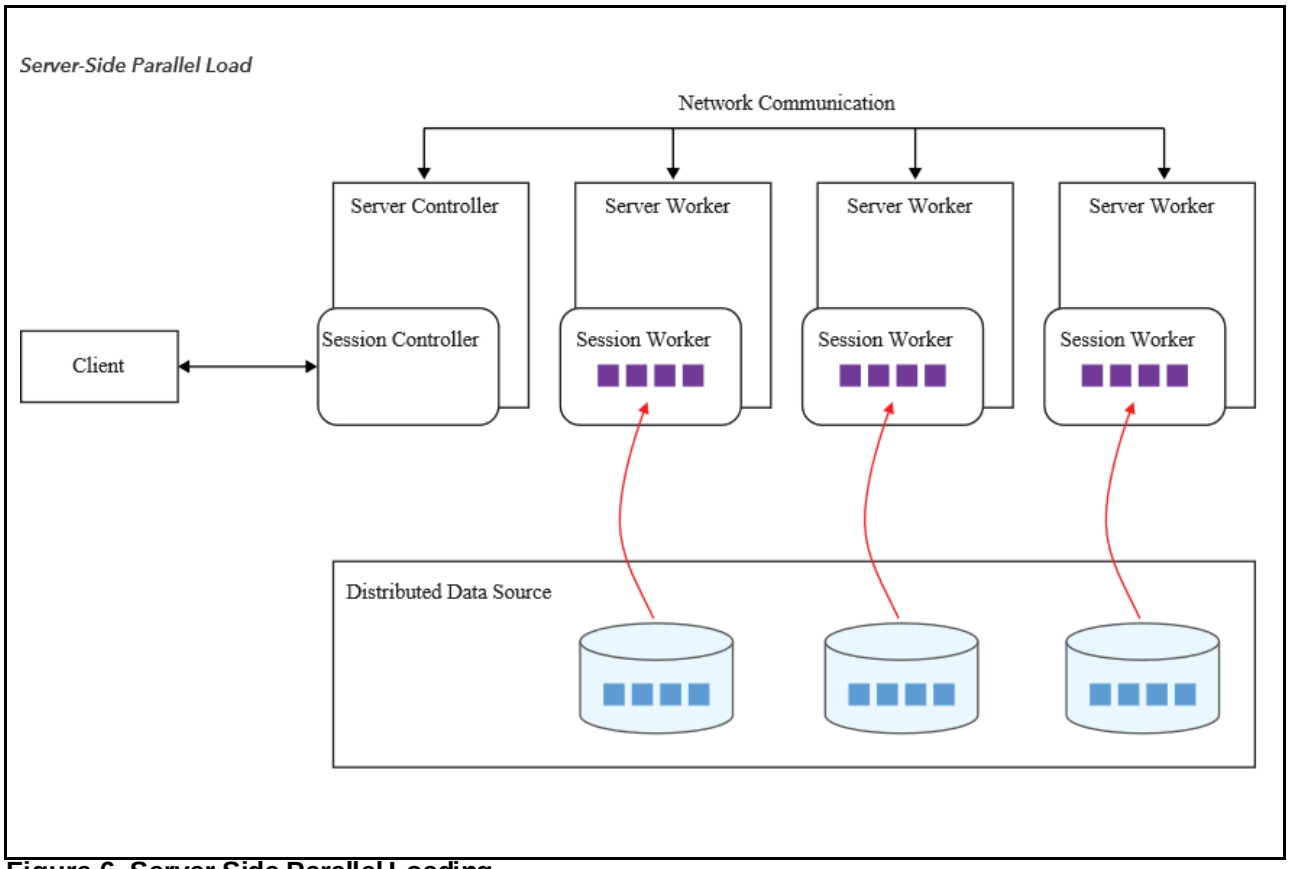

Figure 6. Server-Side Parallel Loading

USING PROC CASUTIL TO MANAGE DATA

PROC CASUTIL works with tables in SAS Cloud Analytic Services, SAS data sets in SAS libraries and external files. The procedure has three functional areas:

- Data transfer
- Table and file information
- Dropping tables and deleting files

CLIENT-SIDE LOAD DATA WITH SERVER-SIDE SAVE DATA

The following example shows how to load client-side data to a session in-memory table.

1. The data set sashelp.cars is loaded to caslib CASUSER as table name mytablemycars.
2. The in-memory tables in caslib CASUSER are listed.
3. Save an in-memory table to a server-side data source.
4. The file statement shows the saved file savedmytablemycars.

```
proc casutil sessref=mysess ;
  load data=sashelp.cars outcaslib=casuser casout=mytablemycars  ;
  list tables;
  save casdata='mytablemycars'
       casout='savedmytablemycars';
  list files incaslib='casuser';
```

```
59  proc casutil sessref=mysess ;
 NOTE: The UUID 'a38fcfda-85c4-7f47-8713-8025677e3554' is connected using
session MYSESS.

60    load data=sashelp.cars outcaslib=casuser casout='mytablemycars'  ;
 NOTE: SASHELP.CARS was successfully added to the "CASUSER" caslib as
"mytablemycars".

61    list tables;
 NOTE: Cloud Analytic Services processed the combined requests in 0.01
seconds.
```

The CASUTIL Procedure

Table Information for Caslib CASUSER([redacted])

Table Name	Label	Number of Rows	Number of Columns	NLS encoding	Created	Last M
MYTABLEMYCARS	2004 Car Data	428	15	utf-8	22Feb2017:22:04:01	22Feb

```
62   save casdata='mytablemycars'
63                casout='savedmytablemycars';
 NOTE: Cloud Analytic Services saved the file savedmytablemycars.sashdat in
caslib CASUSER(myid).
 NOTE: The Cloud Analytic Services server processed the request in 0.01
seconds.

64  list files incaslib='casuser'; run;
NOTE: Cloud Analytic Services processed the combined requests in 0.01
seconds.
```

The CASUTIL Procedure

CAS File Information

Name	Permission	Owner	Group	Encryption Method	File Size
dright.sashdat	-rwxr-xr-x	[redacted]	users	NONE	8.3KB
twofmtlib.sashdat	-rwxr-xr-x	[redacted]	users	NONE	9.5KB
drightaa.sashdat	-rwxr-xr-x	[redacted]	users	NONE	8.3KB
sasdmptbl.sashdat	-rwxr-xr-x	[redacted]	users	NONE	14.0KB
savedmytablemycars.sashdat	-rwxr-xr-x	[redacted]	users	NONE	75.0KB

Output 12. PROC CASUTIL to Load and Save an In-Memory Table

SERVER-SIDE LOAD DATA

Loading data server-side provides direct access to data sources and support for parallel loading provides opportunities for the rapid load and processing of data.

In the following example PROC CASUTIL is used for server-side loading:

1. A Teradata data source file, teracars in caslib TERA, is loaded directly from the data source to in-memory table mytableinmemory in caslib CASUSER.
2. PROC CASUTILis used to provide a list of tables in-memory for caslib CASUSER.
3. PROC CASUTILis used to provide a list of files associated with the data source TERA.
4. Note only files and tables for which your user credentials allow access are listed.

```
proc casutil
  incaslib=tera
  outcaslib=casuser;
  load casdata='teracars'
       casout='mytableinmemory';
 proc casutil ;
   list tables incaslib='casuser';
   list files  incaslib='tera';
```

```
36? proc casutil
      incaslib=tera
      outcaslib='casuser';
      load casdata='teracars'
           casout='mytableinmemory';

NOTE: Performing serial LoadTable action using SAS Data Connector to
Teradata.
NOTE: Cloud Analytic Services made the external data from cars available as
      table MYTABLEINMEMORY in caslib CASUSER(myid).

NOTE: The Cloud Analytic Services server processed the request in 0.001
      seconds.
```

```
proc casutil ;
  list tables incaslib='casuser';
```

The CASUTIL Procedure

Table Information for Caslib CASUSER[redacted]

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Source Caslib	C
MYTABLEINMEMORY	428	16	utf-8	28Feb2017:13:30:18	28Feb2017:13:30:18	No	No	No	cars	TERA	

```
  list files  incaslib='tera';
```

The CASUTIL Procedure

CAS File Information

Name	Catalog	Schema	Type
A	TERA	model	TABLE
ANDY_OUT	TERA	model	TABLE

Output 13. PROC CASUTIL Server-Side Loading

HAVE IT YOUR WAY—USER-DEFINED FORMATS

SAS provides formats for controlling how variables are displayed. You can use the FORMAT procedure in your SAS programs to create or migrate user-defined formats. PROC FORMAT supports the management of user-defined formats in catalogs on the SAS client and the management of user-defined formats in format libraries for a CAS session.

A session format library can be temporary for the duration of your session, or global for the duration of the server, and saved to either a caslib or path.

A table contains columns. Columns have attributes. The format is a column attribute. When applying a format to a variable, the format library to which the format is a member must be in the session format search list. This is similar to how the SAS client uses the FMTSEARCH= SAS option.

CREATING USER-DEFINED FORMATS

The following code creates two formats:

1. The formats FLVRFMT and AGEFMT are created in both a client-side catalog and in a CAS session format library.
2. The CASFMTLIB= option identifies the format library name used with the CAS session.
3. Using the CAS statement with LISTFMTSEARCH lists the format library the session searches when a table variable requires a user-defined format.
4. Using the CAS statement with LISTFORMATS lists the format library members in a specific format library that is known to the session.
5. Using the CAS statement with LISTFORMATRANGES lists range information for the format AGEFMT. In order for the format to be found, the format library must be in the format search list for the CAS session.

```
proc format casfmtlib='fmtlib' sessref=mySess;
  value $flvrfmt
    'Chocolate'='Chocolate'
    'Vanilla'='Vanilla'
    'Rum','Spice'='Other Flavor';
  value agefmt (multilabel)
    15 - 29='below 30 years'
    30 - 50='between 30 and 50'
    51 - high='over 50 years';
run;

cas mysess listfmtsearch; run;

cas mysess listformats fmtlibname=fmtlib members; run;

cas mysess listfmtranges fmtname=agefmt; run;
```

```
36? proc format casfmtlib='fmtlib' sessref=mySess;
     value $flvrfmt
       'Chocolate'='Chocolate'
       'Vanilla'='Vanilla'
       'Rum','Spice'='Other Flavor';
     value agefmt (multilabel)
       15 - 29='below 30 years'
       30 - 50='between 30 and 50'
      51 - high='over 50 years';
   run;

NOTE: Both CAS based formats and catalog-based formats will be written. The
      CAS based formats will be written to the session MYSESS.
NOTE: Format library FMTLIB added. Format search update using parameter
APPEND completed.
NOTE: Format $FLVRFMT has been output.
NOTE: Format AGEFMT has been output.

37? cas mysess listfmtsearch; run;
NOTE: FmtLibName = FMTLIB
         Scope = Session

38? cas mysess listformats fmtlibname=fmtlib members; run;
NOTE: Fmtlib = FMTLIB
         Scope = Session
         Fmtsearch = YES
         Format = $flvrfmt
         Format = agefmt

39? cas mysess listfmtranges fmtname=agefmt; run;
NOTE: Format Name = AGEFMT
         Range = 15-29=below 30 years
NOTE: Format Name =
         Range = 30-50=between 30 and 50
NOTE: Format Name =
         Range = 51-HIGH=over 50 years
```

Output 14. PROC FORMAT Creates Formats Both Client-Side and Server-Side

MIGRATING EXISTING USER-DEFINED FORMATS TO A CAS SESSION

The following example identifies how to move an existing client-side format library to a session.

1. Use the LIBNAME statement to identify the client-side library where the format catalog is located.
2. Use the CATALOG procedure to list the formats names.
3. Use PROC FORMAT to gather the catalog formats into a temporary location.
4. Use PROC FORMAT to move the catalog formats to a CAS session-based format library named SESSFMTLIB.
5. Use the CAS statement LISTFORMATS and FMTLIBNAME=SESSFMTLIB to display the list of formats in the session-based format library.

```
libname myfmts 'mylibwithcat';
proc catalog cat=myfmts.formats ;
  contents stat; run;
proc format library=myfmts cntlout=temp ;
proc format cntlin=temp casfmtlib= 'sessfmtlib'  sessref=mysess;
cas mysess listformats fmtlibname=sessfmtlib;
```

```
36? libname myfmts 'mylibwithcat';

37? proc catalog cat=myfmts.formats ;
    contents stat; run;

Contents of Catalog MYFMTS.FORMATS
  #   Name             Type              Create Date          Modified Date
  ------------------------------------------------------------------------
  1   PRICERANGEONE    FORMAT    02/12/2017 19:13:54    02/12/2017 19:13:54
  2   PRICERANGETWO    FORMAT    02/12/2017 19:14:21    02/12/2017 19:14:21

38? proc format library=myfmts cntlout=temp ;

39? proc format cntlin=temp casfmtlib= 'sessfmtlib'  sessref=mysess;

NOTE: Both CAS based formats and catalog-based formats will be written. The
      CAS based formats will be written to the session MYSESS.
NOTE: Format library SESSFMTLIB added. Format search update using parameter
APPEND completed.

NOTE: Format PRICERANGEONE has been output.
NOTE: Format PRICERANGETWO has been output.



40? cas mysess listformats fmtlibname=sessfmtlib;
```

```
NOTE: Fmtlib = SESSFMTLIB
          Scope = Session
          Fmtsearch = YES
          Format = pricerangeone
          Format = pricerangetwo
```

Output 15. PROC FORMAT Creates Formats Both Client-Side and Server-Side

CONCLUSION

Now that you have basic knowledge of how to connect from a SAS client to a CAS server, how to manage a session and choices for managing data, go forth and begin to use and exploit all the features of the CAS server environment.

ACKNOWLEDGMENTS

Software development is a collaborative effort. I want to acknowledge the many contributing parties from development, testing and technical writing. Specifically, Richard Langston and Adam Budlong for PROC FORMAT and PROC CASUTIL contributions. Tip of the hat to the CAS development and testing teams who support the whole shebang of a CAS server. Lastly, I want to acknowledge the technical writing along with the education teams for Global Enablement and Learning and Catherine Truxillo's education team. These teams bring you better words and pictures to describe what we do.

RECOMMENDED READING

SAS Institute Inc. 2016. *SAS® Cloud Analytic Services 3.1: Fundamentals*. Cary, NC.: SAS Institute Inc. Available at http://go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.1&docsetId=casfun&docsetTarget=titlepage.htm&locale=en.

SAS Institute Inc. 2016. "CASUTIL Procedure." In *SAS® Cloud Analytic Services 3.1: Language Reference*. Cary, NC.: SAS Institute Inc. Available at http://go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.1&docsetId=casref&docsetTarget=titlepage.htm&locale=en.

SAS Institute Inc. 2016. "FORMAT Procedure." In *SAS® Viya™ 3.1 Data Management and Utility Procedures Guide*. Cary, NC.: SAS Institute Inc. Available at http://go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.1&docsetId=proc&docsetTarget=titlepage.htm&locale=en.

SAS Institute Inc. 2016. "CAS Statement" In *SAS® Cloud Analytic Services 3.1: Language Reference*. Cary, NC.: SAS Institute Inc. Available at http://go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.1&docsetId=casref&docsetTarget=titlepage.htm&locale=en.

SAS Institute Inc. 2016. "Working with User-Defined Formats." In *SAS® Cloud Analytic Services 3.1: Accessing and Manipulating Data*. Cary, NC.: SAS Institute Inc. Available at http://go.documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.1&docsetId=casdataam&docsetTarget=p0ag1zldya1l3kn1rp3zlp0zvow5.htm&locale=en.

CONTACT INFORMATION

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

Denise Poll
SAS Institute, Inc.
Denise.Poll@sas.com