Paper SAS1905-2018

What's New in SAS® Data Connectors for SAS® Viya®

Salman Maher and Chris Dehart, SAS Institute Inc., Cary, NC

ABSTRACT

The latest release of SAS® Viya® provides an expanded set of features for accessing your data. This presentation provides an overview of the latest features available with SAS® data connectors. During the presentation, we discuss and provide examples of the following new product capabilities:

- expanded support form more data sources
- pushing SQL queries from SAS® Cloud Analytic Services (CAS) to databases
- saving CAS tables to third-party data sources

The examples are drawn from a wide variety of relational databases including, but not limited to, Microsoft SQL Server and IBM Db2.

INTRODUCTION

SAS Viya introduces two new types of data access components: SAS data connectors and SAS data connect accelerators. These components provide data access capabilities between SAS Cloud Analytic Services on a SAS Viya platform and various data sources. SAS data connectors connect to the data source and load data in a serial mode. SAS data connectors operate on principles like the SAS/ACCESS LIBNAME engines to connect to and retrieve data from a database (or in case of Hadoop, a data platform).

SAS data connect accelerators extend SAS data connectors' functionality by enabling a parallel data load capability between the database clusters and CAS. SAS data connect accelerators use the SAS Embedded Process framework to orchestrate such parallelization

These new data access components expand the existing SAS data access family of products and, in SAS Viya 3.2, are available via two types of product offerings:

- SAS/ACCESS Interface to a particular data source (on SAS Viya) includes the corresponding SAS data connector
- SAS In-Database Technologies for a particular data source (on SAS Viya) includes the corresponding SAS data connect accelerator

SAS DATA CONNECTORS

The SAS data connector is the primary tool for connecting CAS to your databases and data platforms like Hadoop. It uses the database client installed on the CAS controller node to communicate to your database to load data and fetch metadata. The general flow is that the SAS Viya client submits a table action request to the CAS controller. The CAS controller parses the request and engages the SAS data connector to execute the action (Figure 1). For simple requests that do not require data to be loaded into CAS, the SAS data connector will establish a connection to your database and post the request. The database will then return the answer for your request back to the SAS data connector on the CAS controller at which time the results will be forward to the client that initiated the action (1).

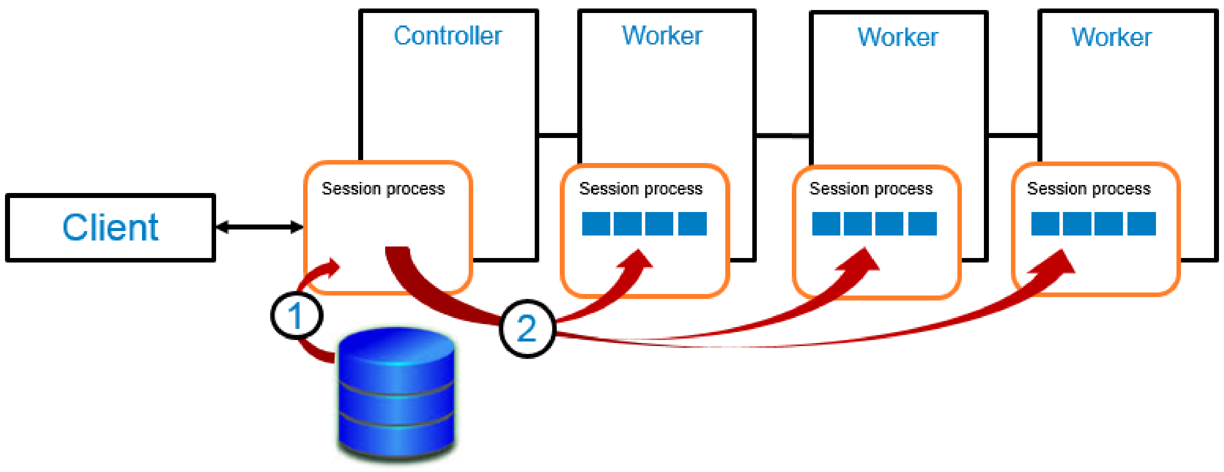


Figure 1 Data Flow for Serial Data Transfer

For requests that require data to be loaded into a CAS table, the work flow is slightly different. In this case, the SAS Viya client submits a table action request to the CAS controller like before and the SAS data connector then initiates a data transfer from your database to the CAS controller (1). Once the data lands on the CAS controller it is redistributed to all the worker nodes in your CAS cluster (2). This load process is considered to be a serial load because the data is sent initial to the controller node before it is distributed to all the work nodes.

SAS DATA CONNECT ACCELERATORS

The SAS data connect accelerators use the SAS Embedded Process to load a table from a data source into a CAS table (Figure 2). The CAS controller node communicates with the SAS Embedded Process over a socket (1) and initiates the data transfer. The SAS Embedded Process then opens connections to the CAS worker nodes (2) and transfers the table data directly to the worker nodes. The worker nodes add the data to the CAS table. Because the data is sent directly to the worker nodes by the SAS Embedded Process, rather than routing all the data through the controller node for distribution to the worker nodes, the work is spread across multiple nodes and the data can be loaded from the data source into CAS in parallel.

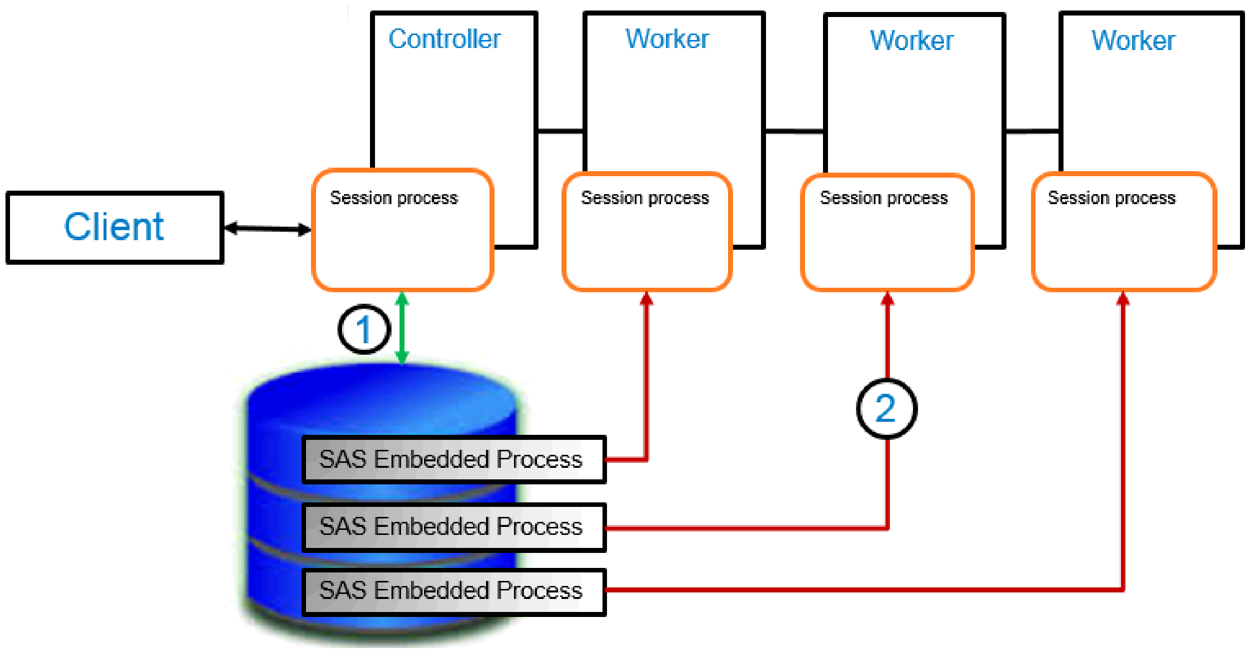


Figure 2 Data Flow for Parallel Data Transfer

The DATATRANSFERMODE= option of the CASLIB statement determines whether the table will be loaded using the SAS data connector or the SAS data connect accelerator. There are three possible settings for the DATATRANSFERMODE= option -- "SERIAL", "PARALLEL", or "AUTO". Specifying DATATRANSFERMODE="SERIAL" on your caslib will cause the table to be loaded using the SAS data connector. Specifying DATATRANSFERMODE="PARALLEL" in your CASLIB statement will cause the table to be loaded using the SAS data connect accelerator. If you specify a value of "AUTO", CAS will attempt to load the table using the SAS data connect accelerator, but if the table load should fail for some reason, for example, if the SAS Embedded Process is not installed or has not been started on your data source cluster, then CAS will retry the table load using the SAS data connector. The default setting for the DATATRANSFERMODE= option is "SERIAL". When you load a table into CAS, you can override the DATATRANSFERMODE= setting that was specified in the CASLIB statement by specifying the DATATRANSFERMODE= option in the DATASOURCE= argument of the table load.

```
/* creates new caslib for parallel execution*/
 caslib myTeraCaslib sessref=mysess
      datasource=(srctype="teradata",
           server="myTeraServer",
           username="user",
           password=*****,
           database="test",
           dataTransferMode="parallel");
```

During a table load, you will see a "Note" level message indicating how the table is being loaded. For parallel load the message is:

```
NOTE: Performing parallel LoadTable action using SAS Data Connector
Accelerator for Teradata.
```

For serial load it is:

```
NOTE: Performing serial LoadTable action using SAS Data Connector to
Teradata.
```

All SAS data connectors and SAS data connect accelerators need to have appropriate database clients installed and configured on the CAS controller node.

Multinode Data Transfer

With the release of SAS Viya 3.3 CAS data connectors for DBMS data sources, you can take advantage of some or all the CAS nodes when reading or writing data. This is known as multinode mode. This mode uses multiple CAS nodes to connect to a data source, which helps speed performance. A CAS controller node coordinates data transfer to and from worker nodes through a configurable number of concurrent connections with the data source. The controller node directs each CAS worker node on how to query the source data to obtain the needed data. Each worker node then connects directly to the data source independently. As a result, multiple data streams can move data simultaneously.

For multinode mode to be enabled:

- The database client must be installed on one or more CAS worker nodes.
- The DATATRANSFERMODE= option must be set to “AUTO” or “SERIAL”.
- The NUMREADNODES= option for the loadTable action and/or the NUMWRITENODES= option for the save action must specify a value other than 1.
- The target table must have a numeric variable.
- The target database must support the MOD function.

The SAS data connector will check the target data source table for a numeric variable. It takes the first numeric variable that it finds and uses that value to divide the table into slices. Division is accomplished by using the MOD function with the number of nodes that you specify. The higher the cardinality of the numeric variable, the easier the data can be divided into slices.

When you specify DATATRANSFERMODE="AUTO", the SAS data connector attempts data transfer in parallel first and then in serial mode if parallel transfer fails. With serial data transfer, if the values for NUMREADNODES= and NUMWRITENODES= are other from 1, then data transfer uses multiple nodes to perform the transfer.

SAS VIYA CLIENTS

With SAS Viya, you are now able to work in a variety of client environments. The following examples will give you the basics on how to create a SAS data connector connection for the supported clients.

SAS Client

The SAS client provides two ways to define a SAS data connector. It is good time to note that a SAS data connector reference in SAS is now as a caslib. Caslibs are the mechanism for accessing data within CAS. They provide a fluid way to hold tables and data source information used by your CAS session.

The two ways to define a caslib are with the CASLIB statement or with the CASLIB procedure (PROC CAS). Regardless of which method you choose to use, you always need to first specify the CAS server host and create a CAS session in your program.

```
options cashost="myCASHost.sas.com";

/* creates cas session */
cas mysess;
```

CASLIB Statement

When using a CASLIB statement, you specify data connector parameters within the DATASOURCE= argument. You will find that this is very similar to a LIBNAME statement with one of the SAS/ACCESS engines. One major difference is that the CASLIB statement does not try to connect to your data source when it is executed. The connection is deferred until the first database request is made by a CAS action.

```
/* creates new caslib */
caslib pgLib sessref=mysess
  datasource=(srctype="postgres",
              server="myPostgresServer",
              database="test",
              username="user",
              password="password"
             );
```

The name that you give to your caslib must be unique within the session that you specified with the SESSREF= parameter in the CASLIB statement. If your caslib name is the same as a global caslib, the global caslib is effectively hidden from use within your session.

CAS Procedure (PROC CAS)

For PROC CAS, you specify your data connector options within the DATASOURCE= argument for an addCasLib action. The caslib name is specified by the LIB= parameter of the addCasLib action.

```
proc cas;
   session mysess;
   action addCaslib lib="pgLib"
   datasource={srctype="postgres",
               server="myPostgresServer",
               database="test",
               username="user",
               password="password"
              };
run;
```

By default, the scope of your caslib is tied to your session. This allows the caslib that is defined with your PROC CAS statement to be used by other procedures that reference as CAS session.

Lua Client

To use the SAS Lua Client Interface for Viya you need to have the SAS Scripting Wrapper for Analytics Transfer (SWAT) package for Lua installed. You can get the latest version from the support.sas.com download page. In addition, you need to be running a 64-bit Lua version 5.2 or 5.3. Once you have SWAT installed, you can import the SWAT package and create a connection to pull data via a SAS data connector.

```
-- Load the CAS objects for use
swat = require 'swat'

-- Start a session in CAS (host, port, userid, password)
s = swat.CAS('myCASHost.sas.com', 5570,'user',’password’)

-- Load the PostgreSQL Data Source object
r=s:loadDatasource{name="postgres"}

-- Define the PostgreSQL caslib
r = s:addCaslib{lib="pglib",
                datasource={srctype="postgres",
                            server="myPostgresServer",
                            database="test",
                            username="user",
                            password="password"
                           }
               }
```

Python Client

To use the SAS Python Client Interface for Viya you need to have the SWAT package for Python installed. You can get the latest version from the support.sas.com download page or GitHub. In addition, you need to be running a 64-bit Python version 2.7, 3.4, or 3.5 on Linux. Once you have SWAT installed, you can import the SWAT package and create a connection to pull data via a SAS data connector.

```
-- Load the CAS objects for use
import swat

-- Start a session in CAS (host, port, userid, password)
conn = swat.CAS('myCASHost.sas.com', 5570,'user',’password’)

-- Load the PostgreSQL Data Source object
conn.loaddatasource(name="postgres")

-- Define the PostgreSQL caslib
conn.addcaslib(datasource={'srctype':'postgres',
                           'server':'myPostgresServer',
                           'database':'test',
                           'username':'user'
                           'password':'password'},
                           lib='pglib')
```

Java Client

To use the SAS Java Client Interface for Viya you need to have the cas-client JAR on your Java classpath. You can get the latest version of the cas-client JAR from the support.sas.com download page. In addition, to the cas-client JAR you need to be using a Java 8 run-time environment with ANTLR Runtime (3.5.2) and Google Protocol Buffers (2.6.1) on your classpath.

There are two ways to define a caslib in Java. One is with the predefined data source options classes and the other is by using a general purpose java.util.HashMap.

Using Predefined Data Source Option Classes

All predefined, data source options classes are located in the com.sas.cas.actions package. Names start with “Ds” and are followed by the source type name. For example, the data source class for PostgreSQL is com.sas.cas.actions.Dspostgres. Note the data source type names are in all lowercase letters. Here is an example.

```
// Instantiate a new client. Set the host, port, username, and password
CASClientInterface client = new CASClient("myCASHost.sas.com",5570,
   "user","password");

// Setup the data source options for PostgreSQL.
Dspostgres dsPostgreSQL = new Dspostgres();
dsPostgreSQL.setServer("myPostgresServer");
dsPostgreSQL.setDatabase("test");
dsPostgreSQL.setUser("user");
dsPostgreSQL.setPassword("password");

// Create the casLib reference to the PostgreSQL data source
AddCaslibOptions addCasLibOptions = new AddCaslibOptions();
addCasLibOptions.setName("pgLib");
addCasLibOptions.setParameter(AddCaslibOptions.KEY_DATASOURCE,
   dsPostgreSQL);

// Add the PostgreSQL casLib to the CAS Server.
CASActionResults<CASValue> results = client.invoke(addCasLibOptions);
```

Using General Purpose Option Classes

If you do not want to use one of the predefined data source option classes or if you cannot find one for a data source that you know is supported by CAS, then you can define your data source using a java.util.HashMap class. In this case, the key values for the hash map are the attributes that need to be set for your data source. Note that the `srctype` value must be set when using a hash map.

```
 // Instantiate a new client. Set the host, port, user name, and
    password
 CASClientInterface client = new CASClient("myCASHost.sas.com",5570,
   "user","password");

// Set up the datasource options
Map<String, Object> dsPostgreSQL = new HashMap<String, Object>();
dsPostgreSQL.put("srctype", "postgres");
dsPostgreSQL.put("database", "test");
dsPostgreSQL.put("server", "myPostgresServer");
dsPostgreSQL.put("password", "password");
dsPostgreSQL.put("username", "user");

// Create the casLib reference to the PostgreSQL data source
CASActionOptions addCasLibOptions = new
   CASActionOptions(null,"addCasLib");
addCasLibOptions.setParameter("lib","pgLib" );
addCasLibOptions.setParameter("datasource", dsPostgreSQL);

// Add the PostgreSQL casLib to the CAS Server
CASActionResults<CASValue> results = client.invoke(addCasLibOptions);
```

WORKING WITH CAS ACTIONS

A CAS action is a task that is performed by the CAS server at your request. The server parses the arguments of the request and invokes the action function. Actions that can be used with SAS data connectors include loadTable, columnInfo, fileInfo, save, and deleteSource,

LOADTABLE ACTION

The loadTable action directs the server on your behalf to load a table from a specified caslib data source into the CAS server. The action at a minimum takes a CASLIB= parameter that describes the origin of your table to load, a PATH= parameter that is the table name of your DBMS table to load, and a CASOUT= parameter that specifies the location for the output table. For example, if you have a table in your PostgreSQL DBMS named "cars" that you want to load into CAS with the name “mycars”, you could use the following loadTable statement:

```
proc cas;
   session mysess;
   action loadTable / caslib="pgLib" path="cars" casout="mycars";
quit;
```

To enable multinode data transfer, you need to set the NUMREADNODES= option to a value other than 1. If you set the value to 0, then all nodes containing a database client will participate in the loadTable action. For example, to distribute the above loadTable across three CAS worker nodes set the NUMREADNODES= option to 3.

```
proc cas;
    session mysess;
    action loadTable / caslib="pgLib" path="cars"
                       casout="mycars" options={numReadNodes=3};
run;
```

Subset Your Columns

With the loadTable action VARS= parameter, you can specify which columns are returned from your loadTable action. This allows you to bring back only those columns that are needed for your analysis in CAS. For example, if you had a table in your DBMS named "cars" that contained the columns "make", "model", "year", "color", and "units", but you were only interested in bringing in the "color" and "units" columns to analyze in CAS, you could use the following loadTable statement to subset on those columns:

```
proc cas;
   session mysess;
   action loadTable / caslib="pgLib" path="cars"
                      vars={"color", "units"} casout="mycars";
run;
```

Filtering Your Data

The loadTable WHERE= parameter allows you to filter data before it is loaded into a CAS table. The filter happens either in the database or on the CAS server as the data is read. If the target database is able to prepare the specified WHERE= parameter, then the filter will occur in the database and only filtered data will be transferred to the CAS server. If the prepare call fails, then the filter of the data is done on the CAS server. For example, given the same "cars" table mentioned above, you could use this code to subset on the "year" column just to read in all the cars built in 2016:

```
proc cas;
   session mysess;
   action loadTable / caslib="pgLib"
                      path="cars"
                      where="year = 2016"
                      casout="cars2016";
run;
```

With SAS Viya 3.3, there is not any indication within the log if the WHERE clause was passed down to the database. To see if the WHERE clause is pushed down you will need to look at the trace log for the CAS library. In the trace log you will see entries like these:

```
ENTER TKTSPrepare
[TKTS_PARSE_TREE] DEFAULT TEXTUALIZATION: "select * from "test"."cars" where
( ("test"."cars"."year"=2016) ) "
DEFAULT: select * from "test"."cars" where ( ("test"."cars"."year"=2016) )
DRIVER SQL: " select * from "test"."cars" where ( ("test"."cars"."year"=2016)
)  "
EXIT TKTSPrepare with return code 0 (TKTS_SUCCESS)
```

The TKTS_SUCCESS return code on the line EXIT TKTSPrepare indicates that the database prepare was successful and the WHERE clause can be pushed to the database. See section “SAS DATA CONNECTOR TRACING” on how to setup tracing.

FEDSQL EXECDIRECT ACTION

With SAS Viya 3.3, the fedSql.execDirect action was updated to allow submitting queries to DBMS caslib data sources. This allows for implicit pass-through (IP) to be enabled with both PROC FEDSQL and the fedSql.execDirect action. IP is the process of translating a FedSQL query into equivalent DBMS-specific SQL so that it can be executed completely on the DBMS. This improves query response time and helps limit the amount of data transferred to the CAS server. When a FedSQL query is submitted to a single data source, an attempt is made to implicitly pass the full query down to the data source for processing. If the targeted data source is not able to prepare the query, then IP will fail and the request is processed locally on the CAS server.

To be eligible for IP:

- All the tables in the FedSQL request must exist in the same caslib.
- All the tables in the FedSQL request must reside on the data source. They cannot already have been loaded into the CAS session.
- The target data source must be able to prepare the SQL request.

To submit your FedSQL request to a caslib, you must fully qualify the table using your caslib in the FROM clause. For example, the following FedSQL query will return all 2016 Fords with 6 cylinders from a “cars” table stored in a PostgreSQL database:

```
proc cas;
   session mysess;
   action fedsql.execdirect
                query="select *
                       from ""pgLib"".""cars""
                       where make='Ford' and Cylinders=6";
run;
```

As you can see, the table is referenced in the FROM clause as caslib.tablename. In this case “pgLib”.”cars”. Note the two sets of double quotation marks around the caslib name and the table name. This is required when using the PostgreSQL caslib as table names and column names are case sensitive. If the FedSQL query is successfully pushed down to the database, a success note is written to the log. The log for the code above is:

```
   73          proc cas;
   74              session mysess;
   75              action fedsql.execdirect
   76                     query="select * from ""pgLib"".""cars"" where
make='Ford' and Cylinders=6";
   77          run;
   NOTE: Active Session now mysess.
   NOTE: Added action set 'fedsql'.
   NOTE: The SQL statement was fully offloaded to the underlying data source
via full pass-through

   78          quit;

   NOTE: PROCEDURE CAS used (Total process time):
         real time           0.74 seconds
         cpu time            0.10 seconds
```

In addition to submitting FedSQL queries with the fedSql.execDirect CAS action, you can also do so with PROC FEDSQL. This is accomplished by setting the SESSREF= option to your CAS session and fully qualifying the table using your caslib.

```
    proc fedsql sessref=mysess;
        create table public.fords as select * from "pgLib"."cars" where 
make='Ford' and Cylinders=6;
    run;
```

The above example shows how to save the results of the FedSQL query into a new CAS table called "fords" under the public caslib.

DELETESOURCE ACTION

With SAS Viya 3.3, support for the deleteSource action was added to the SAS data connectors. By calling this action on your caslib, you can drop a table from your target database. In this example, we are dropping the table "cars2016out" from the PostgreSQL caslib "pgLib".

```
    proc cas;
       session mysess;
       action deleteSource / caslib="pgLib" source=”cars2016Out”;
    run;
```

SAVE TABLE ACTION

With SAS Viya 3.3, support for the save table action was added to the SAS data connectors. With this action you can now write in-memory CAS tables back to your DMBS. At a minimum, the action takes a caslib where the table will be written to, the name of the table to be created, and a TABLE= parameter that describes the source table to write out. For example, if you wanted to write back the "mycars" table loaded into CAS from the above loadTable action to your PostgreSQL DMBS as "myCarsFromCAS", you could use the following save action.

```
    proc cas;
      session mysess;
      action save / caslib="pgLib" name="myCarsFromCAS"
                    table={caslib="pgLib" name="mycars"};
   run;
```

The execution of the save table action will end up resulting in a CREATE TABLE statement being executed against your DBMS. This action will always create a new table. If a table already exists in your DBMS with the same name, the action will fail unless the REPLACE=TRUE option is specified.

You can use the WHERE= and VARS= options under the TABLE parameter to filter data and columns without having to create a temporary table. For example, the following action will subset the "cars" table to only makes from 2016 and create a new table in PostgreSQL named "my2016Cars" with only one column named "make".

```
      proc cas;
        session mysess;
         action save / caslib="pgLib" name="my2016Cars"
             table={caslib="pgLib" name="mycars"
                    vars={“make”}
                    where=”year = 2016”};
      run;
```

To enable multinode data transfer, you need to set the NUMWRITENODES= option to a value other than 1. If you set the value to 0, then all nodes containing a database client will participate in the save action. For example, the following save action will divide the insert across three CAS worker nodes.

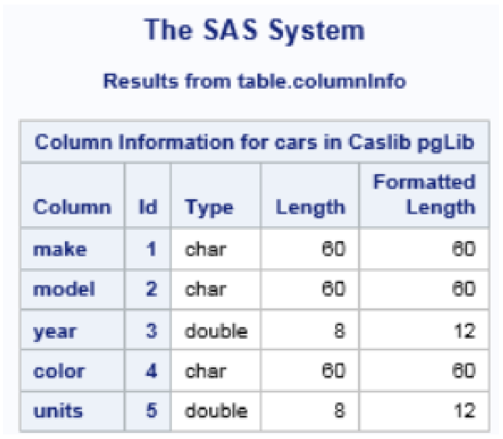
```
proc cas;
  session mysess;
  action save / caslib="pgLib" name="my2016Cars"
         table={caslib="pgLib" name="mycars"
                vars={"make"}
                where="year = 2016"};
         options={numWriteNodes=3};
run;
```

COLUMNINFO ACTION

The columnInfo action can be used to obtain metadata information about the columns in your DBMS tables. The action takes a TABLE= parameter that describes the caslib and the table name of your DBMS table. The following example gets the column information for the “cars” table:

```
 proc cas;
   session mysess;
   action columnInfo / table={caslib="pgLib" name="cars"};
run;
```

Here is the output from this example call to columnInfo.



The SAS System

Results from table.columnInfo

Column Information for cars in Caslib pgLib				
Column	Id	Type	Length	Formatted Length
make	1	char	60	60
model	2	char	60	60
year	3	double	8	12
color	4	char	60	60
units	5	double	8	12

Output 1 Results from table.columnInfo

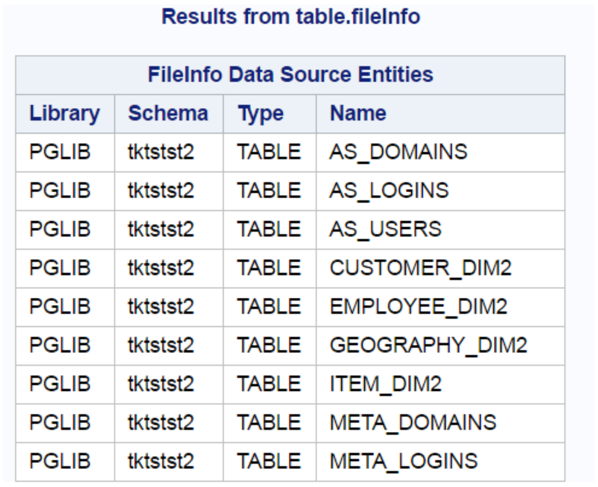
This is a good time to talk about scoping with the columnInfo action. If a DBMS table has not been loaded into CAS and you call columnInfo with the DBMS table name, the column metadata that you receive is the DBMS view of the table. However, if you call a loadTable action and load that DBMS table **into the same caslib** and then call columnInfo, you will receive the column metadata from the loaded CAS table, **NOT** the DBMS table. For this reason, a best practice is to load your DBMS tables into a different caslib from your DBMS caslib by specifying a user caslib in the CASOUT= parameter of the loadTable action.

FILEINFO ACTION

The fileInfo action provides a list of tables and views that are accessible with a specified caslib data source. The action takes a CASLIB= parameter that describes the caslib you want to get the list of tables and views from. For example, this is the PROC CAS call to get the list of tables from the pgLib caslib defined above:

```
proc cas;
  session mysess;
  action fileInfo / caslib="pgLib";
quit;
```

Here is the output from this example call to fileInfo.



Results from table.fileInfo

FileInfo Data Source Entities			
Library	Schema	Type	Name
PGLIB	tktstst2	TABLE	AS_DOMAINS
PGLIB	tktstst2	TABLE	AS_LOGINS
PGLIB	tktstst2	TABLE	AS_USERS
PGLIB	tktstst2	TABLE	CUSTOMER_DIM2
PGLIB	tktstst2	TABLE	EMPLOYEE_DIM2
PGLIB	tktstst2	TABLE	GEOGRAPHY_DIM2
PGLIB	tktstst2	TABLE	ITEM_DIM2
PGLIB	tktstst2	TABLE	META_DOMAINS
PGLIB	tktstst2	TABLE	META_LOGINS

Output 2 Results from table.fileInfo

Search Patterns in the fileInfo Action

You can limit the results of the fileInfo action by using wildcard characters to filter by filename. The fileInfo path argument accepts a search pattern as part of the path. The search pattern is used when the fileInfo WILDINGNORE= parameter is set to FALSE. Using a pattern limits the results returned to the files that match the pattern. By default, search patterns are enabled.

The search pattern characters are:

- Percent sign (%), which represents any sequence of zero or more characters
- Underscore (_), which represents any single character
- Backslash (\), which acts as an escape character, used to include underscores, percent signs, and the escape character as literals

The following table illustrates different search patterns for the string ‘cwd’.

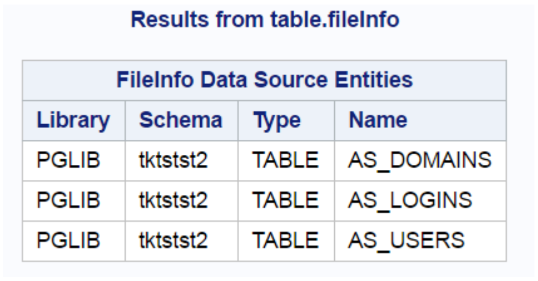
Search Pattern	Description
cwd%	Matches any table name beginning with "cwd"
%cwd%	Matches any table name containing “cwd”
cwd_	Matches any 4-character table name that begins with "cwd"
cwd_%	Matches any table beginning with “cwd_”
‘%cwd’	Use single quotation marks prevent SAS from interpreting %cwd as a SAS macro name

Table 1 Search Patterns

Here is the PROC CAS code for the pgLib example above expanded to return only tables that begin with “AS_”:

```
proc cas;
  session mysess;
  action fileInfo / caslib="pgLib" path=’AS\_%’;
quit;
```

Here is the output from this example:



Results from table.fileInfo

FileInfo Data Source Entities			
Library	Schema	Type	Name
PGLIB	tktstst2	TABLE	AS_DOMAINS
PGLIB	tktstst2	TABLE	AS_LOGINS
PGLIB	tktstst2	TABLE	AS_USERS

Output 3 Results from table.fileInfo

By default, matching is case-sensitive. The ‘AS_%’ pattern does not match tables that begin with a lowercase ‘as_’. You can enable a case-insensitive search by setting the WILDSENSITIVE= parameter to FALSE.

OPTIONS FOR CAS ACTIONS

SPECIFYING OPTIONS

Options for CAS actions are called parameters, and these parameters function much like the LIBNAME and data set options in SAS/ACCESS LIBNAME engines. For most of the actions the DBMS-specific parameters for the SAS data connectors are specified through an OPTIONS= container parameter. This example shows how to pass a specific schema name into your fileInfo action call to only bring back those DBMS tables that exist in a particular DBMS schema.

```
proc cas;
  session mysess;
  action fileInfo / caslib="pgLib" options={schema=”public”};
quit;
```

The exception to the OPTIONS= container parameter in actions is the addCaslib action, where all the DBMS-specific options are specified inside the DATASOURCE= container parameter.

OVERRIDING OPTIONS

Unlike SAS/ACCESS options, SAS data connector parameters are strictly hierarchical in nature. This means that all the parameters are defined at the top level, which in our case means the caslib level. Any action that uses the caslib can override those parameters at the action level. For example, you could define a top-level schema for your database connection at the caslib level with the following addCaslib action:

```
proc cas;
    session mysess;
    action addCaslib lib="pgLib"
    datasource={srctype="postgres",
                  server="myPostgresServer",
                  database="test",
                  username="user",
                  password="password"
                  schema="public"
                 };
run;
```

This means that any action that uses the “pgLib” caslib would automatically use the “public” schema for the database. If you wanted to, for example, look at the column information in another schema during that same PROC CAS session, you could simply override the SCHEMA= parameter in the columnInfo action:

```
        action columnInfo / table={caslib="pgLib" name="cars"}
                            options={schema="test"};
```

This columnInfo would then show column metadata for a table “cars” in the “test” schema instead of the “public” schema.

TRACING

In addition to the SAS logging facility, the SAS data connectors and SAS data connect accelerators provide a set of data source options to enable more detail tracing of the calls to the database.

SAS DATA CONNECTOR TRACING

The SAS data connector driver tracing is used when you want to see the communication between the SAS data connector and the database. The SAS data connector writes a record of each command that is sent to the database to the trace log based on the specified tracing level. Valid values for the tracing levels are shown in the following table:

DRIVER_TRACE= values	**Description**
ALL	Sends all commands to the trace file
API	Sends API method calls to the trace file
DRIVER	Sends driver-specific information to the trace file
SQL	Sends only SQL statements that are sent to the database to the trace file

Error! Reference source not found.

You specify your tracing level with the DRIVER_TRACE= option when defining your caslib.

```
caslib pgLib sessref=mysess
  datasource=(srctype="postgres",
  server="postgres",
  username="Joe",
  password="pa$$word1",
  database="test",
  DRIVER_TRACE=”SQL”,
  DRIVER_TRACEFILE=”~/logs/MyDataConTrace.log”);
```

When turning on driver tracing, you are required to specify the name and location of the trace file where you want the tracing information to be sent. This is done with the DRIVER_TRACEFILE= option in the CASLIB statement.

By default, the trace file is overwritten with every new connection to the database. You can control this behavior with DRIVER_TRACEOPTIONS= option. The DRIVER_TRACEOPTIONS= option enables you to control the formatting and other properties of the trace file. Valid values for the trace options are shown in the following table:

DRIVER_TRACEOPTIONS= values	**Description**
APPEND	Appends trace information to the end of an existing trace file. The contents of the file are not overwritten.
THREADSTAMP	Prepends each line of the trace log with a thread identification.
TIMESTAMP	Prepends each line of the trace log with a time stamp.

Table 3 Trace File Options

The following sample creates a PostgreSQL caslib with tracing fully turned on.

```
      caslib pgLib sessref= mysess
        datasource=(srctype="postgres",
                    server="postgres",
                    username="Joe",
                    password=" pa$$word1",
                    database="test",
                    DRIVER_TRACE=”ALL”,
                    DRIVER_TRACEFILE=”~/logs/MyDataConTrace.log”,
                    DRIVER_TRACEOPTIONS=”TIMESTAMP”);

      caslib oCasLib sessref=mysess path=”/data/myData”
   datasource=(srctype="path”);

      proc casutil sessref=mysess;
        load casdata="oerdat01" casout="myOerdat01Table"
             incaslib=”pgLib” outcaslib="oCasLib";
        contents casdata=”myOerdat01Table" incaslib="oCasLib ";
     run;
     quit;
```

Here is sample output from the above job.

```
21.07.09.53:        ENTER TKTSDriverConnect
21.07.09.53:                0x00000000f9739c20
21.07.09.53:                0x0000000000000000
21.07.09.53:                0x0000000002746060 [     143]
"UID={Joe};PWD=XXXXXXXX;DRIVER_TRACE={ALL};DRIVER_TRACEFILE={~/
logs/MyDataConTrace.log};DRIVER_TRACEOPTIONS={TIMESTAMP};SERVER={postgres};DATABASE={test}"
21.07.09.55:        ENTER TKTSExecDirect
21.07.09.55:                0x00000000f97174a0
21.07.09.55:                0x000000000273c8b0 [      24] "SELECT * FROM "oerdat01""
21.07.09.55:                          24
21.07.09.55:
21.07.09.55:        DEFAULT: SELECT * FROM "oerdat01"
21.07.09.55:        DRIVER SQL: "SELECT * FROM "oerdat01"" on connection 0x00000000f97174a0
21.07.09.55:
21.07.09.55:        SQLNumResultCols: rc=0, ColumnCount=3
21.07.09.56:        SQLDescribeCol: iCol=1,name="I",type=6,colSize=17,decimalDigits=0,nullable=1
21.07.09.56:        SQLDescribeCol: iCol=2,name="J",type=6,colSize=17,decimalDigits=0,nullable=1
21.07.09.56:        SQLDescribeCol: iCol=3,name="W",type=6,colSize=17,decimalDigits=0,nullable=1
21.07.09.56:
21.07.09.57:        EXIT TKTSExecDirect with return code 0 (TKTS_SUCCESS)
21.07.09.57:        ENTER TKTSFetch
21.07.09.57:                0x00000000f97174a0
21.07.09.57:
21.07.09.57:        SQLBindCol: iCol=1,PostgresCType=8,dataPtr=0x00000000f93cf0e0,len=8,indPtr=0x00000000f93cf0f8
21.07.09.58:        SQLBindCol: iCol=2,PostgresCType=8,dataPtr=0x00000000f93cf0e8,len=8,indPtr=0x00000000f93cf100
21.07.09.58:        SQLBindCol: iCol=3,PostgresCType=8,dataPtr=0x00000000f93cf0f0,len=8,indPtr=0x00000000f93cf108
21.07.09.58:
21.07.09.58:        EXIT TKTSFetch with return code 0 (TKTS_SUCCESS)
```

SAS DATA CONNECT ACCELERATOR TRACING

To get tracing information when loading data with the SAS data connect accelerator, you will have to set the data source options TRACEFILE= and TRACEFLAGS=. The TRACEFILE= option specifies the name and location of the trace file where you want your tracing information to be sent. This is similar to the DRIVER_TRACEFILE= option used by the SAS data connectors. The TRACEFLAGS= option is a set of flags that are used to determine how information is placed in the trace log. Valid values for the trace flags are shown in the following table:

TRACEFLAGS= values	Description
TIMESTAMP	Adds a time stamp values before every API call
APPEND	Appends trace information to the end of an existing trace file. The contents of the file are not overwritten

Table 4 Trace Flags Values

The following Teradata caslib enables SAS data connect accelerator logging and sets all of the trace flags. Note that the flags are separated by a pipe symbol.

```
/*Creates a Teradata caslib with EP tracing enabled*/
caslib mycaslibTera sessref=&sess
        datasource=(srctype="teradata",
                    server="server",
                    dataTransferMode="parallel",
                    username="myID",
                    password="myPW",
                    database="database1",
                    traceFile="~/dcEPTrace.log",
                    traceFlags="TIMESTAMP|APPEND”);
```

In addition to the driver tracing, you can specify the DFDEBUG= parameter in the CASLIB statement and/or loadTable action to get additional information back from the SAS Embedded Process. Valid values for the DFDEBUG= parameter are shown in the following table:

DFDEBUG= values	Description
epAll	This will turn on tracing in the EP and return debugging information from the EP job back to the client. This value is only supported for the SAS Data Connect Accelerator for Hadoop.
sqlDetails	This will log the generated SQL back to the client.

Table 5 DFDEBUG Values

The following example enables this additional detail debug SQL logging for loading the table “load00” into the CAS Server with the SAS Data Connect Accelerator for Teradata:

```
proc cas ;
  action loadTable / caslib="tdlib"
                     path=”load00”
                     dataSourceOptions={dfdebug={"sqldetails"}};
run;
```

Here is a sample log output for the above job.

```
NOTE: Connected to: host=myhost  user= model database= model.
NOTE:
select   i.databasename  ,i.tablename  ,i.columnname  ,i.columnposition  ,i.uniqueflag  ,c.columntype  ,c.columnlength
  ,c.DecimalTotalDigits  ,c.DecimalFractionalDigits  from dbc.indicesV i join dbc.columnsV c    on c.databasename =
i.databasename   and c.tablename = i.tablename   and c.columnname = i.columnname where i.databasename =
'model'   and i.tablename = 'load00'   and i.indexnumber = 1
NOTE: HELP COLUMN "model"."load00".*
NOTE: CREATE VOLATILE TABLE sas_run_2469471_933314302 AS ( SELECT   n.NodeId, n.VprocId, c.dt, hps1.*,
hps2.*, hps3.* FROM   (select ProcId as NodeId, min(VprocNo) as VprocId    from table(syslib.monitorvirtualconfig())
v    group by 1) as n JOIN   (select hashamp(hashbucket(hashrow(day_of_calendar))) as
VprocId,      min(day_of_calendar) as dt    from sys_calendar.calendar    group by VprocId) as c ON   c.VprocId =
n.VprocId CROSS JOIN   (select 1 as one, cc as ModelText from _hpsvl933314302 where idd = 1) as hps1 CROSS
JOIN   (select 2 as two, cc as ModelFmt from _hpsvl933314302 where idd = 2) as hps2 CROSS JOIN   (select 3 as
three, cc as GridParms from _hpsvl933314302 where idd = 3) as hps3 ) WITH DATA UNIQUE PRIMARY INDEX (dt)
ON COMMIT PRESERVE ROWS;
NOTE: set session function trace using 'yes' for trace table SAS_SYSFNLIB.SASEP_TRACE_TABLE;
NOTE:  select SAS_SYSFNLIB.sas_load_model('LOAD',NodeId,VprocId,933314302,ModelText,ModelFmt,NULL)  as
results from sas_run_2469471_933314302;
NOTE:  SELECT SAS_SYSFNLIB.sas_load_model('UNLOAD',NodeId,VprocId,933314302,NULL,NULL,NULL)  as
results from sas_run_2469471_933314302;;
NOTE: select Trace_Output from SAS_SYSFNLIB.SASEP_TRACE_TABLE order by Vproc_ID, sequence;
NOTE: set session function trace off;
NOTE: delete from SAS_SYSFNLIB.SASEP_TRACE_TABLE
```

As you can see, the above log will give you detailed information about the SQL query that the SAS Embedded Process sends to the database.

CONCLUSION

The SAS Viya platform is a powerful cloud-ready environment designed to supercharge your analytics, and the SAS data connectors provide the bridge from SAS Viya to your existing DBMS data. The SAS data connector family provides the functionality you need to successfully integrate your third-party database data with the power analytics available in the Cloud Analytic Services.

RECOMMENDED READING

- SAS Institute Inc. 2016. “Working with SAS Data Connectors.” *SAS Cloud Analytic Services: Language Reference*. Available at http://documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.1&docsetId=casref&docsetTarget=p1ez56aqp5uvukn1f96jscujvplm.htm
- SAS Institute Inc. 2016. “Quick Reference for Data Connector Syntax.” *SAS Cloud Analytic Services: Language Reference*. Available at http://documentation.sas.com/?cdcId=vdmmlcdc&cdcVersion=8.1&docsetId=casref&docsetTarget=n0wxsz0rzamws5n1ldbepubijenk.htm

ACKNOWLEDGMENTS

The authors extend their heartfelt thanks and gratitude to these individuals:

Lisa Brown, SAS Institute Inc., Cary, NC

Julia Schelly, SAS Institute Inc., Cary, NC

Jeff Bailey, SAS Institute Inc., Cary, NC

Barbara Kemper, SAS Institute Inc. Cary, NC

CONTACT INFORMATION

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

Chris DeHart
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Chris.DeHart@sas.com
https://www.sas.com

Salman Maher
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Salman.Maher@sas.com
https://www.sas.com

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.

```
/*

This sample is assuming that you have the SAS Data Connector for PostgreSQL and Teradata installed on
your SAS Viya system. I also assuming you have SASHELP.CARS loaded in the default schema for both
PostgreSQL and Teradata

*/

options cashost="<Insert your cas host here>"

/* creates cas session */

cas mysess port=5570;


/* creates new caslib */

caslib pgLib sessref=mysess

    datasource=(srctype="postgres",

          server="<Insert Postgres Server>",

          username="<Insert User Name>",

          password="<Insert Password>");


/*Using traditional postgres libname to upload sashelp.cars*/

libname libpg postgres server="<Insert Postgres Server>" user="<Insert User Name>"

            preserve_tab_names=yes password="<Insert Password>";


/*upload sashelp.cars*/

data libpg.cars; set sashelp.cars;run; quit;


proc cas;

       session mysess;

       action fileInfo / caslib="pgLib";

quit;


/* proc for using the data connector */
```

```
proc casutil sessref=mysess;
   title "list files pgLib";
   list files incaslib="pgLib"; /* list tables on the database */
run;
quit;

proc cas;
    session mysess;
    action columnInfo / table={caslib="pgLib" name="cars"};
run;

/*
 * Load all rows for columns "color" and "units"
 * from the PostgreSQL table "cars" into the CAS table
 * "mycars". Note the CAS table "mycars" is associated to the
 * CAS Library pgLib in this example
 */
proc cas;
    session mysess;
    action loadTable / caslib="pgLib" path="cars"
                       vars={"color","units"}
                       casout="mycars";
run;

/*
 * Get column information for the "mycars" table just created
 * Get table details for the "mycars" table.
 * The tableDetails action only works on CAS tables. You will get an
 * error if you apply that action to a database table.
 */
proc cas;
    session mysess;
    action columnInfo / table={caslib="pgLib" name="mycars"};
    action tableDetails / caslib="pgLib" name="mycars";
run;

proc cas;
    session mysess;
    action loadTable / caslib="pgLib" path="cars"
                    vars={"make","color","units"}
                    where="year = 2016"
                    datasource={numReadNodes=4}
                    casout="cars2016";
run;

proc cas;
   session mysess;
   action tableDetails / caslib="pgLib" name="cars2016";
quit;



/* List tables load into the CAS session mysess */
proc casutil sessref=mysess;
   list tables;
run;
quit;

proc cas;
```

```
   session mysess;

   action fedsql.execdirect / query="select * from ""pgLib"".""cars""";

run;

quit;


proc cas;

   session mysess;

   action fedsql.execdirect /

         query="create table public.foo AS select * from ""pgLib"".""cars"" where make='Ford' and
Cylinders=6";

run;

quit


/*

 * Save cars2016 table back to the PostgreSQL server

 * NOTES: name variable in tables section is the table to save. The name option is the output table
name;

 */

proc cas;

   session mysess;

   action save / caslib="pgLib" name="cars2016Out"

               table={caslib="pgLib" name="cars2016"}

               options={numWriteNodes=4};

run;


proc cas;

   session mysess;

   action save / caslib="pgLib" name="redCars" replace=true

                    table={caslib="pgLib" name="cars2016"
```

```
                              vars={"make"}
                              where="color > red"}
                        options={numWriteNodes=4};
run;

/*
 * Delete cars and cars2016 table we added PostgreSQL at the beginning
 */
proc cas;
   session mysess;
   action deleteSource / caslib="pgLib" source="cars";
   action deleteSource / caslib="pgLib" source="cars2016Out";
   action deleteSource / caslib="pgLib" source="redCars";
run;

/* creates libname to cas */
libname x cas sessref=mysess;

/* outputs data from table loaded into cas above */
proc sql;
  create table baseCars2016 as select * from x.cars2016;
run;
quit;

caslib _all_ list;

/*Force drop of caslib pgLib*/
proc cas;
  action dropCaslib / caslib="pgLib" quiet=TRUE;
```

```
quit;


caslib _all_ list;



/*** Loading Data with SAS Data Connect Accelerator for Teradata ***/
proc cas;
            session mysess;
            action loadDataSource / name="teradata";
quit;



/* creates a Teradata caslib */
caslib teraLib sessref=mysess
       datasource=(srctype="teradata",
                server="<Insert Teradata Server>",
                dataTransferMode="parallel",
                username="<Insert user name>",
                password="<Insert password>");

caslib ndbmslib path="/data/sassfm/sampledata" datasource=(srctype="path");


/* proc for using the data connector */
proc casutil sessref=mysess;
  load casdata="cars" casout="casCars"  incaslib="teraLib" outcaslib="ndbmslib";
  contents casdata="casCars" incaslib="ndbmslib";
run;
quit;
```

```
 /* outputs data from table loaded into cas above */

proc sql;

  select * from x.casCars;

run;

quit;


proc cas;

   session mysess;

   action loadtable / caslib="teraLib" casout="casCars2" path="cars";

   action tableDetails / caslib="teraLIb" name="casCars2";

quit;


caslib teraLib drop;


/* removes cas session */

cas mysess terminate;
```