Paper SAS0331-2017

Introduction to SAS® Data Connectors and SAS® Data Connect Accelerators on SAS® Viya™

Chris DeHart, Salman Maher, and Barbara Kemper, SAS Institute Inc., Cary, NC

ABSTRACT

For many years now you have learned the ins and outs of using SAS/ACCESS® software to move data into SAS® to do your analytics. With the new open, cloud-ready SAS® Viya™ platform comes a new set of data access technologies known as SAS Data Connectors and SAS Data Connect Accelerators. This paper describes what these new data access products are and how they integrate with the SAS Viya platform. After reading this paper, you will have the foundation needed to load data from third-party data sources into SAS Viya.

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 (CAS) 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 similar to 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 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 Connectors are 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).

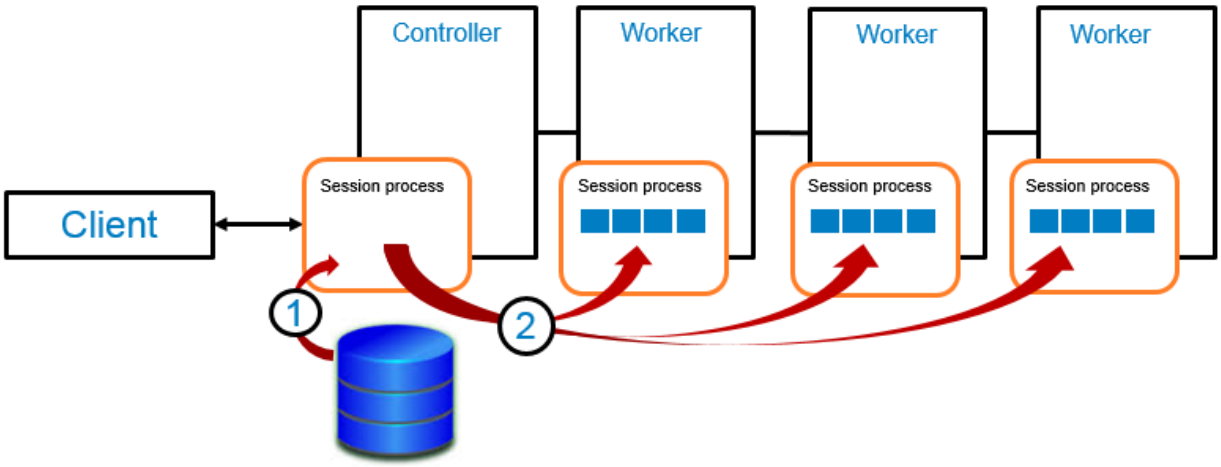

Figure 1 Data Flow for Serial Data Transfer

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.

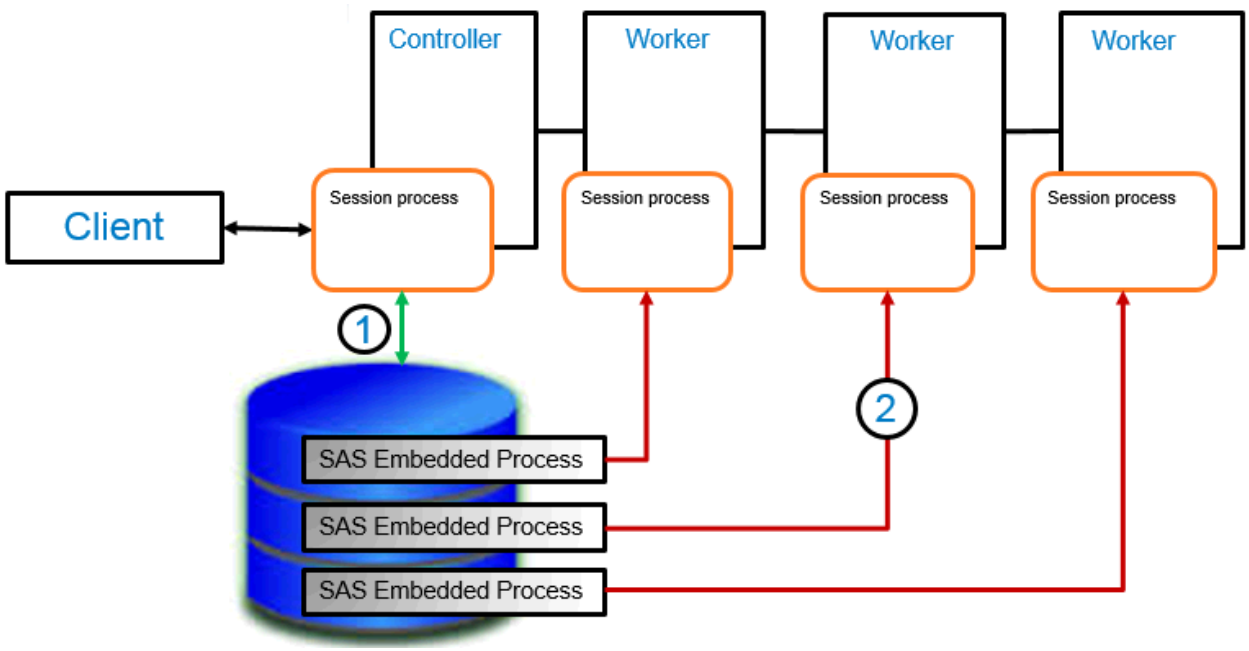

Figure 2 Data Flow for Parallel Data Transfer

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 dataTransferMode is "serial". When you load a table into CAS, you can override the dataTransferMode setting that was specified in the caslib statement by specifying dataTransferMode in the DataSource options parameter of the load table.

```
/* 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.

SAS VIYA CLIENTS

With the release of 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 gives you 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 CAS statement or with the CASLIB procedure (PROC CAS). Regardless of which method you choose to use, you need always need to specify the CAS server host and create a CAS session first 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 parameter. 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 you specified with the sessref parameter on 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 parameter 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 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 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 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 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 you need to be using a Java 8 runtime environment with ANTLR runtime (3.5.2) and Google Protocol Buffers (2.6.1) setup 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 “D” 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 most be set in 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, and fileInfo.

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 for specifying the setting for the output table. For example, if you had a table in your PostgreSQL DBMS named "cars" that you want to load in 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;
```

Subset Your Columns

With the loadTable action vars parameter, you can specify which columns are to be returned from your loadTable action. This allows you to only bring back 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"
      varlist={"color", "units"}
      casout="mycars";
run;
```

Filtering Your Data

If you want to filter the data before it is loaded into a CAS table, then use the data source dbmsWhere parameter. With the dbmsWhere parameter, you can specify your database WHERE clause to be passed directly to the target database without modification. 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"
       datasource={dbmsWhere="year = 2016"}
       casout="cars2016";
run;
```

It is a good time to note that using the WHERE option in a LOAD statement or using the where parameter for a loadTable action will result in an error. You must use the dbmsWhere option if you want to pass a WHERE clause to the DBMS through the data connector.

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;
```

Below 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 loadTable.

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 table from the pgLib caslib defined above:

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

Below 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 option wildignore 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

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 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;
```

Below 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 parameter container.

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.

LOGGING

The primary logging method used by the SAS Data Connectors is the SAS logging facility. In addition to the SAS logging facility, the SAS Data Connectors and SAS Data Connect Accelerators also provide a set of data source options to enable more detail tracing of the calls are being made to your database.

SAS LOGGING FACILITY

The SAS Data Connectors use the SAS logging facility to log interactions between themselves and CAS as well as log processes followed when executing a CAS action. The high-level logger used by the SAS Data Connectors is “App.cas.tkcastkts” and for the SAS Data Connect Accelerator it is “App.cas.actions”. These loggers can be enabled within the logging configuration file on the CAS server or within your SAS program. Here is an example of the PROC CAS code that enable logging for the SAS Data Connectors and set the logging levels:

```
proc cas;
session mysess;
/*Sets the data connectors logging level to TRACE*/
action log / logger='App.cas.tkcastkts' level='trace';
/*Sets the data connect accelerator logging level to DEBUG*/
action log / logger='App.cas.actions' level='debug';
quit;
```

The following lines are written to the SAS log for this statement showing that the logging levels for the specified loggers have been set:

```
56
57 Proc cas;
58   action log / logger='App.cas.tkcastkts' level='trace';
59   action log / logger='App.cas.actions' level='debug';
60 quit;

NOTE: Active Session now MYSESS.
{App.cas.tkcastkts=trace}
{App.cas.actions=debug}
NOTE: PROCEDURE CAS used (Total process time):
      real time           0.03 seconds
      cpu time            0.01 seconds
```

.

The logging level available are:

Level	Description
fatal	The `FATAL` level designates very severe error events that will most likely cause the application to abort.
error	The `ERROR` level designates error events that might still allow the application to continue running
warn	The `WARN` level designates potentially harmful situations.

Level	Description
Info	The `INFO` level designates informational events that highlight the progress of an application at a coarse-grained level.
debug	The `DEBUG` level designates fine-grained informational events that are most useful to debug an application
trace	The `TRACE` level designates finer-grained informational events

Table 2 Logging Levels

The following SAS code generates TRACE logging information:

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

/* creates cas session */
cas mysess;

proc cas;
  session mysess;
  /*Sets the data connectors logging level to TRACE*/
  action log / logger='App.cas.tkcastkts' level='trace';
quit;

/* creates new caslib */
caslib pgLib sessref=mysess
  datasource=(srctype="postgres",
              server="postgres",
              username="Joe",
              password="pa$$word1",
              database="test");

/* proc for using the data connector */
proc casutil sessref=mysess;
  list files incaslib="pgLib"; /* list tables on the database */
run;
quit;

/* This PROC CAS call will return the same results */
/* as the above PROC CASUTIL call                  */
proc cas;
  session mysess;
  action fileInfo / caslib="pgLib";
quit;
```

This is a sample of logging information that is returned by the code:

ls

```
2016-06-17T17:34:17,359 INFO[00000007] qstauto localhost 22169 userid [tkcasaimp.c:2849] - ++
action builtins.log / logger=‘App.cas.tkcastkts', level='debug';
2016-06-17T17:46:16,831 DEBUG[00000007] qstauto localhost 22169 userid [tkcastkts_util.c:621]
- Connection String:
DRIVER=POSTGRES;uid="Joe";pwd=***;server="postgres";database="test";catalog="PGLIB";
```

You can see that each logging message contains the time the message was written, the server that the action was performed on, and the user who executed the action.

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 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. The tracing levels are:

Level	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

Table 2 Logging

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 you want the tracing information to be sent to. This is done with the DRIVER_TRACEFILE option on 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. The trace options are:

Trace Options	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 2 Logging 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;
```

Below 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 option traceFile and traceFlags. The traceFile option takes the name and location of the trace file you want your tracing information to be sent too. 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. The trace Flags are:

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 5 Trace Flags

The following Teradata casLib enables the 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 on the caslib and/or loadTable to get additional information back from the EP. The valid values for the dfDebug parameter are:

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 6 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;
```

Below 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 the SAS Embedded Process is sending 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 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

Tatyana Petrova, SAS Institute Inc., Cary, NC

Jeff Bailey, 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
http://www.sas.com

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

Barbara Kemper
100 SAS Campus Drive
Cary, NC 27513
SAS Institute Inc.
Barbara.Kemper@sas.com
http://www.sas.com