An Introduction to SAS® Viya™ 3.1 Programming for SAS®9 Programmers

Servers and Sessions

SAS Cloud Analytic Services

The high-performance processing power of the SAS Viya platform is SAS Cloud Analytic Services (CAS). CAS is a server that provides the run-time environment for data management and analytics with SAS. (Run-time environment refers to the combination of hardware and software where data management and analytics take place.)

The CAS server has the following characteristics:

- The server can run on a single machine or as a distributed server on multiple machines. For both modes, the server is multi-threaded for high-performance analytics.
  - The distributed server consists of one controller and one or more workers. This architecture is often referred to as a massively parallel processing (MPP) architecture.

    Using single-server symmetric multi-processing (SMP), the threaded processing of the CAS server is shared by multiple CPUs or is shared between multiple cores of a single CPU.
  - The distributed server has a communication layer that supports fault tolerance. A distributed server can continue processing requests even after losing connectivity to some nodes. The communication layer also enables you to remove or add worker nodes from a server while it is running.
- The CAS server can manage all of your data and easily share data with multiple users.
- Code that runs on the CAS server runs in a distributed, multi-threaded environment. You can program using new high-performance SAS procedures or CAS actions. CAS actions are the smallest unit of work for the CAS server. To program using CAS actions, you use either the new CAS procedure or you use the SAS Scripting Wrapper for Analytics Transfer (SWAT) interface from Python and Lua, or the CAS socket protocols for Java. The language interface contains the classes that you need to communicate with the CAS server.

The DATA step, and the DS2 and FedSQL languages run in the CAS server. The CAS server does not support SAS catalogs.

- The CAS server processes in-memory tables. The source data for the in-memory tables can come from SAS data sets, server-side files, event stream processing, and database files. Database files can be loaded serially or in parallel.
- SAS Studio provides a SAS programming environment for developing and submitting programs to the server. Languages such as Python, Lua, and Java can submit code to the CAS server by using the open-source code interface.

See Also

- *SAS Cloud Analytic Services: Fundamentals*

The SAS Workspace Server

The SAS workspace server uses the V9 engine to process SAS data sets locally and to read external data. The new data mining and machine learning procedures, statistical procedures, and other supporting procedures run on the CAS server, not the SAS workspace server. Output is returned to the client. The DATA step, the SAS macro language, the data management and utility procedures, and DS2 and FedSQL run in the SAS workspace server. The SAS workspace server supports SAS catalogs.

Server Sessions and Executing Code

When you log on to SAS Studio, a session with the SAS workspace server is created. To use SAS Viya programming, you start a session with the CAS server so that the CAS server can operate on your in-memory data tables.

Because the DATA step can run locally as well as in the CAS server, SAS first determines whether the DATA step code can run in the CAS server. If attempts to run the DATA step in the CAS server fail, SAS determines whether the code can execute in the SAS Workspace Server.

A First Look at Programming Using SAS Cloud Analytic Services

Examples and Sample Data

These examples show the essentials of programming in the CAS server using SAS and Python. In both environments, you must be connected to the CAS server and have a session with the server. If you are programming in SAS, the CAS server connection is created for you. You then initiate a session with the server using the CAS statement. In Python, you specify the session name when you connect to the CAS server.

Some examples use data from the Sashelp library. If you find that a data set is not in the Sashelp library, you can install the sample data to the Sashelp library if you have sudo privileges. From a Linux command line, enter **`sudo yum install sas-samplesml`**. If you do not have sudo privileges, contact your administrator to install the sample data.

To download CSV files, see SAS Viya Example Data Sets.

A Gradient Boosting Model Procedure

This example initiates a session with the CAS server, loads data to the server, and creates a gradient boosting model using the GRADBOOST procedure:

```
cas casauto;                                                        /* 1 */
caslib casuser;                                          ;          /* 2 */

libname mycas cas;                                                  /* 3 */

data mycas.junkmail;                                                /* 4 */
   set sashelp.junkmail;
run;

proc gradboost data=mycas.junkmail outmodel=mycas.gradboost_model;  /* 5 */
   input Address Addresses All Bracket Business CS CapAvg CapLong
         CapTotal Conference Credit Data Direct Dollar Edu Email
         Exclamation Font Free George HP HPL Internet Lab Labs
         Mail Make Meeting Money Order Original Our Over PM Paren
         Parts People Pound Project RE Receive Remove Semicolon
         Table Technology Telnet Will You Your _000 _85 _415 _650
         _857 _1999 _3D / level = interval;
   target class /level=nominal;
   output out=mycas.score_at_runtime;                               /* 6 */
   ods output FitStatistics=fit_at_runtime;
run;
```

1. Create a session with the CAS server.
2. Use the CasUser caslib. A caslib is an in-memory space for tables. The CAS server operates only on tables in a caslib. In this example, CasUser is the name of the personal caslib that your administrator set up for your user ID.
3. The LIBNAME statement for the CAS engine binds the libref MyCas to the active caslib, CasUser.
4. The DATA step loads the Sashelp.Junkmail data set to the server-side table Junkmail in the CasUser caslib.
5. Execute the GRADBOOST procedure. The input data comes from the CasUser caslib because the libref MyCas is bound to the CasUser caslib. The same is true for the output model table Gradboost_Model.
6. Specify the output scoring table with the OUTPUT statement.

To further explore this example and to view the output, see the GRADBOOST procedure.

See Also

- CAS Sessions on page 11
- Caslibs on page 12
- The CAS Engine on page 13
- Load Data to a Caslib on page 13

A Factorization Machine Model Using the Factmac CAS Action

This example uses the Factmac action, which uses observations in a data table to train a factorization machine model. In this case, the data is derived from companies that provide movies for online viewing. A company wants to offer its customers recommendations of movies that they might like. These recommendations are based on ratings that are provided by users.

```
cas casauto;                                                /* 1 */
caslib casuser;                                             /* 2 */

proc casutil;                                               /* 3 */
    load file="u.data"
    casout="movlens"
    importoptions=(filetype="CSV" delimiter="TAB" getnames="FALSE"
                   vars=("userid" "itemid" "rating" "timestamp"));
run;

proc cas;                                                   /* 4 */
      action factmac result=R / table={name="movlens"},     /* 5 */
      outModel={name="factors_out", replace=true},          /* 6 */
      inputs={"userid", "itemid"},                          /* 7 */
      nominals={"userid", "itemid"},                        /* 8 */
      target="rating",                                      /* 9 */
      maxIter=20, nFactors=10, learnStep=0.15,              /* 10 */
      output={casout={name="score_out", replace="TRUE"},    /* 11 */
      copyvars={"userid","itemid","rating"}};
run;
```

1 Create a session with the CAS server.

2 Use the CasUser caslib. A caslib is an in-memory space for tables. The CAS server operates only on tables in a caslib. CasUser is the personal caslib that is set up for your user ID by the administrator.

3 Use the CASUTIL procedure to load the data to the in-memory table Movlens using the import options that are specified in the `importoptions` parameter. The file u.data contains the movie ratings.

PROC CASUTIL is a new data management procedure for managing CAS server tables.

4 Execute the CAS actions with the CAS procedure.

PROC CAS is a new procedure for executing CAS actions. In addition, you can code using the new CAS language (CASL) within PROC CAS.

5 Execute the Factmac action from the Factmac action set, and place the results in the variable R. The table parameter names the input data table to be analyzed. An action sends a request to the server and returns a result. The results are usually a data table or a metadata table.

6 The `outModel` parameter creates output for the fitted parameter estimates to the factors_out data table.

7 The `inputs` parameter specifies the input variables.

8 The `nominals` parameter specifies that the `userid` and `itemid` variables are nominal.

9 The `target` parameter specifies the target variable.

10 The `maxIter` parameter specifies the number of iterations for the optimization. The `nFactors` parameter specifies the number of factors to be estimated for the factorization machine model. The `learnStep` parameter specifies the learning step size for the optimization.

11 The `output` parameter names the in-memory table in the `casout` parameter and specifies that an existing table is to be replaced.

For more information about this example, see the MovieLens data example for the Factorization Machine action set.

See Also

- CAS Actions on page 22
- *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*

A Python Program to Categorize Continuous Variables into Bins

You can use CAS actions to program in Python. You must have the SAS Scripting Wrapper for Analytics Transfer package to connect to the CAS server. The SAS Scripting Wrapper for Analytics Transfer package contains the classes that are required to connect to the CAS server and the CAS action classes. The following example uses the Binning action to categorize continuous variables into bins. It uses a table that contains information about cars.

```
import swat                                                    /*1*/
s=swat.CAS("cloud.example.com", 5570)                          /*2*/
from IPython.core.display import display, HTML                 /*3*/
r=s.table.addCaslib(caslib="casdata",                          /*4*/
                    dataSource={"srcType":"path"},
                    path="/rdstore/data/casdata")
r=s.table.loadTable(casOut={"name":"cars", "replace":True},    /*5*/
      caslib="casdata",
      importOptions={"fileType":"CSV"},
      path="cars.csv"
      )
s.datapreprocess.binning(table={"name":"cars"},                /*6*/
                         vars=["MPG_City"]
                        )
```

1 Import the swat library.

2 Specify the CAS host name and CAS port number to use if you are not already connected to a CAS session. The CAS session name is S. You prefix CAS action calls with the session name and a period (.).

3 Import display functionality and HTML functionality for use in this example.

4 Create the CasData caslib by using the table.addCaslib action.

5 Load the cars CSV file from the CasData library by using the table.loadTable action.

6 Call the Binning action in the Data Preprocess action set by using the cars file. This call specifies to use MPG_City as the analysis variable.

For more information about this example, see " Categorize Continuous Variables into Bins" in *SAS Cloud Analytic Services: Analytics Programming Guide*.

See Also

- *SAS Cloud Analytic Services: Getting Started with Python*

Getting Started with Machine Learning

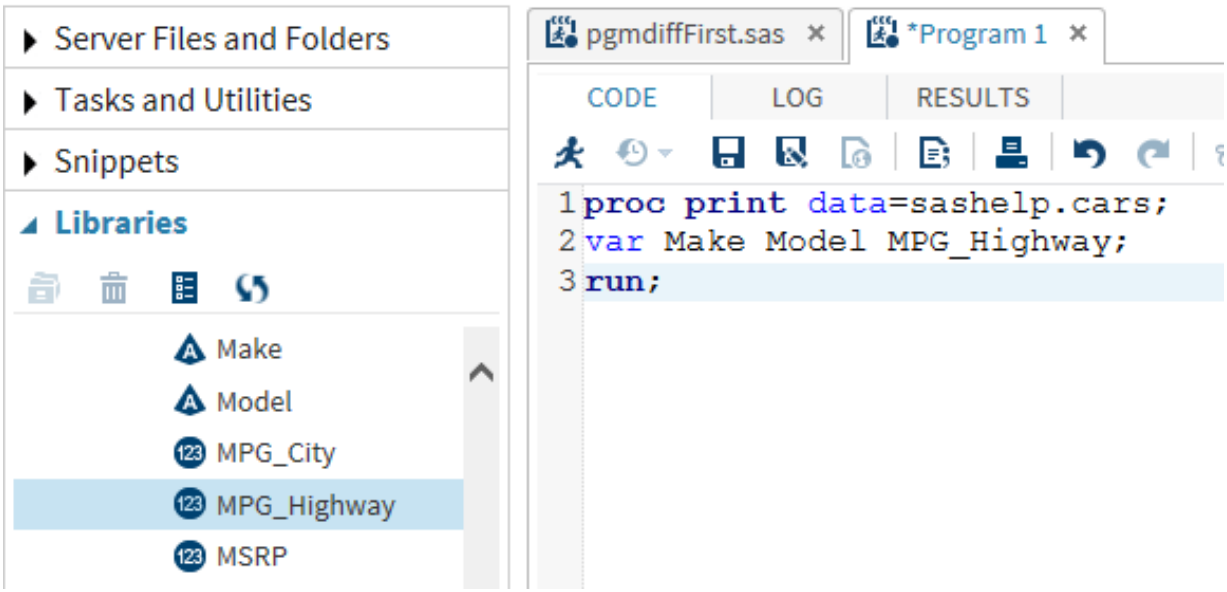
Getting Started with SAS Visual Data Mining and Machine Learning is a case study that demonstrates the new high-performance analytic procedures. You can explore and prepare data, create and assess models, and score new data. Follow along with this case study by downloading the data and copying the code to the SAS Studio programming interface.

Programming Interfaces

SAS Studio

SAS Studio is a web programming editor. It is the only graphical editor that is supported in SAS Viya. SAS Studio is designed to help you write your SAS programs as quickly and accurately as possible. You can program using the point-and-click interface, or you can enter your code directly on the **CODE** tab.

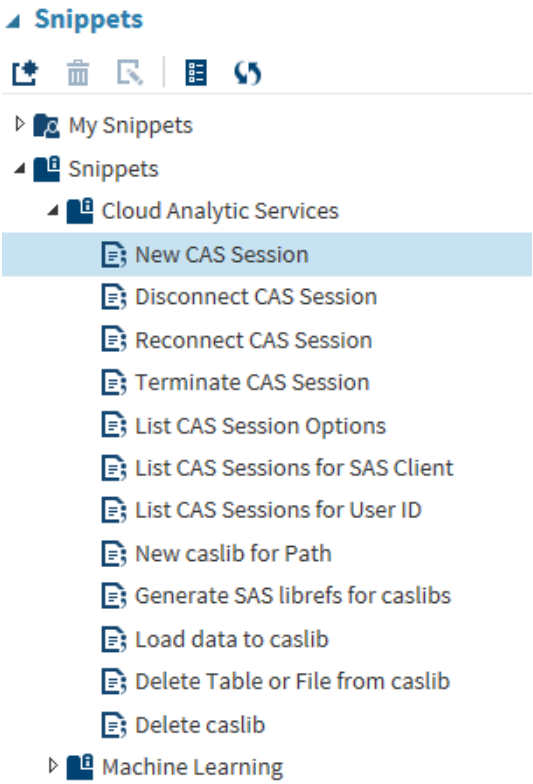
- Click **Libraries** in the Navigation pane to access all of your libraries and the tables in those libraries.
- Save time when you are writing a program by dragging items from the Libraries section to your program. SAS Studio adds the code for those items to your program. In the following figure, the variables Make, Model, and MPG_highway were dragged to the program in the **CODE** tab:



- Each time you submit code, the **LOG** tab displays messages that indicate whether the submission was successful. Table output from your program appears on the **RESULTS** tab. By default, SAS Studio creates HTML5 output. On the **RESULTS** tab, you can click one of the first three buttons to download the results in HTML5, PDF, or RTF formats:

To assist you in coding, SAS Studio provides snippets and tasks that automatically generate code when you drag them to the **CODE** tab. A snippet is code that you might use frequently. The following example explains how to use snippets in your programming.

1 In the Navigation pane, expand Snippets for a list of Cloud Analytic Services and Machine Learning snippets.



2 Drag the New CAS Session snippet to the **CODE** tab and submit the code. A CAS session named MySession that uses the default caslib CasUser is established. (A caslib is an in-memory space where the CAS server operates on tables.)

```
/*****************************************************************************/
/*  Start a session named mySession using the existing CAS server connection */
/*  while allowing override of caslib, timeout (in seconds), and locale     */
/*  defaults.                                                                */
/*****************************************************************************/

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

3 Drag the **Load data to caslib** snippet to the **CODE** tab. Delete the code above and below the PROC CASUTIL code that loads a SAS data set from a Base engine library.

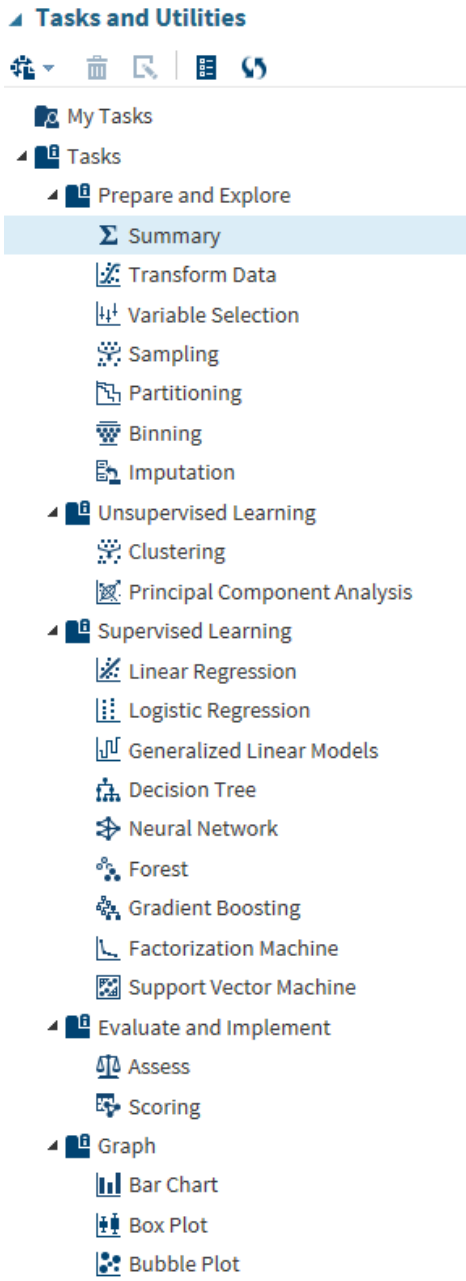
```
/*****************************************************************************/
/*  Load SAS data set from a Base engine library (library.tableName) into    */
/*  the specified caslib ("myCaslib") and save as "targetTableName". The     */
/*  PROMOTE option makes loaded data available to all active sessions.       */
/*****************************************************************************/

proc casutil;
        load data=library.tablename outcaslib="myCaslib"
        casout="targetTableName" promote;
run;
```

4 Customize the code:

- Replace `library.tablename` with `sashelp.cars`.
- Replace `myCaslib` with `Casuser`.
- Replace `targetTableName` with `cars`.

5 Submit the code. The Cars data set from the Sashelp library is loaded to the Cars table in the CasUser caslib.

6 Now that you have data in a caslib, you can use the **Machine Learning** snippets or the **Tasks** to explore, model, and score data. Here are the tasks that assist you by generating code:



See the following videos for a demonstration of using the SAS Studio point and click interface:

- Video: Code Snippets
- Video: Exploring Data

See Also

- *Getting Started with Programming in SAS Studio*
- *SAS Studio: User's Guide*

Batch and Interactive Line-Mode Processing

You can submit code in batch mode and interactive line mode from the command line as you have in previous releases of SAS, with these differences:

- Only Zip, SFTP, and URL access methods are available in SAS Viya.
- You must set the CAS server port in the SAS command or autoexec file. For example:

```
-casport 5570
```

See Also

- *Batch and Line Mode Processing in SAS Viya*

Open-Source Code for Python, Lua, and Java

SAS open-source code provides an interface to the CAS server using CAS actions from Python, Lua, and Java. A CAS action is the smallest unit of work in the server. Loading an in-memory table from a file is an example of an action. Related CAS actions are grouped in an action set. For example, the table.addTable and table.loadtable actions are actions in the Table action set.

The client machine that runs Python, Lua, or Java applications is required to use the following:

- 64-bit Linux
- 64-bit version of Python, Lua, or Java

  The following versions of the scripting languages are compatible with the SAS interfaces:

  - Python 2.7.x or Python 3.4.x
  - Lua 5.2 or Lua 5.3
  - Oracle JDK SE Java 8
  - OpenJDK version 1.8.0.65

For Python and Lua, you use the SAS Scripting Wrapper for Analytics Transfer interface to connect to the CAS server. For Java, you use the CAS socket protocols for Java to connect to the CAS server. You can download the client interfaces from these locations:

Python	http://support.sas.com/downloads/package.htm?pid=1977
Lua	http://support.sas.com/downloads/package.htm?pid=1975
Java	http://support.sas.com/downloads/package.htm?pid=1976

This Python example connects to the CAS server, starts a session, and executes the listSessOpts action to list the name, type, and current value for all of the session options in session MyCasSess.

```
import swat

# Create session CASAUTO. Specify the CAS host and port for your site.
s = swat.CAS("cloud.example.com", 5570)
s.sessionName(name="mycassess")

# Get the current session options.
```

```
res = s.sessionProp.listSessOpts()

# Print the option name and value for each option.
print "Current session options:"
display(res.SessOpts.ix[:,['Name','Type','Value']])
```

Many of the action examples in the Help Center show examples in the new CAS language (CASL), Lua, and Python. For example, the Aggregation Action Set examples provide CASL, Lua, and Python links below the navigation buttons. You can find additional examples in the SAS Viya community.

Submitting SAS 9 Code to SAS Viya

Using SAS/CONNECT, SAS 9.4 users can connect to and use SAS Viya. SAS users can use SAS/CONNECT statements to create a virtual bridge between the two environments. The SAS/CONNECT bridge enables SAS 9.4 solution users to leverage SAS Viya performance while using familiar projects, interfaces, and SAS code.

Video: Bridging SAS 9 and SAS Viya

For more information, see *Integrating SAS 9.4 and SAS Viya*.

Data Migration to UTF-8 Session Encoding

In SAS Viya, UTF-8 is the only session encoding. UTF-8 is variable-width, multi-byte encoding. One UTF-8 character can be 1–4 bytes in length. SAS Viya requires that data set character encoding match the SAS session encoding. When they do not match, the data set must be transcoded to UTF-8.

If SAS Viya can determine the data encoding, it automatically transcodes non-UTF-8 encodings to UTF-8 when it reads data. If the encoding cannot be determined, you must specify a transcoding option in the statement that you use to read the data.

CAUTION! Data truncation can occur during transcoding. If a data set column is not wide enough to store the results that are encoded as UTF-8 when data transcoding occurs during the reading of character data, then data might be truncated. You must use the CVP (Character Variable Padding) engine to widen variable lengths when the data is read. If a variable width is associated with a format, the format width is not increased. You must re-create the format using the FORMAT procedure. If there is the possibility of truncated data, an error message appears in the log.

Because the ASCII encoding is compatible with the UTF-8 encoding, no transcoding is necessary. ASCII is a 7-bit encoding standard that provides a basic set of 128 characters. It encodes the uppercase and lowercase letters of the English alphabet, punctuation marks, the digits 0–9, and control characters.

Indexes and format catalogs must also be compatible with the UTF-8 session encoding.

For complete information about migrating SAS data sets to SAS Viya, see these publications:

- *Migrating Data to UTF-8 for SAS Viya*
- *SAS Viya FAQ for Processing UTF-8 Data*

See Also

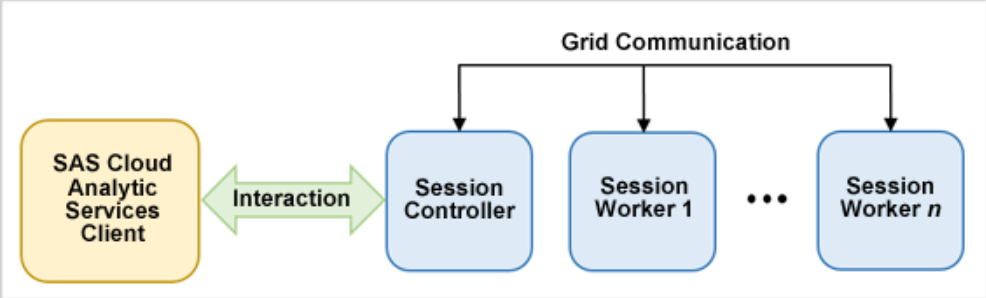
- "LIBNAME Statement, CVP Engine" in *SAS Viya Statements: Reference*
- "Avoiding Character Data Truncation By Using the CVP Engine" in *SAS Viya National Language Support: Reference Guide*

CAS Sessions

When you log on to SAS Studio in SAS Viya, SAS creates a single session with the workspace server. You use the workspace server as you did in SAS 9 or in other prior SAS releases. To use CAS, you explicitly start a session with the CAS server.

When you create a session, the server first authenticates your identity. If your identity is successfully authenticated, the server starts the session controller process. That process sends a message to each machine in the cluster to start a session process for your client, as shown in the following figure.

Session Processes in a Distributed System



Sessions provide the following functionality:

- user identification.
- fault isolation for each session. If a problem occurs in your session, it does not impact other sessions, the server, or the client.
- efficiency. Resources that are visible only to your session are used, and large amounts of data are processed concurrently when needed
- resource tracking, which is enabled through the use of resource metrics

You use the CAS statement to create a session. Add the CAS statement to your program either by entering the statement or by using the SAS Studio code snippet **New CAS Session**:

```
/*****************************************************************************/
/*  Start a session named mySession using the existing CAS server connection */
/*  while allowing override of caslib, timeout (in seconds), and locale     */
/*  defaults.                                                                */
/*****************************************************************************/

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

You specify the values that are particular for your environment.

See Also

- "Sessions" in *SAS Cloud Analytic Services: Fundamentals*
- "CAS Statement" in *SAS Cloud Analytic Services: Language Reference*
- "Working with Code Snippets" in *SAS Studio: User's Guide*
- "Editing the Autoexec File" in *SAS Studio: User's Guide*

Caslibs

The CAS server operates only on tables that reside in the active caslib.

A caslib is an in-memory space to hold tables, authorization information, and data source information. Authorized users can add or manage caslibs with the CASLIB statement. Caslib authorization is set by your administrator. In some instances, such as when you copy native CAS tables that are not in memory, a caslib is required although data is not copied to the caslib.

Caslibs can be of the following types:

- personal and automatically available for a given user
- pre-defined by an administrator
- manually added

When you start a CAS session, your personal caslib is added by default. It is the active caslib, and it is named Casuser(*your-user-ID*). When you specify the caslib in your code, you use the name CasUser only, you do not need to specify CasUser(*your-user-ID*). Your personal caslib is available only to you, and you cannot share the tables in your caslib. To share tables, you can promote your personal caslib to be global scope. Also, your site administrator can define caslibs with global scope for anyone to use.

The following output lists the metadata that is associated with a personal caslib. The code to obtain this information is `caslib casuser list;`.

```
57         caslib casuser list;
NOTE: Session = MYSESSION Name = CASUSER(my-user-ID)
         Type = PATH
         Description = Personal File System Caslib
         Path = /r/cloud.sas.com/vol/vol999/u61/my-user-ID/casuser/
         Definition =
         Subdirs = Yes
         Local = No
         Active = Yes
         Personal = Yes
```

You use the CASLIB statement to create a new caslib if you are authorized to do so. When you use the CASLIB statement, that caslib becomes the active caslib.

The active caslib is used as the default data source if you do not override it. You can override the active caslib using the CASUTIL procedure or the CASLIB= data set option for a CAS engine libref.

Only one caslib can be active at a time.

To view all caslibs, enter `caslib _all_ list;`

SAS Studio provides a **Cloud Analytic Services** snippet to create a new caslib:

```
/*****************************************************************************/
/*  Create a CAS library (myCaslib) for the specified path ("/filePath/")    */
/*  and session (mySession).  If "sessref=" is omitted, the caslib is        */
/*  created and activated for the current session.  Setting global makes     */
/*  myCaslib visible in all active sessions. The default is local.  Setting  */
/*  subdirs extends the scope of myCaslib to subdirectories of "/filePath".  */
/*****************************************************************************/

caslib myCaslib datasource=(srctype="path") path="/filePath/" sessref=mySession global subdirs;
```

Specify the values that are particular for your environment.

In the following example, the CASLIB statement creates a global caslib MyVaPublic for an HDFS data source.

```
caslib Myvapublic path="/vapublic"
                          datasource=(srctype="hdfs") global ;
```

Alternatively, you can specify the caslib as an option in the CAS statement:

```
cas casauto sessopts=(caslib="casuser");
```

Video: Accessing, Manipulating, and Saving Data Using SAS Cloud Analytics on SAS Viya.

See Also

- "Caslibs" in *SAS Cloud Analytic Services: Fundamentals*
- "CASLIB Statement" in *SAS Cloud Analytic Services: Language Reference*
- "CASLIB= LIBNAME Option" in *SAS Cloud Analytic Services: Language Reference*
- "CASUTIL" in *SAS Cloud Analytic Services: Language Reference*

The CAS Engine

The CAS engine connects the SAS Workspace Server with the active caslib in the CAS server. A CAS engine libref must be defined before the following types of data transfers can occur between the two servers:

- loading SAS data sets to a caslib as an in-memory table
- downloading CAS tables or metadata as input to a DATA step or SAS procedure when it executes on the SAS Workspace Server

  Note: Downloading a CAS table with millions of rows can cause network degradation.

When you define a CAS libref using the LIBNAME statement, the CAS engine binds with the active caslib unless you specify the LIBNAME statement CASLIB= option.

Here are two examples of the CAS LIBNAME statement:

```
libname mycas cas;
```

```
libname mycas cas caslib=testTables;
```

See Also

"CAS LIBNAME Statement" in *SAS Cloud Analytic Services: Language Reference*

Load Data to a Caslib

You can load a SAS data set, database tables, or a CAS table to a caslib. After the data is in a caslib, you can use a DATA step, procedures, CAS actions, or DS2 or FedSQL operations on the CAS table. Tables are not automatically saved when they are loaded to a caslib. You can use PROC CASUTIL to save tables. After you save native CAS tables, they have the file extension .sashdat.

Use the DATA Step to Load Data

You can use the DATA step SET statement to load a SAS data set to CAS:

```
cas casauto;                /* 1 */
caslib;                     /* 2 */

libname mycas cas;          /* 3 */

data mycas.cars;
   set sashelp.cars;        /* 4 */
run;
```

1 Create the CasAuto session to the CAS server.

2 Use the caslib CasUser;

3 Create the libref MyCas and bind it to the CasUser caslib.

4 Load the Sashelp.Cars data set to the CasUser caslib.

Use PROC CASUTIL to Load Data

This example uses PROC CASUTIL to load a data set from the Sashelp library to the caslib MyCaslib. The second PROC CASUTIL saves the table.

```
cas casauto;                                          /* 1 */

caslib mycaslib datasource=(srctype="path")
   path="/myCasTables/" sessref=casauto;              /* 2 */

libname mycas cas;                                    /* 3 */

proc casutil outcaslib="mycaslib" ;
   load data=sashelp.heart replace;                   /* 4 */

run;

data mycas.heart;
   set mycas.heart;
   keep status BP_status weight_status
        smoking_status chol_status deathcause;        /* 5 */
run;

proc print data=mycas.heart; run;                     /* 6 */

proc casutil  incaslib="mycaslib" outcaslib="mycaslib"; /* 7 */
   save casdata="heart" replace;
run;
```

1 Start the CAS session Casauto.

2 Create the caslib CasUser. The data source for this caslib is a server directory path and the caslib is associated with the CasAuto session. If you omit the SESSREF option, the caslib is associated with the active session.

3 Create a CAS engine libref to load the data to. By default, the LIBNAME statement binds the libref to the active caslib.

4 Load the Sashelp.Heart data set to the CasUser caslib using the LOAD DATA statement.

5 Keep a limited number of variables from the original data set.

6 View the results. The table Mycas.Heart is an in-memory table. This table exists in memory for the duration of your CAS session. PROC PRINT processes in the SAS session by transferring the data from mycas.heart table on the CAS server.

   Note: Printing large SAS Cloud Analytic Services tables can cause a large amount network traffic. You can use the OBS= system option to limit the number of rows the PRINT procedure processes.

7 Save the table to the server using the SAVE CASDATA statement in PROC CASUTIL. To save a table in CAS, you specify the caslib and the table.

Using PROC CASUTIL, you can perform several table maneuvers such as loading and dropping tables, listing table, and promoting tables. This code lists all tables in the active caslib:

```
proc casutil;
   list tables;
run;
```

Video: Loading Data

See Also

- "CASUTIL" in *SAS Cloud Analytic Services: Language Reference*
- "Tables Action Set Details" in *SAS Cloud Analytic Services: System Programming Guide*

Use CAS Actions to Load Data

PROC CAS is a new procedure that interacts with the CAS server by using CAS actions. An advantage to using CAS actions for table management is that there are more features to the Table actions that you can use in PROC CAS than are available in PROC CASUTIL. Here is an example that loads the iris.sashdat table from the server to the active caslib and names the table IRIS:

```
proc cas;
   table.loadtable /path="iris.sashdat" casOut={name="IRIS"};
run;
```

Some of the features in the Table action set that are not in PROC CASUTIL are managing caslibs, creating user-defined table attributes, partitioning tables, fetching table rows from a table, updating table rows, and creating a star schema based on a view. See Table Action Set: Syntax for a complete list of Table actions.

See Also

- "CAS" in *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*
- CAS Actions

Use Data Connectors to Load Data

A data connector is the software that is used with a caslib to read server-based data sources like databases and Hive. Data connectors for file-based caslibs control reading data files such as file paths and setting the file encoding. Each data source type has a set of connection options that are specific for that data source.

Loading tables into the server from a caslib's data source is the most efficient way to load data. You use the CASLIB statement DATASOURCE= option to load data using a data connector.

Here is an example that specifies the connection options for a Hive data source:

```
caslib hivelib desc="Hive Caslib"
     datasource=(srctype="hive",
```

```
dataTransferMode="parallel",
hadoopjarpath="/data/cdh54/sdm/lib",
hadoopconfigdir="/data/cdh54/sdm/conf",
username="hiveuser",
server="hive01.example.com",
schema="default");
```

See Also

"Data Connectors" in *SAS Cloud Analytic Services: Language Reference*

Data Types

The SAS Cloud Analytics Services (CAS) server supports the VARCHAR data type in addition to the CHARACTER and NUMERIC data types, which are traditionally supported by SAS. Variables that are created using the VARCHAR data type are varying width and use character semantics, rather than being fixed-width and using byte semantics of the traditional CHARACTER data type. Using the VARCHAR data type in the DATA step in the CAS server has some restrictions. For more information, see "VARCHAR Data Type " in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.

The CHARACTER and NUMERIC data types continue to be the supported data types for processing in the SAS Workspace Server.

The DS2 language supports several additional data types. On the CAS server, DS2 converts non-native data types to CHARACTER, NUMERIC, or VARCHAR. For information about data types that are supported for specific data sources, see "Data Type Reference" in *SAS Viya: DS2 Language Reference*.

The CAS language (CASL) determines the data type of a variable when it is assigned. You do not declare variables in CASL.

In the following table, the letter Y indicates the data types that are supported for programming on the CAS server, and in the last column, on the Workspace server:

Data Type	CAS Actions	CASL	Procedures and DATA Step	DS2	FedSQL	Workspace Server Processing
BIGINT				Y		
BLOB		Y				
BOOLEAN		Y				
CHARACTER (CHAR)	Y	Y	Y	Y	Y	Y
DATE		Y		Y		
DATETIME		Y				
DOUBLE	Y	Y	Y	Y	Y	
FLOAT				Y		
INTEGER				Y		

Data Type	CAS Actions	CASL	Procedures and DATA Step	DS2	FedSQL	Workspace Server Processing
INT32		Y				
INT64		Y				
ITEMS		Y				
LISTS		Y				
NCHAR				Y		
NUMERIC (NUM)						Y
NVARCHAR				Y		
SMALLINT				Y		
STRING UTF-8		Y				
TABLE		Y				
TIME		Y		Y		
TIMESTAMP				Y		
TINYINT				Y		
VARCHAR	Y		Y	Y	Y	

* Some of the SAS utility procedures read CAS tables or metadata about the tables, but do not run on the CAS server. For example, the PRINT procedure reads and prints CAS tables, including VARCHAR columns. For a list of utility procedures that support CAS data, see SAS Procedures.

** The SAS NUMERIC data type is the same as a DOUBLE data type.

Note: The CVP LIBNAME engine converts the CHAR data type to VARCHAR if you specify the CVPVARCHAR=YES LIBNAME option.

Explore Data Using Statistical Procedures

The statistical procedures run only on in-memory CAS tables. The procedures run in parallel over multiple CPUs in a single-server configuration or over multiple nodes in a compute cluster. The procedure statements and their options initiate the CAS actions that execute on the CAS server.

For examples, see the individual procedures listed in the table.

Procedure	Description
ASSESS	Assesses and compares supervised learning models. ■ For a supervised learning model that has a nominal target, the ASSESS procedure produces lift information and receiver operating characteristic (ROC) information. ■ For a regression model, the ASSESS procedure performs a quantile binning of the predictions, and then returns the summary statistics of the response variable for each bin. ■ Calculates fit statistics such as average square error, mean square logarithmic error, mean absolute error, mean consequential error, and multiclass log loss.
BINNING	Performs binning, which is a common step in the data preparation stage of the model-building process. You can use binning to classify missing variables, reduce the impact of outliers, and generate multiple effects. The generated effects are useful and contain certain nonlinear information about the original interval variables. The BINNING procedure supports several binning methods and can calculate the weight of evidence (WOE) and information value (IV), based on binning results. Video: Transforming Data
CARDINALITY	Determines a variable's cardinality or limited cardinality. The cardinality of a variable is the number of its distinct values. The limited cardinality of a variable is the number of its distinct values that do not exceed a specified threshold. Video: Selecting Features
GENSELECT	Provides model fitting and model building for generalized linear models. The GENSELECT procedure fits models for standard distributions in the exponential family, such as the normal, Poisson, and binomial distributions. It also fits models for responses, such as those modeled as the beta, generalized Poisson, and negative binomial distributions. For all these models, the GENSELECT procedure provides forward and backward variable selection. It also provides model selection by the LASSO method.
KCLUS	Performs clustering, which is a common step in data exploration. Video: K-Means Clustering Video: Principal Component Analysis
LOGSELECT	Fits binary and binomial response models.
MDSUMMARY	Computes basic descriptive statistics in parallel for CAS tables.
NLMOD	Uses either nonlinear least squares or maximum likelihood to fit nonlinear regression models.
PARTITION	Performs simple random sampling, stratified sampling, and oversampling to produce a table that contains a subset of observations or partitioned observations. Video: Exploring Data
PCA	Performs principal component analysis, which is a multivariate technique for examining relationships among several quantitative variables. Video: Principal Component Analysis

Procedure	Description
PLSMOD	Fits models by using one of a number of linear predictive methods, including partial least squares (PLS). The procedure accounts for variation in the predictors under the assumption that directions in the predictor space that are well sampled should provide better prediction for new observations when the predictors are highly correlated.
QTRSELECT	Chooses and fits quantile regression models. Quantile regression is a systematic statistical methodology for modeling quantile functions of a response variable, conditional on explanatory variables. The procedure offers extensive capabilities for customizing the model selection with a variety of selection and stopping criteria. PROC QTRSELECT also provides a set of diagnostics for the selected models.
REGSELECT	Fits and performs model selection for ordinary linear least squares models.
TREESPLIT	Builds tree-based statistical models for classification and regression. Video: Auto-tuning: Decision Tree
VARIMPUTE	Performs numeric variable imputation, which is a common step in data preparation. Video: Transforming Data
VARREDUCE	Performs both supervised and unsupervised variable selection. You can use the VARREDUCE procedure to read data in distributed form and to perform variable selection in parallel. Video: Selecting Features

This example performs a simple random sampling:

```
cas casauto;                                                          /* 1 */
caslib casuser;

libname mylib path="/mySource/";                                      /* 2 */
libname mycas cas;                                                    /* 3 */

data mycas.hmeq;                                                      /* 4 */
   set mylib.hmeq;
run;

proc partition data=mycas.hmeq samppct=10 seed=10 nthreads=1;          /* 5 */
   output out=mycas.out2 copyvars=(job reason loan value delinq derog);  /* 6 */
   display 'SRSFreq';
run;

proc print data=mycas.out2(obs=20);
run;
```

1. Create a CAS session and caslib.
2. Define the libref for the data. The source data is found in the path **/mySource/**.
3. Define the libref for the CAS engine.
4. Load the Hmeq data set to the caslib.

5 Perform a simple random sampling. The SAMPPCT=10 option requests that 10% of the input data be sampled.

6 The OUTPUT statement requests that the sampled data be stored in a table named mycas.out2. The COPYVARS= option lists the variables to be copied from mycas.hmeq to mycas.out2.

7 The DISPLAY statement requests that the SRSFreq ODS table be displayed.

Data Mining and Machine Learning Procedures

CAS procedures enable you to have the familiar experience of coding SAS procedures, but behind each procedure statement is one or more CAS actions that run across multiple machines.

For examples, see the individual procedures listed in the table.

Procedure	Description
ASTORE	Creates score code that can use code from an analytic store, as well as DS2 code. The procedure also has some analytic store management functions.
BOOLRULE	Enables you to extract Boolean rules from large-scale transactional data. The BOOLRULE procedure can automatically generate a set of Boolean rules by analyzing a text corpus that has been processed by the TEXTMINE procedure and is represented in a transactional format.
FACTMAC	Implements the factorization machine model. The flexible factorization machine model has applications in predictive modeling and recommendation. Video: Factorization Machines
FOREST	Creates a predictive model called a forest, which consists of several decision trees. A decision tree in a forest trains on new training data that is derived from the original training data that is presented to the FOREST procedure. Using different data to train different trees reduces the correlation of the predictions of the trees, which in turn should improve the predictions of the forest. Video: Auto-tuning: Forest Video: Ensemble Methods: Forest
GRADBOOST	Creates a predictive model called a gradient boosting model, which consists of multiple decision trees. The procedure samples the original data without replacement to create the training data for an individual tree and performs the action of sampling multiple times throughout a run. Each set of training data created is referred to as a subsample. The GRADBOOST procedure trains a decision tree by splitting the subsampled data, then splitting each resulting segment, and so on recursively until some constraint is met. Video: Ensemble Methods: Gradient Boosting

Procedure	Description
NETWORK	Provides a number of network analysis algorithms that use an abstract graph or network as input, help explain network structure, and compute important network measures. Depending on the application, this type of network analysis can stand on its own to provide independent value, or it can support machine learning models.
NNET	Trains a multilayer perceptron neural network. This procedure can also use a previously trained network to score a data table (referred to as stand-alone scoring), or it can generate SAS DATA step statements that can be used to score a data table. Video: Auto-tuning: Neural Network
SVMACHINE	Implements the support vector machines (SVM) algorithm. A popular data-mining area-classification method, the SVM algorithm computes support-vector machine-learning classifiers for the binary pattern recognition problem. This method has been broadly used in fields such as image classification, handwriting recognition, financial decision-making, and text mining.
TEXTMINE	Integrates natural language processing and statistical analysis to analyze large-scale textual data. Video: Text Mining
TMSCORE	Scores textual data. In text mining, scoring is the process of applying parsing and singular-value decomposition projections to new textual data. Video: Text Mining

This example recommends movies that are based on user ratings:

```
proc casutil;                                                    /*  1  */
   load file="~/data/mydata.csv"
   casout="movlens"                                              /*  2  */
   importoptions=(filetype="CSV" delimiter="TAB" getnames="FALSE"  /*  3  */
                  vars=("userid" "itemid" "rating" "timestamp"));
run;

proc factmac data=mycas.movlens nfactors=10 learnstep=0.15       /*  4  */
             maxiter=20  outmodel=mycas.factors;

   input userid itemid /level=nominal;
   target rating /level=interval;
   output out=mycas.out1 copyvars=(userid itemid rating);
run;

proc print data=mycas.factors(obs=10);                           /*  5  */
run;
```

1 Load the data.

2 Specify the name of the in-memory table.

3 Specify options for importing a CSV file and the variables to use in the table.

4 Use PROC FACTMAC to predict movie ratings.

5 Print the first 10 rows of the output table.

To see the results, see PROC FACTMAC.

Programming with CAS Actions

PROC CAS and the CAS Language (CASL)

PROC CAS enables you to interact with the CAS server from SAS by using the CAS language and CAS actions. PROC CAS and CASL are similar to the DATA step and PROC IML in that you can use functions and various types of statements, including control statements (DO, GOTO, IF-THEN, SELECT), to create a CAS program. You use CASL statements as statements in PROC CAS. PROC CAS provides built-in functions, or you can create your own functions.

This simple program loads the table iris.sashdat to the active caslib (default) to perform a correlation:

```
proc cas;                                                                  /* 1 */
   table.loadtable / path="iris.sashdat" casOut={name="IRIS"};             /* 2 */
   simple.correlation result=x
      / table={groupBy={"Species"}, name="IRIS", orderBy={"SepalLength"}};  /* 3 */
run;
print x;                                                                   /* 4 */
run;
```

1 The PROC CAS statement is always the first statement in the CASL language.

2 The table.loadtable action loads the table iris.sashdat to the in-memory table IRIS by using the active caslib. To define parameters, such as casOut, use {}.

3 The action simple.correlation measures the linear correlation between two variables. The groubBy parameter combines two or more data sets and enables you to group information in the output according to the values of one or more variables. The orderBy parameter specifies the columns by which to order the data within a partition. The results are placed in the result variable x. By placing the results in a variable, the results can be used later in the program by using the x variable.

4 The PRINT statement prints the output data sets from variable x. If, in step 3, the results were not assigned to variable x, the output would appear in SAS Studio on the **RESULTS** tab without the PRINT statement.

See Also

- "CAS" in *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*
- table.loadtable action

CAS Actions

CAS actions perform a single task. CAS actions are aggregated with other actions in an action set. Action sets usually contain actions that are based on common functionality. You access an action by using the notation *action-set.action*. For example, the Table action set contains all of the actions for working with tables, such as adding a table (table.addTable), modifying table attributes (table.attribute), loading data (table.loadDataSource), and dropping a table (table.dropTable).

CAS action sets exist for data mining and machine learning, statistics, general analytics, access control, DATA step, DS2. FedSQL, streaming data, session management, CAS server management, and table management.

In SAS, you submit CAS actions using PROC CAS and CAS language (CASL) statements. You can use the SAS open-source code to submit CAS actions using Python, Lua, Java, and REST APIs.

To begin learning about programming with CAS actions, start with *Getting Started with CASL*. Also, in *SAS Cloud Analytic Services: System Programming Guide*, review Programming Basics, Working with Results, and Working with By-Groups.

Review and run the PROC CAS examples to begin to understand how to run CAS actions. Most of the CAS action examples in the following documents have examples in CASL, Lua, and Python:

Document	Action Set Types
SAS Cloud Analytic Services: Data Mining and Machine Learning Programming Guide	Machine Learning Text mining
SAS Cloud Analytic Services: Statistics Programming Guide	Statistics
SAS Cloud Analytic Services: System Programming Guide	CAS System: Access control Server Properties Language: DATA Step, DS2, FedSQL, Transpose Streaming Data Session Methods Session Properties System Table Transpose
SAS Cloud Analytic Services: Analytics Programming Guide	CAS Analytics: Aggregate action Data preprocessing Analytics

This example investigates how the Major League Baseball (MLB) players' salaries depend on performance measures for the players' previous season and MLB career. To address a concern that some players who are outliers could dominate your least squares analysis, you can use the Quantreg action to obtain a median regression model:

```
cas mysess cassessopts=(caslib="casuser");                    /* 1 */

libname mycas cas;                                            /* 2 */

data mycas.baseball;                                          /* 3 */
   set sashelp.baseball;
run;

proc cas;                                                     /* 4 */
   quantreg.quantreg/                                         /* 5 */
      table={name='baseball'},
```

```
        class={'league','division'},
        model={depvars='Salary',
               effects={'nAtBat', 'nHits', 'nHome', 'nRuns',
                        'nRBI', 'nBB', 'yrMajor', 'crAtBat',
                        'crHits', 'crHome', 'crRuns', 'crRbi',
                        'crBB', 'league', 'division', 'nOuts',
                        'nAssts', 'nError'}};
run;
```

1. Use the MySess session and the CasUser caslib. Other examples in this document use a separate CASLIB statement. You can specify caslibs in options in the CAS statement and in the LIBNAME statement.
2. The LIBNAME statement binds the CAS engine libref, MyCas, to the active caslib, CasUser.
3. Load the data set sashelp.baseball to the in-memory CAS table Baseball.
4. CAS action programming using CASL begins with the PROC CAS statement.
5. Use the Quantreg action to fit a quantile regression model. Within action code, when you refer to a table in a caslib, you specify only the table name, not the libref. The class variables are League and Division, and the dependent variable is Salary.

To continue exploring this example and to view the example output, see the Quantreg action.

PROC CASUTIL

The CASUTIL procedure has several table and caslib management functions:

- display table metadata
- delete files from a data source
- unload a table from a caslib
- lists tables in a caslib
- load tables to a caslib
- promote a table to global scope
- save a table in a caslib to a data source

This example loads a CSV file in to CAS:

```
caslib csvfiles task=add type=dnfs                              /* 1 */
  path="/data/csv/"
  desc="Spreadsheets and CSV source data.";

proc casutil;
   list files;

   load casdata="County_Population.csv"                         /* 2 */
        importoptions=(filetype="csv" getnames="true")
        casout="county_population";

   list tables;
quit;
```

1 The TYPE=DNFS option in the CASLIB statement specifies a distributed Network File System (NFS) caslib type. This type requires every machine that is used for the server to have network access to the specified path. The CASLIB statement also sets the active caslib.

2 The IMPORTOPTIONS= option specifies the file type and options for reading the data into the server.

See Also

"CASUTIL" in *SAS Cloud Analytic Services: Language Reference*

SAS Procedures

SAS Viya includes some of the procedures that you are familiar with from previous releases of SAS. You must define a CAS libref for the library that the procedures that process CAS tables or CAS table metadata. Most of these procedures run in the SAS workspace and not on the CAS server. The data or metadata used by these procedures transfer data from the CAS server to SAS. Large tables might cause performance degradations.

Procedure	Description	Processes CAS Data
APPEND	Adds rows from a CAS table to the end of a SAS data set, and adds rows from a SAS data set to the end of a CAS table.	Yes
CONTENTS	Shows the contents of a CAS table and prints the directory of the caslib.	Yes
COPY	Copies tables to and from libraries.	Yes
DATASETS	Manages CAS tables.	Yes
DELETE	Deletes SAS data sets and CAS tables.	Yes
DS2	Runs DS2 code.	Yes
EXPORT	Writes a SAS data set to delimited files or to JMP files.	No SAS/ACCESS to PC Files must be purchased to add support for other file types, such as Microsoft Access databases for DBMS=ACCESS, Microsoft Excel workbooks, dBase database (DBF) files, and Lotus spreadsheets.
FEDSQL	Submits FedSQL code.	Yes, with limitations on page 28
FORMAT	Creates user-defined formats.	Yes PROC FORMAT stores user-defined formats in a CAS format library, which is associated with a CAS session. CAS user-defined formats must be associated with a variable before CAS processes a table.
HTTP	Processes data from the web.	No
IMPORT	Reads external data into a SAS data set.	No

Procedure	Description	Processes CAS Data
JAVAINFO	Shows information about the version of Java on your system.	No
OPTIONS	Lists the current settings of SAS system options in the SAS log.	No
PRINT	Prints SAS data sets and CAS tables.	Yes
PRINTTO	Redirects output.	No
PRODUCT_STATUS	Lists the SAS products that are installed on your system.	No
PWENCODE	Encodes passwords.	No
SGPANEL	Creates a panel of graph cells for the values of one or more classification variables.	Yes Graphing large tables can cause performance degradations, You can summarize or reduce large amounts of data before attempting to graph it. You can limit the number of rows to graph by using the OBSMAX= option to indicate the maximum number of rows to process.
SGPLOT	Creates statistical graphics such as histograms and regression plots, in addition to simple graphics such as scatter plots and line plots.	Yes Graphing large tables can cause performance degradations, You can summarize or reduce large amounts of data before attempting to graph it. You can limit the number of rows to graph by using the OBSMAX= option to indicate the maximum number of rows to process.
SGSCATTER	Creates a paneled graph of scatter plots for multiple combinations of variables.	Yes Graphing large tables can cause performance degradations, You can summarize or reduce large amounts of data before attempting to graph it. You can limit the number of rows to graph by using the OBSMAX= option to indicate the maximum number of rows to process.
SORT	Sorts SAS data sets	No
TRANSPOSE	Creates a CAS table by restructuring the values in a CAS table and transposing selected columns into rows	If the DATA= and OUT= options point to CAS, the transposition is performed within CAS by invoking the CAS Transpose action.

DATA Step

The DATA step can run in your SAS session as well as in the CAS server. In your SAS session, the DATA step runs on the workspace server. In CAS, the DATA step runs in a CAS server session on a single server (called symmetric multiprocessing, or SMP) or across multiple computers in parallel (called massively parallel processing, or MPP). In a compute cluster, the same DATA step program is replicated across the CAS cluster, where each DATA step thread runs on a portion of the data. By default, the DATA step runs in all available threads on every computer node in the cluster. For each node, the DATA step processes only the table rows that are allocated to that node.

You can use the DATA statement in your program or you can include your DATA step code by using the dataStep.runCode action in PROC CAS or in a supported third-party language.

To process in CAS, you load data to a caslib.

In this example, the LIBNAME statement for the CAS engine binds the MyCas library to the active caslib. Running in the SAS Workspace Server, the SET statement loads a subset of the Sashelp.Cars data set (cars where the weight is greater than 6000 pounds and five variables) to the active caslib.

```
cas cas auto;                              /* 1 */
libname mycas cas caslib=casuser;          /* 2 */
data mycas.cars (where=(weight>6000));     /* 3 */
   set sashelp.cars;
   keep make model type weight MPG_City;
run;
```

The following DATA step runs in the CAS server. When the input and output librefs use the CAS engine, the DATA step runs in the CAS server.

```
 data mycas.cars2;                         /* 4 */
   set mycas.cars;
   if mpg_city > 25 then eff='Y';
      else eff='N';                        /* 5 */
   put 'Thread number: '  _threadid_       /* 6 */
       'on worker node '  _hostname_;
run;
```

1. Start a session with the CAS server.
2. Create the libref MyCas and bind it to the CasUser caslib.
3. Load a subset of the sashelp.cars data set to the CasUser caslib.
4. Create the table cars2 in the CasUser caslib from the CAS table Cars.
5. Specify whether the cars in the cars2 table run efficiently.
6. Write the thread and worker node where the data was processed.

See "DATA Step Basics" in *SAS Cloud Analytic Services: Accessing and Manipulating Data* for an overview of running the DATA step in CAS.

Not all DATA step language elements that you used in previous releases of SAS are appropriate for distributed processing in the CAS environment. For a list of these language elements, see the category table in the respective documentation:

- Hash objects and Java objects

- Data Set Options
- Formats
- Functions
- Statements: DATA step and global statements

If the language element specifies a category of CAS in the documentation, it can run in the CAS server.

There are also some restrictions when you use the DATA step in the CAS server. For a list of these restrictions, see the following topics:

- "Restrictions for DATA Step Processing in CAS" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*
- "Restrictions for Using the VARCHAR Data Type in the DATA Step" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*

Using BY variables in a cluster, CAS distributes table data across multiple computers, or nodes. It divides the rows equally among all the available nodes in the cluster and distributes them to each node. For information about BY processing behavior in CAS, see the following topics:

- "How CAS Groups Data with BY Variables" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*
- "How BY Variables Affect Multithreaded DATA Step Execution" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*

The rules for CAS table variable names are different from rules for SAS data set variable names. CAS table variable name that are greater than 32 characters in length or that contain certain special characters must be used as an n-literal character. An example of using an n-literal name is `input 'Bob''s Asset Number'n;`.

A version of SAS prior to SAS Viya cannot run a SAS Viya DATA step stored program or view. SAS Viya can run a DATA step or stored program that was written for an earlier version of SAS.

Macro Language

Macros are supported in SAS Viya, but only in the SAS session. Macros can be useful in generating the code that runs in CAS, but the macro language itself is not supported in CAS.

DS2

DS2 runs in SAS Viya as it has in previous releases of SAS, with one exception. You cannot use FedSQL queries in DS2 code when you run in the CAS server.

In addition to using PROC DS2 to run DS2 code, you can also use the ds2.runDS2 action by using PROC CAS or a supported third-party language.

FedSQL

In CAS, for SQL operations, you can use PROC FEDSQL and the fedSql.execDirect action using PROC CAS or a supported third-party language.

You can submit SQL queries on SAS data sets and CAS tables, with some exceptions for support in CAS.

In CAS, FedSQL supports these statements:

- CREATE TABLE AS
- SELECT
- DROP TABLE

SAS data sets and CAS tables cannot be joined.

The SELECT operations have some limitations. These operations are not supported in CAS:

- SET operations
- correlated subqueries
- views
- DS2 UDFs
- SQL pass-through

PROC FEDSQL cannot save or promote CAS tables. You must use PROC CASUTIL to save or promote a FedSQL CAS table.

These PROC FEDSQL statements are supported for SAS data sets in a local SAS session:

- CREATE TABLE
- CREATE INDEX
- CREATE VIEW
- ALTER TABLE
- DESCRIBE VIEW
- DELETE
- DROP TABLE
- DROP INDEX
- DROP VIEW
- INSERT
- SELECT
- UPDATE

See Also

- "FedSQL Action Set Details" in *SAS Cloud Analytic Services: System Programming Guide*
- "CAS" in *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*
- "Dynamically Executing FedSQL Statements from DS2" in *SAS Viya: DS2 Programmer's Guide*

User-Defined Formats

You can store user-defined formats in SAS Viya in catalogs to use in a SAS session, or you can store them in a format library on the CAS server. Format libraries are associated with a CAS session, or they can be promoted to global scope to be available to all CAS sessions. These are server-side formats that the server uses when an analysis is performed according to formatted values.

You continue to use PROC FORMAT to create formats. A new option, CASFMTLIB, names the CAS format library for the active caslib where the format is stored. Without the CASFMTLIB= option, formats continue to be stored in a format catalog.

You must assign a format to a variable before the table is loaded into the server. After a table is in memory, you cannot assign a format. You can assign a format with the FORMAT statement in a DATA step or with the CASUTIL procedure.

This example adds and saves a user-defined format:

```
cas casauto sessopts=(caslib="casuser");

proc format library=work.formats casfmtlib="casformats";              /* 1 */
  value enginesize
    low - <2.7 = "Very economical"
    2.7 - <4.1 = "Small"
    4.1 - <5.5 = "Medium"
    5.5 - <6.9 = "Large"
    6.9 - high = "Very large"
;

cas casauto savefmtlib fmtlibname=casformats                          /* 2 */
   table=enginefmt replace;

proc casutil;
  format enginesize enginesize.;                                      /* 3 */
  load data=sashelp.cars casout="cars" replace;
  contents casdata="cars";
quit;
libname mycas cas;                                                    /* 4 */

proc mdsummary data=mycas.cars;
  var mpg_highway;
  groupby enginesize / out=mycas.mpg_hwy_by_size;                     /* 5 */
run;

proc print data=mycas.mpg_hwy_by_size;
  var enginesize--_mean_;
run;
```

1 The FORMAT procedure creates a format that is named Enginesize. On the SAS client, the format is temporary and is stored in the Work library. The CASFMTLIB= option adds the same format to your CAS session in a format library that is named Casformats.

2 The CAS statement is used with the SAVEFMTLIB option to persist the format as a SASHDAT file in the data source that is associated with the active caslib. The CasUser caslib is typically a PATH type and uses a file system. The TABLE= option specifies the name and results in a file that is named enginefmt.sashdat.

3 The FORMAT statement is used with the CASUTIL procedure to assign the Enginesize format to the Enginesize variable. The format is applied to the in-memory instance of the Cars table.

4 A CAS LIBNAME engine libref is assigned so that SAS procedures can access the Cars table that is in memory on the server. The assignment does not specify a caslib, so the active caslib is used.

5 The MDSUMMARY procedure provides descriptive statistics for the Cars table. The results are grouped by the formatted values of the Enginesize variable (five values) instead of by the numeric value. As a result, the output table, Mpg_hwy_by_size, has five rows.

Output in SAS Viya

The data mining and machine learning procedures create in-memory output tables that contain the procedure results. Some of these tables are used as input for scoring, and some are the results of scoring. Several of these procedures save their models as an analytic store. An analytic store is a binary object that contains that procedure's state after it completes the training phase of data analysis. You use an analytic store with the ASTORE procedure to score data at a later time.

By default, SAS Studio displays HTML output on the **RESULTS** tab. The **OUTPUT DATA** tab displays the table metadata if your program creates a CAS table. If you do not specify the casOut parameter in a CAS action, all results appear on the **RESULTS** tab.

The type and number of tables vary based on the code that you submit.

Here are some result tables.

The CASUTIL Procedure

Caslib Information	
Library	CASUSER( )
Source Type	PATH
Description	Personal File System Caslib
Path	
Session local	No
Active	Yes
Personal	Yes

The CASUTIL Procedure

Table Information for Caslib CASUSER( )

Table Name	Label	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
SOME_CARS	Some makes from the Sashelp.Cars sample data.	9	15	utf-8	26Jul2016:10:36:44	26Jul2016:10:36:44	Yes	No	No	No

The CASUTIL Procedure

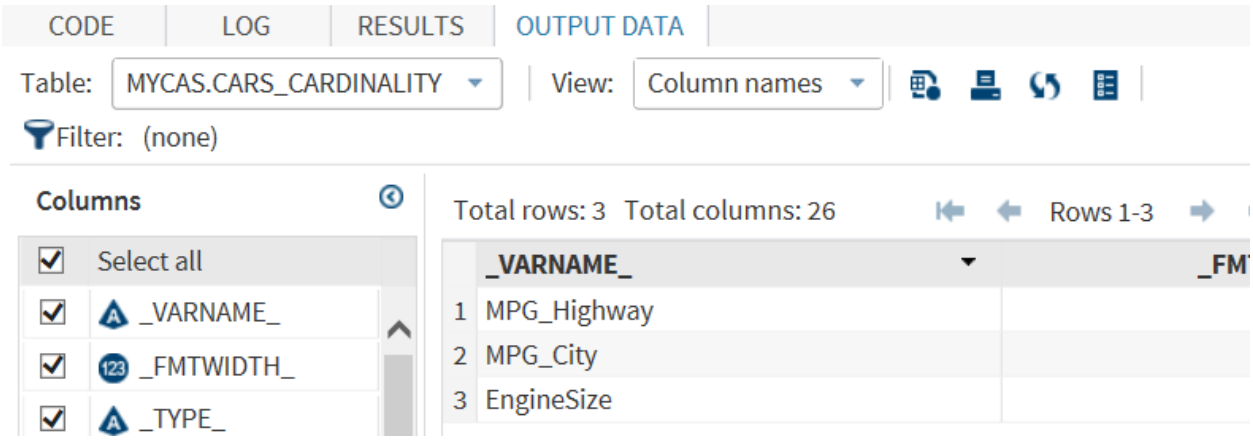
Table Information for Caslib CASUSER( )

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
CARS_CARDINALITY	3	26	utf-8	26Jul2016:10:36:45	26Jul2016:10:36:45	No	No	No	No

Detail Information for cars_cardinality in Caslib CASUSER( ).

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated
ALL	1	1	3	1080	0	0	0	0	0	1	1080

Column Information for CARS_CARDINALITY in Caslib CASUSER( )					
Column	Label	Type	Length	Formatted Length	Format
VARNAME	Variable name	char	32	32	$
FMTWIDTH	Width of the variable formatted value	double	8	12	BEST
TYPE	Type of the raw values	char	1	1	$
RLEVEL	Recommended level for analytics	char	10	10	$
ORDER	Variable sort order	char	8	8	$
MORE	Have more unreported levels	char	1	1	$
CARDINALITY	Number of levels	double	8	12	BEST
NOBS	Number of observations	double	8	12	BEST
SUMFREQS	Total summation of frequencies	double	8	12	BEST
NMISS	Number of missing values	double	8	12	BEST
MISSFMT	Format of the missing value	char	32	32	$
VISIBLE	Percentage of the visible part of the report	double	8	12	BEST
MIN	Minimum numeric value	double	8	12	BEST
MAX	Maximum numeric value	double	8	12	BEST
MEAN	Mean	double	8	12	BEST
STDDEV	Standard deviation	double	8	12	BEST
SKEWNESS	Skewness	double	8	12	BEST
KURTOSIS	Kurtosis	double	8	12	BEST
MFREQ	Maximum frequency	double	8	12	BEST
MFREQFOUNDLEVEL	Found maximum frequency in the visible part of the report	char	1	1	$
MFREQNUM	Numeric level with the maximum frequency	double	8	12	BEST
MFREQCHR	Character level with the maximum frequency	char	32	32	$
MFREQCFMT	Formatted level with the maximum frequency	char	32	32	$
LASTNUM	Last raw numeric value (cutoff)	double	8	12	BEST
LASTCHR	Last raw character value (cutoff)	char	32	32	$
LASTCFMT	Last formatted value (cutoff)	char	32	32	$



For quick access to HTML, PDF, and RTF output, use the first three icons on the **RESULTS** tab to download the files.

See Also

- *SAS Studio: User's Guide*