SAS® Cloud Analytic Services 3.3: User's Guide

SAS® Documentation
June 23, 2020

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2017. *SAS® Cloud Analytic Services 3.3: User's Guide*. Cary, NC: SAS Institute Inc.

SAS® Cloud Analytic Services 3.3: User's Guide

SAS Institute Inc., SAS Campus Drive, Cary, NC 27513-2414

June 2020

3.3-P2:casref

Contents

About This Book

Audience

This book supports the use of SAS Cloud Analytic Services by participating SAS solutions. It explains key concepts and provides essential instructions. The emphasis is on server operation, data management, and security configuration.

What's New in 9.4 for SAS Cloud Analytic Services

Overview

For an introduction to programming on the CAS server, see *An Introduction to SAS Viya Programming*.

In SAS 9.4M5, SAS has been enhanced to provide access to CAS.

- New options have been added to the CAS statement to connect to the SAS Metadata Server.
- The CONTENTS statement for the CASUTIL procedure is enhanced to provide information about column index size, compression ratio, and compressed data size.
- New functions have been added to provide information about your CAS session.
- New system options have been added to control data transfer between SAS and CAS.
- A new macro has been added to migrate user-defined formats from SAS catalogs to CAS format libraries.

Enhanced CAS Statement

The following options are new in SAS 9.4M5:

- The AUTHDOMAIN= option specifies the name of an authentication domain object registered on the SAS Metadata Server that associates user credentials with an identity.
- The CASSERVERMD= option specifies the name of a server object registered on the SAS Metadata Server that associates SAS Cloud Analytic Services connection parameters with a server name.

The following options are new in SAS Viya 3.3:

- The AUTHDOMAIN=_SASMETA_ option requests that the SAS Metadata Server generate a one-time password that can be used to access CAS. This option enables stored processes, token-based workspace servers, and pooled workspace servers to connect to CAS without the need for stored user credentials.
- The HOST= option specifies the machine name for the primary control node of the server and the backup control node, if configured. If the primary control node fails and a backup controller is available, all of the sessions connect to the backup controller automatically.

- The LOADFORMATS= option loads formats that are stored in an item-store file into a format library in the active session. The format item-store file is created using the FMTC2ITM procedure. The LOADFORMATS= option and the FMTC2ITM procedure are useful for migrating user-defined formats from SAS to CAS.

For more information, see "CAS Statement".

Enhancements to the CASUTIL Procedure

The results of the CONTENTS statement are enhanced to include the index size, compressed size, and compression ratio.

Functions

The following functions are new in SAS 9.4M5:

- CLIBEXIST on page 273indicates whether a caslib name exists.
- GETCASURL on page 274 returns the CAS Server Monitor URL.
- GETLCASLIB on page 274 returns the caslib for a CAS LIBNAME engine libref that was specified by the CASLIB= LIBNAME option.
- GETLSESSREF returns the session reference for a CAS LIBNAME engine libref.
- GETLTAG returns the tag for a CAS LIBNAME engine libref that was specified by the TAB= LIBNAME option.
- GETSESSOPT returns the value of a CAS server session option.
- SESSFOUND returns a value to indicate whether a named session that you started in your SAS session is found.

System Options

The following system option is enhanced in SAS Viya 3.3:

- For a fault tolerant environment, you can specify the backup host controller using the "CASHOST= System Option" on page 299. If the primary controller fails, sessions are automatically connected to the backup host controller.

The following system options are new in SAS 9.4M5:

- Data transfer of CAS tables can impede system performance if the CAS table is very large. You can use the CASDATALIMIT= system option to limit the amount of data in a single CAS table that can be transferred from the CAS server to SAS. By default, the amount of data that can be read from a CAS table is 100M.
- When SAS writes data to the CAS server using the CAS engine, the engine estimates the number of bytes that are needed to transcode the data to UTF-8 based on the character set of the SAS session. The character set is specified by the ENCODING= system option. SBCS environments estimate one byte in UTF-8 for every one byte in

the local encoding. DBCS environments estimate 1.5 bytes in UTF-8 for every one byte in the local encoding. You can use the CASNCHARMULTIPLIER= system option to replace the estimate with an explicit value of the byte multiplier when you know the number of bytes that are needed to represent the data in UTF-8.

- By default, the DSCAS system option on page 308 is set so that the DATA step runs on your CAS server without specifying a session reference option in the DATA statement. If NODSCAS is set, the SESSREF= DATA statement option is required for the DATA step to run on the CAS server. To run the DATA step in SAS, you would set the NODSCAS option.

Macros

To use your SAS user-defined formats in the CAS server, you can migrate them from a SAS catalog to the CAS server using the autocall macro %UDFSEL. This macro generates a SELECT statement that you can use with the FORMAT procedure to migrate only the user-defined formats that your data is using. User-defined formats are stored in a CAS library in the CAS server. For more information, see %UDFSEL Autocall Macro .

Data Connectors

The following data connectors are new in SAS Viya 3.3:

- SAS Data Connector to Amazon Redshift
- SAS Data Connector to DB2 for UNIX and PC Hosts
- SAS Data Connector to Microsoft SQL Server
- SAS Data Connector to SAP HANA
- SAS Data Connector to SPD Engine Files and SAS Data Connect Accelerator for SPD Engine Files

The following enhancements have been made in SAS Viya 3.3 for all data connectors:

- ability to save CAS tables to your data source
- ability to delete a table in your data source from CAS
- support for multi-node data transfer when DATATRANSFERMODE="SERIAL"
- support for INTEGER data types

Chapter 1

Accessing Data

Introduction

SAS Viya analytical procedures and some DATA step features use input data from in-memory tables on SAS Cloud Analytic Services only. This document takes you through common tasks for loading and accessing your input data with SAS Cloud Analytic Services. The methods that these tasks use for loading and accessing are:

- caslibs
- CASUTIL procedure
- DATA step

You can use SAS/CONNECT to transfer remote data sources directly into in-memory tables. However, SAS/CONNECT is a separately licensed product. For more information about SAS/CONNECT, see *SAS/CONNECT User's Guide*.

Terms to Be Familiar With

Before we begin, here are a few important terms:

active caslib
: your session must have a default location for server-side data access. This is the active caslib. The term "active caslib" is used rather than default caslib because the caslib that your session uses is modified as caslibs are added and dropped.

caslib
: the mechanism for accessing data with SAS Cloud Analytic Services. At its simplest, a caslib provides access to files in a data source, such as a database or file system directory, and to in-memory tables.

data connector
: a data connector is the software that is used with a caslib to read server-based data sources like databases and Hive. There are a few data connectors for file-based caslibs. These data connectors are used to control reading data files such as setting the file encoding.

file
: the source data that is in a caslib's data source. For a caslib that uses a path-based data source, this is natural. For a caslib that uses a database as a data source, the tables in the database are referred to as files.

session
: when you initially connect to SAS Cloud Analytic Services, your session is started on the server. Data access and communication is performed through the session. Your programs communicate with the session to request actions. Many sessions can operate concurrently, actions execute serially within a session. In most cases, programmers start and use one session only.

table
: is used to refer to in-memory data. After a file (using the preceding definition) is loaded into the server, it is referred to as a table.

Common Tasks for Accessing and Manipulating Data

Table 1.1 *Common Tasks for Accessing Data*

Task	Sample Syntax

Load a SAS data set.
Tip: This is a good learning step if you are familiar with SAS and want to learn how SAS libraries, data sets, and SAS Cloud Analytic Services work together.
	`proc casutil;`
	`  load data=`*`libref.member-name`*
	`  casout="`*`table-name`*`";`
	`run;`

Load a client-side data file.	`proc casutil;`
	`  load file="`*`/path/to/file.suffix`*`"`
	`  casout="`*`table-name`*`";`
	`run;`

List caslibs. This shows you the server-side data sources that SAS Cloud Analytic Services can access.	`caslib _all_ list;`
Add a file-based caslib. *Tip:* Remember that the specified PATH= must be accessible from the host for the SAS Cloud Analytic Services controller.	`caslib data datasource=(srctype="path")` `  path="/data01";`

Determine the data files in a caslib that the server can access.	`proc casutil;`
	`  list files incaslib="`*`name`*`";`
	`run;`

Load a server-side data file.	`proc casutil;`
	`  load casdata="`*`file-name.suffix`*`"`
	`  casout="`*`table-name`*`";`
	`run;`

Load a table from database.	`proc casutil;`
	`  load casdata="`*`database-table-name`*`"`
	`  casout="`*`table-name`*`";`
	`run;`

List in-memory tables.	`proc casutil;`
	`  list tables incaslib="`*`name`*`";`
	`run;`

Assign a CAS engine libref and bind it to a caslib. This is how you access tables with a SAS procedure or the DATA step.	`libname mycas cas caslib="`*`name`*`";`

Load a Client-Side File

Prerequisites

The following example assumes that you have a Microsoft Excel file. The sample code assumes that a file named **/data/WorldData.xlsx** is available in the **Server Files and Folders** section of SAS Studio.

Example

This example shows how to load data from a SAS library and how to load data from a Microsoft Excel file. This approach is appropriate for smaller tables and ad hoc data analysis.

```
caslib hps datasource=(srctype="hdfs")                                  /* 1 */
       path="/hps";

proc casutil incaslib="hps" outcaslib="hps";                            /* 2 */

   load data=sashelp.iris promote;                                      /* 3 */

   contents casdata="iris";

   load file="/data/WorldData.xslx" casout="worlddata";                 /* 4 */

   contents casdata="worlddata";
run;
```

1 Add a caslib to access the **/hps** directory in HDFS. By default, adding a caslib sets it as the active caslib.

2 The CASUTIL procedure statement includes the INCASLIB= and OUTCASLIB= options. This is a best practice to ensure that tables are read from the caslib that you expect and are saved to the caslib that you expect.

3 The DATA= argument indicates that the table is transferred from the SAS client host to SAS Cloud Analytic Services. Replace the Sashelp.Iris value with a libref and table that you want to use. The PROMOTE option makes the Iris table a global-scope table and available to other sessions that use the Hps caslib. Use the CONTENTS statement to ensure that the table includes the column names and data types that you expect.

4 The FILE= argument indicates that the file is a client-side file that is accessible to SAS and not to SAS Cloud Analytic Services. The file, WorldData.xlsx, is transferred to the server and then imported with a table name of Worlddata. This LOAD statement does not include the PROMOTE option, so the in-memory Worlddata table can be accessed only from the same session.

Key Ideas

- The LOAD DATA= and LOAD FILE= statements in the CASUTIL procedure are used for accessing client-side data.
- The CONTENTS statement is used to display information such as column names and data types.
- By default, when you add a caslib, that caslib becomes the active caslib. Use the NOTACTIVE option to add a caslib without making it active.

See Also

- Chapter 5, "CASUTIL Procedure," on page 110
- "CASLIB Statement" on page 91

Load a Server-Side File

Prerequisites

The following example assumes the following:

- You can create a small CSV file in the file system that is associated with the directory for your personal caslib.
- The Microsoft Excel file used in this example has historical consumer price index (CPI) data for food from http://www.ers.usda.gov/data-products/food-price-outlook.aspx.

Example

This example shows how to access two server-side files and load the data into CAS:

- a CSV file that describes the performance of a toy catapult. The first line of the file does not contain column names. The example shows how to specify names.
- a Microsoft Excel file.

For the CSV file, a description for the data is shown in the example. The values are as follows:

```
5,10,10,11,10,11,3
5.5,16,3,16,1,15,6
6,23,0,18,7,20,5
6.5,23,3,28,6,26,0
7,27,3,25,10,23,1
```

The program is as follows:

```
cas casauto sessopts=(caslib="casuser");                    /* 1 */
```

```
libname mycas cas caslib="casuser";                                          /* 2 */

/* first, load the data from the CSV file */
proc casutil incaslib="casuser" outcaslib="casuser";

  contents casdata="catapult.csv";                                           /* 3 */

  load casdata="catapult.csv" casout="catapultraw"
       importoptions=(filetype="csv"                                         /* 4 */
          encoding="latin1"
          getnames="false"
          vars(
            (name="turns",     label="Number of turns",       type="double"),
            (name="first_ft",  label="Feet for first try",    type="double"),
            (name="first_in",  label="Inches for first try",  type="double"),
            (name="second_ft", label="Feet for second try",   type="double"),
            (name="second_in", label="Inches for second try", type="double"),
            (name="third_ft",  label="Feet for third try",    type="double"),
            (name="third_in",  label="Inches for third try",  type="double")
          )
       ) replace;

    save casdata="catapultraw" replace;                                      /* 5 */
    contents casdata="catapultraw.sashdat";
quit;

data mycas.catapult (promote=yes) / sessref=casauto;                         /* 6 */
   set mycas.catapultraw;
   first  = 12 * first_ft  + first_in;
   second = 12 * second_ft + second_in;
   third  = 12 * third_ft  + third_in;
run;

proc casutil incaslib="casuser";
   contents casdata="catapult";                                              /* 7 */
   droptable casdata="catapultraw";                                          /* 8 */
quit;

/* simple scatter plot */
proc sgplot data=mycas.catapult;                                             /* 9 */
     scatter x=turns y=first;
     scatter x=turns y=second;
     scatter x=turns y=third;

     /*--X Axis--*/
     xaxis grid label="Number of turns";

     /*--Y Axis--*/
     yaxis grid label="Distance, in inches";;
run;

/* second, load the Excel file */
proc casutil incaslib="casuser" outcaslib="casuser";
   list files;                                                               /* 10 */
   contents casdata="historicalcpi.xls";
```

```
   load casdata="historicalcpi.xls" casout="historicalcpi"            /* 11 */
        label="Historical CPI data, 1974 through 2014; updated 3/3/2015";

   save casdata="historicalcpi" replace;
quit;
```

1 The SESSOPTS= option is used with the CASLIB= session option to ensure that the Casuser personal caslib is set as the active caslib.

2 The CAS engine LIBNAME statement assigns the Mycas libref and binds it to the Casuser caslib.

3 The CONTENTS statement shows the file information and column information for the CSV file. See Figure 1.1 on page 8.

4 The CASDATA= argument indicates that the file is read from the caslib's data source. The IMPORTOPTIONS= specify how to read the file.

5 The SAVE statement makes a copy of the imported data as a SASHDAT file. This is part of the data life cycle. If the file is imported correctly, then subsequent analyses of the data can begin from the SASHDAT file. The CONTENTS statement shows that the column names and labels are applied. See Figure 1.2 on page 8.

6 The DATA step is used to combine each set of foot and inch measures into a single column. The PROMOTE= option is used to make the table available to other sessions that you start. The SESSREF= option is used to ensure that the DATA step runs in CAS.

7 The last CONTENTS statement is used to display the table information, table details, and column information for the in-memory table. See Figure 1.3 on page 9.

8 The DROPTABLE statement is used to free the memory resources that are used for the data from the CSV file. The copy of the data that was made with the SAVE statement is not deleted, only the in-memory resources are freed.

9 The SGPLOT procedure uses the Mycas CAS engine libref. When Mycas was assigned at the start of the program, the CASLIB= option bound it to the Casuser caslib. This ensures that the libref always accesses tables in that caslib.

10 The LIST FILES statement is used to list the files in the caslib's data source. In this case, the personal caslib, Casuser, uses the OS file system. See Figure 1.4 on page 9.

11 The LOAD CASDATA= argument specifies the Historicalcpi.xls file. The LABEL= option is used to specify a description of the data.

Results

The following display shows the results of the CONTENTS statement. Notice that the anticipated column names match the first line of the CSV file. This is corrected in the subsequent LOAD CASDATA= statement when the GETNAMES= option is set to false.

Figure 1.1 *CONTENTS Statement Results for the CSV File*

The CASUTIL Procedure

File Information for catapult.csv in caslib CASUSER([redacted]).

Name	Permission	Owner	Group	File Size	Last Modified
catapult.csv	-rw-r--r--	[redacted]	[redacted]	0.1KB	01Feb2016:14:50:35

Column Information for catapult.csv in Caslib CASUSER([redacted])

Column	Type	Length	Formatted Length
5	double	8	12
10	double	8	12
10	double	8	12
11	double	8	12
10	double	8	12
11	double	8	12
3	double	8	12

The following display applies to the second CONTENTS statement, after the LOAD CASDATA= statement that include the IMPORTOPTIONS= settings. Notice that the column names and labels are applied.

Figure 1.2 *CONTENTS Statement Results for the SASHDAT File*

File Information for catapultraw.sashdat in caslib CASUSER([redacted]).

Name	Permission	Owner	Group	File Size	Last Modified
catapultraw.sashdat	-rwxr-xr-x	[redacted]	[redacted]	24.2KB	01Feb2016:19:27:28

Column Information for catapultraw.sashdat in Caslib CASUSER([redacted])

Column	Label	Type	Length	Formatted Length
turns	Number of turns	double	8	12
first_ft	Feet for first try	double	8	12
first_in	Inches for first try	double	8	12
second_ft	Feet for second try	double	8	12
second_in	Inches for second try	double	8	12
third_ft	Feet for third try	double	8	12
third_in	Inches for third try	double	8	12

Figure 1.3 *CONTENTS Statement Results for the In-Memory Table*

Table Information for Caslib CASUSER([redacted])

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Duplicated Rows	View	Source Name	Compressed
CATAPULT	5	10	utf-8	01Feb2016:19:28:25	01Feb2016:19:27:56	Yes	No	No	_T_1AAD1589_7FF56342BBA8	No

Detail Information for catapult in Caslib CASUSER([redacted]).

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	5	5	5	400	0	0	0	5	400	0	0

Column Information for CATAPULT in Caslib CASUSER([redacted])

Column	Label	Type	Length	Formatted Length
turns	Number of turns	double	8	12
first_ft	Feet for first try	double	8	12
first_in	Inches for first try	double	8	12
second_ft	Feet for second try	double	8	12
second_in	Inches for second try	double	8	12
third_ft	Feet for third try	double	8	12
third_in	Inches for third try	double	8	12
first		double	8	12
second		double	8	12
third		double	8	12

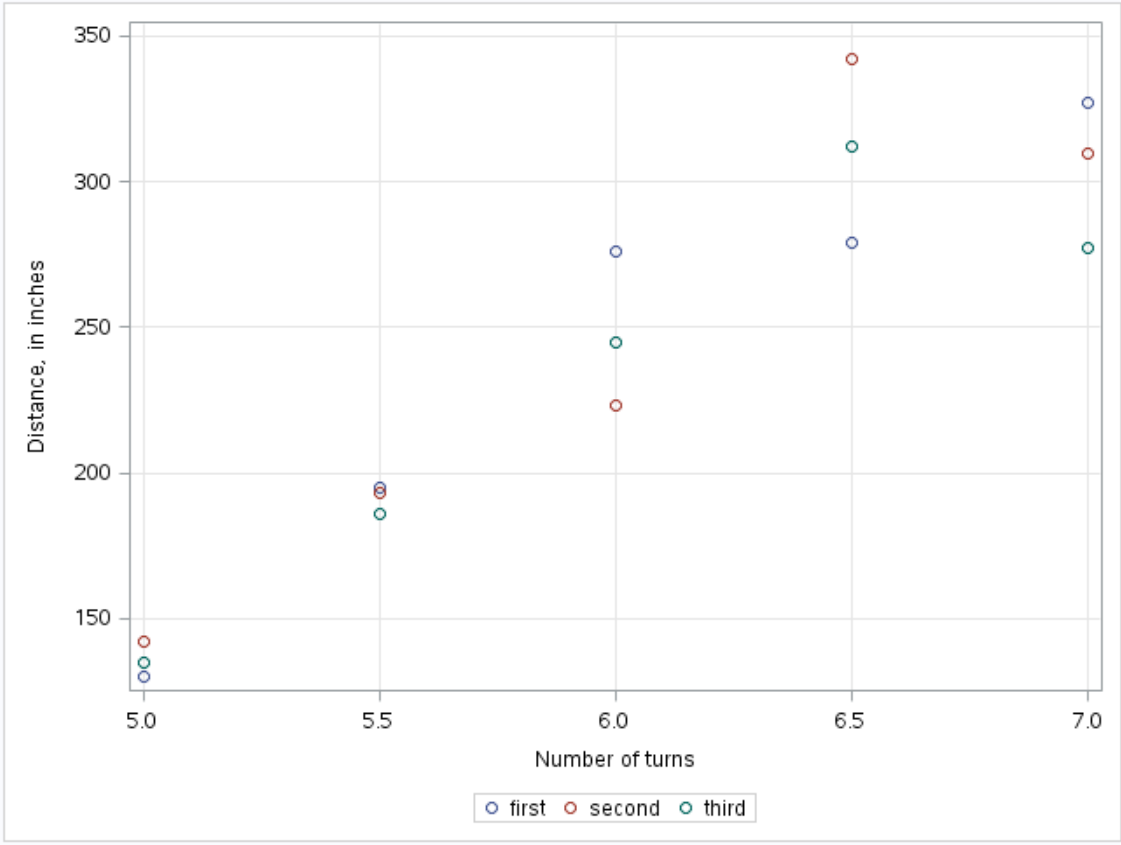
Figure 1.4 *LIST FILES Statement Results for a Path-Based Caslib*

The CASUTIL Procedure

File Information for root of caslib CASUSER([redacted]).

Name	Permission	Owner	Group	File Size	Last Modified
catapult.csv	-rw-r--r--	[redacted]	[redacted]	0.1KB	01Feb2016:14:50:35
historicalcpi.xls	-rwxr-xr-x	[redacted]	[redacted]	56.5KB	02Feb2016:19:08:19
mycas.distinct.sashdat	-rwxr-xr-x	[redacted]	[redacted]	7.2KB	12Nov2015:16:48:12
cars_large_part.sashdat	-rwxr-xr-x	[redacted]	[redacted]	20.4MB	26Jan2016:12:44:45

Figure 1.5 *Scatter Plot Results for the Catapult Table*



The following display shows the results of the CONTENTS statement for the Historicalcpi.xls file. By default, the column names are read from a file.

Figure 1.6 *CONTENTS Statement Results for the XLS File*

The CASUTIL Procedure

File Information for historicalcpi.xls in caslib CASUSER([redacted]).					
Name	**Permission**	**Owner**	**Group**	**File Size**	**Last Modified**
historicalcpi.xls	-rwxr-xr-x	[redacted]	[redacted]	56.5KB	02Feb2016:19:08:19

Column Information for historicalcpi.xls in Caslib CASUSER([redacted])				
Column	**Type**	**Length**	**Formatted Length**	**Format**
Consumer price indexes historical data, 1974 through 2014	varchar	0	0	$
B	varchar	0	0	$
C	varchar	0	0	$
D	varchar	0	0	$
E	varchar	0	0	$
F				
			0	$
	varchar	0	0	$
AO	varchar	0	0	$
AP	varchar	0	0	$
AQ	varchar	0	0	$
AR	varchar	0	0	$
AS	varchar	0	0	$
AT	varchar	0	0	$

Key Ideas

- Reading files from a caslib's data source is the most efficient way to access data. One key to recognizing that data is read from a caslib's data source is the presence of the CASDATA= argument.
- The CONTENTS statement in the CASUTIL procedure can display information for files, tables, and columns. To view information about a file, specify the filename, including the suffix in the CASDATA= option. After the table is loaded into memory, you drop the suffix or use the table name that you specified in the CASOUT= option.
- You can specify IMPORTOPTIONS= to describe how to load the data. For delimited files, the common options are to specify the file encoding and whether to get column names from the first line of the file.
- After the data for a file is imported, save a copy as a SASHDAT file.

See Also

- Chapter 5, "CASUTIL Procedure," on page 110
- "CASLIB Statement" on page 91
- "DATA Step Basics" in *SAS Cloud Analytic Services: DATA Step Programming*

Load a Database Table

Prerequisites

The following example assumes the following:

- You are granted access to data in the data source.
- You know the connection information such as host, port, and so on.
- Your SAS Cloud Analytic Services installation is licensed and configured to use the client software for the data source vendor that you want to access. For installation-time configuration information, see *SAS Viya for Linux: Deployment Guide*.

Example

Loading tables into the server from a caslib's data source is the most efficient way to load data. In this example, a table is read from Oracle and the in-memory table is kept in the same caslib.

```
caslib oralib datasource=(                                          /* 1 */
    srctype="oracle",
    uid="DBUSER",
    pwd="secret",
    path="//dbserver.example.com:1521/dbname",
    schema="DBUSER"
    );


proc casutil;
   list files;                                                      /* 2 */

   droptable casdata="sales" quiet;                                 /* 3 */
   contents casdata="sales";                                        /* 4 */

   load casdata="sales" casout="sales" promote                      /* 5 */
       label="Fact table for User-to-Item Analysis"
       varlist=(
            (name="USERID"  label="User ID"),
            (name="ITEMID"  label="Item ID")
       );

   contents casdata="sales";                                        /* 6 */
quit;
```

```
run;
```

1 Add a caslib that uses Oracle as the data source. Oralib becomes the active caslib for the session and the subsequent programming statements use it for input and output.

2 The LIST FILES statement displays the tables that are available in the Oracle database.

3 The DROPTABLE statement includes the QUIET option. Running this statement is useful on repeated runs because it ensures that no table named Sales can be in-memory to interfere with the subsequent LOAD CASDATA= statement.

4 Because the first CONTENTS statement follows the DROPTABLE statement, this ensures that the table information and column information from Oracle are read.

5 The CASDATA= argument in the LOAD statement indicates that the Sales table is read from the caslib's data source (Oracle) into SAS Cloud Analytic Services. Options are specified to add labels to the table and columns.

6 Because the last CONTENTS statement follows the LOAD statement, table information and column information is displayed for the in-memory copy of the Sales table that was read from Oracle.

Key Ideas

- The CASLIB on page 91 statement adds a server-side data source to SAS Cloud Analytic Services.
- In this example, the active caslib is Oralib. Remember that when you add a caslib, by default, it becomes the active caslib.
- For information about data source connection parameters, see Chapter 9, “Data Connectors,” on page 167.

Save an In-Memory Table

Example

This example demonstrates the following:

- saving a table to caslib named Hps that uses HDFS as a data source. The table is saved as a SASHDAT file.
- saving a table from a caslib that uses Oracle to a caslib named Hps that uses HDFS as a data source. The table is saved as a SASHDAT file.

```
proc casutil incaslib="hps" outcaslib="hps";
   load casdata="customers.sashdat" casout="customers";
run;

/* From some other application, or a DATA step, the */
/* Customers table is modified with a change that   */
/* is important to save.                            */
proc casutil  incaslib="hps" outcaslib="hps";
```

```
   save casdata="customers" replace;                                    /* 1 */
run;

caslib orsales datasource=(
   srctype="oracle"
   username="DBUSER"
   password="secret"
   path="//dbserver.example.com:1521/sales"
   schema="DBUSER"
   );

proc casutil incaslib="orsales";
   load casdata="sales" casout="sales";                                 /* 2 */

   save casdata="sales" outcaslib="hps";
run;
```

1 The SAVE statement uses the Hps caslib from the OUTCASLIB= procedure statement option. The next time a LOAD statement is used with the Customers table, the table includes the changes. You can use the CASOUT= option to specify an alternative filename other than the default, customers.sashdat.

2 The LOAD statement reads the table named Sales from the Orsales caslib that uses Oracle Database as a data source. To save a copy of the table in HDFS, the SAVE statement uses the OUTCASLIB= option to specify the Hps caslib that uses HDFS as a data source.

Key Ideas

- You can save in-memory tables as SASHDAT files or CSV files in path-based caslibs only. If you need to save a table from a caslib that is not path-based, then you can specify the OUTCASLIB= option.
- For information about data source connection parameters to use in the CASLIB statement, see Chapter 9, “Data Connectors,” on page 167.

Drop an In-Memory Table

Example

When you drop an in-memory table, only the in-memory table is affected. If the table was loaded from a caslib's data source, the table in the data source is unaffected.

```
proc casutil incaslib="hps" outcaslib="hps";
   load casdata="sales.sashdat" casout="sales";                         /* 1 */
run;

proc casutil incaslib="hps";
   droptable casdata="sales";                                           /* 2 */
run;
```

1 The LOAD statement loads a file named Sales.sashdat from the Hps caslib.

2 The DROPTABLE statement drops the in-memory table.

Key Ideas

- If you drop a session-scope table, then only the session that loaded the table is affected.
- If you drop a global-scope table, then the table might be accessed from multiple sessions. The table is dropped after any actions that access the table are complete.
- Be aware that dropping a global-scope table can affect other sessions if the actions that are run by other sessions expect the table to be in memory.

Delete a File from a Caslib's Data Source

Example

When you delete a file from a data source, it does not affect an in-memory copy. The term "file" refers to a file in a caslib with a path-based data source or a table in a caslib with a server-based data source.

The following example demonstrates deleting a file named Sales from the data source associated with the Hps caslib.

```
proc casutil;
   deletesource casdata="sales.sashdat" incaslib="hps";
run;
```

Key Ideas

- Include the filename suffix in the CASDATA= argument.

Data Compression

Overview of Data Compression

SAS Cloud Analytic Services supports and performs all compression for in-memory tables. When you transfer a table to the server and request compression, rows are sent to the server as is and the server compresses them.

All data in a row, both character and numeric variables, are compressed. Every row in a table is compressed. The server does not support some rows in compressed form and others as uncompressed.

For matrices of computed doubles (those with many decimal places), compression might not reduce the storage requirements at all. For rows with many long character variables that consist mostly of blanks, the compression ratio can be very high. For rows with mixed variables, where most doubles do not have fractional parts and most character variables have a small amount of blank padding, the compression ratio is typically moderate. As with most cases of using compression, character variables tend to compress the most and the ratio depends on your data.

Compressed Tables and the DATA Step

This example shows how to use the COMPRESS= data set option for SAS Cloud Analytic Services.

Example Code 1 *Creating a Compressed Table with a DATA Step*

```
libname mycas cas host="cloud.example.com" port=5570;

data mycas.prdsale (compress=yes);
    set sashelp.prdsale;
run;
```

After the table is loaded into memory, you can access the compressed table with the mycas.prdsale table reference.

SAS Cloud Analytic Services supports the APPEND= data set option for compressed tables. This example shows how to add new (uncompressed) rows to the compressed table.

Example Code 2 *Appending Rows to a Compressed Table*

```
data mycas.prdsale(append=yes);
    somelib.newrows;
run;
```

Because the mycas.prdsale table is already compressed, the new rows are automatically compressed as they are appended to the table. Specifying COMPRESS= with APPEND= has no effect. If the table is compressed, the server compresses the new rows. If the table is not compressed, then the server does not compress the new rows even if you specify COMPRESS=YES. The compressed or uncompressed state of the table determines how rows are appended.

Partitioning and compression are supported together. This example creates a new in-memory table that is partitioned and compressed.

Example Code 3 *Creating a Partitioned and Compressed Table*

```
data mycas.iris (partition=(species) compress=yes);
    set sashelp.iris;
run;

data mycas.iris (append=yes);
    set somelib.moreirises;
run;
```

In the first DATA step statement, the Iris data set is loaded into memory on the server. The table is partitioned by the formatted values of the Species variable. The table is also compressed. In the second DATA step statement, the table is appended to with more rows. Because the in-memory table is already partitioned and compressed, the new rows are automatically partitioned and compressed when they are appended.

Compressed Tables and the CASUTIL Procedure

You can use the CASUTIL procedure to load data in memory on SAS Cloud Analytic Services.

```
proc casutil;
   load data=sashelp.prdsale casout="prdsale" compress;
quit;
```

This example uses the COMPRESS option on page 76 to read the Prdsale data set from the sashelp library and compress it in-memory on the server. Be aware that you must specify the COMPRESS option for each table that you want to load in compressed form.

When you read SASHDAT tables into memory, compression depends on these factors for the resulting in-memory tables:

- whether a WHERE clause is used
- whether the SASHDAT table is compressed on disk

If you specify a WHERE clause when loading a SASHDAT file with compression, the server uncompresses the rows as it evaluates the WHERE clause. This results in an uncompressed in-memory table. The memory efficiencies of the SASHDAT table format are forfeited in this scenario because the server had to apply the WHERE clause.

If you do not specify a WHERE clause, the server ignores the COMPRESS option and relies on whether the SASHDAT file is compressed. If the SASHDAT file is compressed, the in-memory representation of the table is also compressed. If the SASHDAT file is not compressed, then neither is the in-memory representation. The server ignores the option so that it can keep the memory efficiencies of the SASHDAT file format: When a SASHDAT table is loaded in memory, the in-memory representation is identical to the on-disk representation.

Performance Considerations

Compression exchanges less memory use for more CPU use. It slows down any request that processes the data. An in-memory table consists of blocks of rows. When the server works with a compressed table, the blocks of rows must be uncompressed before the server can work with the variables. In some cases, a request can require five times longer to run with a compressed table rather than an uncompressed table.

For example, if you want to summarize two variables in a table that has 100 variables, all 100 columns must be uncompressed in order to locate the data for the two variables of interest. If you specify a WHERE clause, then the server must uncompress the data before the WHERE clause can be applied. Like the example where only two of 100 variables are used, if the WHERE clause is very restrictive, then there is a substantial performance penalty to filter out most of the rows.

Working with SASHDAT tables that are loaded from HDFS is the most memory-efficient way to use the server. Using compressed SASHDAT tables preserves the memory efficiencies, but still incurs the performance penalty of uncompressing the rows as the server operates on each row.

Interactions

Here are the interactions for compressed tables and SAS programs.

- You can use a compressed table in programs like any other table.

- You can define calculated columns for compressed tables with the TEMPNAMES= data set option.
- You can append to compressed tables. This is also supported for compressed tables that have partitioning with or without ordering.

See Also

- "APPEND= Data Set Option"
- "COMPRESS= Data Set Option"
- "ORDERBY= Data Set Option"
- "TEMPNAMES= Data Set Option"

Chapter 2

CAS Statement

Dictionary

CAS Statement

Starts and manages your SAS Cloud Analytic Services session.

Syntax

CAS *session-name <option(s)>*;

Summary of Optional Arguments

Authentication option

AUTHDOMAIN=*authentication-domain* | "*authentication-domain*" | _SASMETA_ | "_SASMETA_"
: obtains credentials from the SAS Metadata Server or the SAS Viya credentials service.

AUTHINFO="*authentication-info-file*"
: specifies an authinfo file or netrc file that includes authentication information.

CAS server information options

CASSERVERMD=*server-name* | "*server-name*"
: specifies the name of a server object registered on the SAS Metadata Server that associates SAS Cloud Analytic Services connection parameters with a server name.

LISTABOUT
: writes information about SAS Cloud Analytic Services to the SAS log.

LISTSERVERSTARTOPTS
: lists the SAS Cloud Analytic Services options and their current values.

Connection options

DISCONNECT
: disconnects SAS from the session.

HOST="*host-name*" | ("*primary-host-name*", "*backup-host-name*")
: specifies the machine name for the primary control node of the server and the backup control node (optional).

PORT=*number*
: specifies the port on which the control node of SAS Cloud Analytic Services listens for client connections.

RECONNECT
: reconnects to a session using a session name.

TERMINATE
: terminates the SAS Cloud Analytic Services session.

USER=*user-ID*
: specifies the user ID to use for connecting to SAS Cloud Analytic Services.

UUID="*session-uuid*"
: specifies the UUID of an existing session to which you want to connect.

UUIDMAC=*macro-variable-name*
: specifies a SAS macro variable name into which the UUID of the session is stored.

Session information options

LIST
: writes to the SAS log information about a session in your SAS client.

LISTHISTORY <*history_count* | _ALL_>
: prints the log of actions that were generated by statements and procedures and submitted to the CAS session.

LISTSESSIONS
: writes to the SAS log information about all of the current user's sessions that are known to SAS Cloud Analytic Services.

Session property options

LISTSESSOPTS
: lists the session options.

SESSOPTS=(*session-option(s)*)
: specifies one or more session option settings to apply during or after session start up.

User-defined format options

ADDFMTLIB FMTLIBNAME=*format-library-name* < < <TABLE=*table-name*> <CASLIB=*caslib*> > | <PATH=*path*> > <POSITION=APPEND | INSERT | REPLACE | NONE> <PROMOTE> <REPLACEFMTLIB>
: adds a session format library.

DROPFMTLIB FMTLIBNAME=*format-library-name* <FMTSEARCHREMOVE>
: drops a session-scope or a global-scope format library.

FMTSEARCH=(*name1 <name2…nameN>*) <POSITION=APPEND | INSERT | REPLACE>
: modifies the format library search list for the session.

FMTSEARCH CLEAR
: clears the format library search list for the session.

LISTFMTRANGES FMTNAME=*format-name*

lists the ranges and labels for a format.

LISTFMTSEARCH
displays the format library search list for the session.

LISTFORMATS <FMTLIBNAME=*format-library-name*> <SCOPE=BOTH | SESSION | GLOBAL> <MEMBERS>
lists the user-defined format libraries that are known to SAS.

LOADFORMATS PATH="*path-to-item-store-file*" <FMTLIBNAME=*format-library-name*> <PROMOTE> <REPLACE> <FMTSEARCH=APPEND | INSERT | REPLACE | NONE>
loads user-defined formats from a format item store file that was created with the FMTC2ITM procedure.

PROMOTEFMTLIB FMTLIBNAME=*format-library-name* <REPLACE>
promotes a session-local format library to a global format library.

SAVEFMTLIB FMTLIBNAME=*format-library-name* < < <TABLE=*table-name*> <CASLIB=*caslib*> <REPLACE> > | <PATH=*path*> > <PROMOTE>
saves a session format library to a CAS table or to a file.

Required Argument

session-name
specifies a valid SAS name that is less than 256 characters.

Notes Do not enclose *session-name* in quotation marks.

If session *session-name* does not exist, the session is started, and macro variable _SESSREF_ and SAS system option SESSREF are set to *session-name*.

The SAS Cloud Analytic Services system administrator can limit the number of concurrent sessions on the CAS server. See "Maximum Number of Sessions on the Server" on page 34.

Optional Arguments

ADDFMTLIB FMTLIBNAME=*format-library-name* < < <TABLE=*table-name*> <CASLIB=*caslib*> > | <PATH=*path*> > <POSITION=APPEND | INSERT | REPLACE | NONE> <PROMOTE> <REPLACEFMTLIB>
adds a session format library. By default, the format library is appended to the format library search list.

FMTLIBNAME=*format-library-name*
specifies the name of the format library to add.

Default _FMTLIB*n*, where *n* starts at 1 and is incremented for each format library that is added.

Note If the format library already exists in the session, specify REPLACEFMTLIB to replace the existing format library. Otherwise, an error occurs.

TABLE=*table-name*
specifies the name of the table where the format library was previously saved using SAVEFMTLIB.

Default the name specified in the FMTLIBNAME= option

Notes Do not specify this option with PATH=.

When adding a format library that is stored in a table on disk, you can specify the table name with or without the file extension (.sashdat).

CASLIB=*caslib*
specifies the name of the caslib where the table is saved.

Default the active caslib for the session

Note Do not specify this option with PATH=.

PATH=*path*
specifies the name of the file where the format information is saved.

Requirement The specified path must be readable from the control node of the server.

Note Do not specify this option with TABLE= or CASLIB=.

POSITION=APPEND | INSERT | REPLACE | NONE
specifies the position of this format library in the format-library search list.

APPEND appends this format library to the end of the format-library search list.

INSERT inserts this format library at the beginning of the format-library search list.

REPLACE replaces the current format-library search list with this format library.

NONE does not add this format library to the format-library search list.

Default APPEND

PROMOTE
promotes the format library to global scope so that it is available to all sessions.

REPLACEFMTLIB
replaces the format library if it already exists.

Tip The SAS Cloud Analytic Services system administrator can configure CAS to add format libraries to all sessions and promote them automatically at session start-up. See Configuration File Options in *SAS Viya Administration: SAS Cloud Analytic Services*.

See *SAS Cloud Analytic Services: User-Defined Formats*

Example "Example 11: Add a Format Library That Is Stored in a SASHDAT File" on page 47

AUTHDOMAIN=*authentication-domain* | "*authentication-domain*" | _SASMETA_ | "_SASMETA_"
obtains credentials from the SAS Metadata Server or the SAS Viya credentials service. SAS system option SERVICESBASEURL specifies the root URL for Viya services. If it is set, SAS first requests credentials for *authentication-domain* from the SAS Viya credentials service. If system option SERVICESBASEURL is not set

or if credentials cannot be obtained from the SAS Viya credentials service, SAS requests credentials from the SAS Metadata Server.

If credentials are obtained, a note is written to the SAS log identifying the source of the credentials. Otherwise, notes are written to the SAS log indicating that credentials could not be obtained, and a session is not started.

authentication-domain
specifies the name of an authentication domain object registered on the SAS Viya credentials service or the SAS Metadata Server that associates user credentials with an identity.

SASMETA
requests that the SAS Metadata Server generate a one-time password that can be used to access the CAS server.

Restriction _SASMETA_ is a special value that is valid with the SAS Metadata Server only. The SAS Viya credentials service does not return credentials for _SASMETA_.

Tip This option enables stored processes, token-based workspace servers, and pooled workspace servers to connect to the CAS server without the need for stored user credentials.

Restriction This option is valid only when starting a new session.

Requirements To obtain credentials from the SAS Metadata Server, the server connection parameters must be provided. If the connection parameters are not specified by the SAS system options, you are prompted to provide the connection parameters.

To obtain credentials from the Viya credentials service:

- System option SERVICESBASEURL must specify the base URL for the Viya services.
- Environment variable SAS_VIYA_TOKEN must specify a valid CAS OAuth access token for accessing Viya services. For information about environment variable SAS_VIYA_TOKEN, see SAS Viya: Overview.

To connect to the CAS server using credentials obtained from the SAS Metadata Server, the CAS server host name and port must be specified by the HOST= on page 26 and PORT= on page 30 options, or they must be obtained from the SAS Metadata Server by specifying the CASSERVERMD= on page 24 option.

Interactions Option USER= is ignored when this option is specified.

If AUTHDOMAIN=*authentication-domain* is specified with CASSERVERMD=*server-name*, *authentication-domain* overrides the authentication domain that is stored for *server-name* in the SAS Metadata Server.

See "SERVICESBASEURL= System Option" in *SAS System Options: Reference*

Examples This example requests that the credentials that are stored for identity CASDEMO be used to connect to CAS server casserver.mycompany.com on port 5570:

```
cas casauto authdomain="casdemo"
   host="casserver.mycompany.com" port=5570;
```

This example requests credentials for the authentication domain named CAS for the identity connected to the SAS Metadata Server:

```
cas casauto authdomain="_sasmeta_"
   casservermd="cas";
```

AUTHINFO="*authentication-info-file*"
specifies an authinfo file or netrc file that includes authentication information.

Default SAS system option AUTHINFO=, if set. Otherwise, file **`$HOME/.authinfo`** on Linux hosts.

Restriction This option is valid only when starting a new session.

Note An authinfo file is required when running batch jobs.

See "Create an Authinfo File" in *Client Authentication Using an Authinfo File*

CASSERVERMD=*server-name* | "*server-name*"
specifies the name of a server object registered on the SAS Metadata Server that associates SAS Cloud Analytic Services connection parameters with a server name.

Restriction This option is valid only when starting a new session.

Requirements The metadata system options must specify the connection parameters for the metadata server on which *server-name* is registered, and the metadata server must be running.

The *server-name* registration on the SAS Metadata Server must define the CAS server port, host name, and authentication domain.

Interaction Options HOST=, PORT=, and USER= are ignored when this option is specified.

Example This example requests a connection to CAS, which is the name of a CAS server registered on the SAS Metadata Server:

```
cas casauto casservermd="cas";
```

DISCONNECT
disconnects SAS from the session.

Note The session name is preserved for use with the RECONNECT option for the duration of the SAS session. If the SAS session is terminated after the session is disconnected, use the session's UUID to reconnect. See "Example 8: Connect to an Existing Session" on page 44.

Tip The session time-out value determines the lifetime in seconds of a disconnected session. The session time-out starts when the number of connections becomes zero and no actions are running. After the time-out expires, the session is terminated.

See "Example 7: Disconnect from a Session" on page 44

DROPFMTLIB FMTLIBNAME=*format-library-name* <FMTSEARCHREMOVE>
drops a session-scope or a global-scope format library.

FMTLIBNAME=*format-library-name*
specifies the name of the format library to drop. This option is required.

FMTSEARCHREMOVE
removes the format library from the format search list.

Tip If a session-scope and a global-scope format library with the same name exist, the session-scope format library is dropped first. To drop the global format library in that case, execute the drop command again.

See *SAS Cloud Analytic Services: User-Defined Formats*

Example "Example 18: Drop a Format Library from Session Scope" on page 52

FMTSEARCH=(*name1 <name2…nameN>*) <POSITION=APPEND | INSERT | REPLACE>
modifies the format library search list for the session.

(*name1 <name2…nameN>*)
specifies a list of one or more format-library names enclosed in parentheses. Each name is separated by a space.

POSITION=APPEND | INSERT | REPLACE
specifies the position of the format libraries in the format-library search list.

APPEND — appends the format libraries to the end of the format-library search list.

INSERT — inserts the format libraries at the beginning of the format-library search list.

REPLACE — replaces the current format-library search list with the specified format libraries.

Default APPEND

Tip Global format libraries are not automatically added to your format library search list when you start your session. Use this option to add any global format libraries that you want to use.

See *SAS Cloud Analytic Services: User-Defined Formats*

Example Add format libraries FMTLIB1 and FMTLIB2 to the beginning of the format library search list for session Casauto:

```
cas casauto fmtsearch=(fmtlib1 fmtlib2) position=insert;
```

Examples "Example 14: Add Global Format Libraries to a Session's Format Search List" on page 49

"Example 15: Change the Format Library Search List" on page 50

FMTSEARCH CLEAR
clears the format library search list for the session.

See *SAS Cloud Analytic Services: User-Defined Formats*

Example Clear the format search list for session Casauto:

```
cas casauto fmtsearch clear;
```

HOST="*host-name*" | ("*primary-host-name*", "*backup-host-name*")
specifies the machine name for the primary control node of the server and the backup control node (optional). When both the primary and backup hosts are specified and a connection cannot be established with the primary host, an attempt is made to connect to the backup host. If a connection is established, the session is started, and a note is written to the SAS log indicating the name of the host to which the session is connected. Otherwise, the session is not started.

Alias CASHOST

Default Macro variable _CASHOST_, if set. Otherwise, SAS system option CASHOST.

Restriction This option is valid only when starting a new session.

Interaction This option overrides macro variable _CASHOST_ and SAS system option CASHOST.

Notes Host names can be specified with or without quotation marks.

The primary host name and secondary host name can be specified as a comma-delimited list or as a space-delimited list.

If the specified primary host name is not valid, an error occurs and the session is not started, even when a valid backup host is specified.

Tip To determine whether a backup host is available, start a session on the primary server, and then run the following code:

```
proc cas;
   builtins.listNodes result = r;
run;
   print r.nodelist.where(role eq "controller");
run;
quit;
```

See "_CASHOST_ Macro Variable" on page 279

"CASHOST= System Option" on page 299

Examples Start a session on primary host cloud.example.com using port 5570:

```
cas casauto host="cloud.example.com" port=5570;
```

Start a session on primary host cloud.example.com using port 5570, and specify host cloudbackup.example.com as the backup server:

```
cas casauto host=("cloud.example.com", "cloudbackup.example.com") port=5570;
```

LIST
writes to the SAS log information about a session in your SAS client. The information includes the session name, the session state, the host and port of SAS Cloud Analytic Services to which it is connected, and the session UUID.

Note The server is not accessed for the information.

Tips Use **_ALL_** instead of *session-name* to list information about all of the sessions in your SAS client.

To list information about all of the sessions in all of your SAS clients, use LISTSESSIONS.

See "Example 2: List Information about the Sessions in Your SAS Client" on page 41

LISTABOUT
writes information about SAS Cloud Analytic Services to the SAS log. The information is organized as shown in the following table.

Section	Field	Description
About	CAS	Product name: SAS Cloud Analytic Services.
	Version	Short-form SAS Cloud Analytic Services version.
	VersionLong	Long-form SAS Cloud Analytic Services version.
	Copyright	Copyright information.
System	Hostname	SAS Cloud Analytic Services host name.
	OS Name	SAS Cloud Analytic Services host information.
	OS Family	
	OS Release	
	OS Version	
	Model Number	
	Documentation	SAS Cloud Analytic Services documentation URL.
License	site	Site information.
	siteNum	
	expires	License expiration date and time.
	gracePeriod	Grace period in days.
	warningPeriod	Expiration warning period in days.

LISTFMTRANGES FMTNAME=*format-name*
lists the ranges and labels for a format.

FMTNAME=*format-name*
specifies the name of the format to list. The format libraries in the format-library search list are searched in the order specified. The ranges and labels are listed for the first instance of *format-name* that is found.

See *SAS Cloud Analytic Services: User-Defined Formats*

Example "Example 16: List the Ranges in a Format" on page 51

LISTFMTSEARCH
displays the format library search list for the session.

See *SAS Cloud Analytic Services: User-Defined Formats*

Examples "Example 14: Add Global Format Libraries to a Session's Format Search List" on page 49

"Example 15: Change the Format Library Search List" on page 50

LISTFORMATS <FMTLIBNAME=*format-library-name*> <SCOPE=BOTH | SESSION | GLOBAL> <MEMBERS>
lists the user-defined format libraries that are known to SAS.

FMTLIBNAME=*format-library-name*
specifies the name of the format library. If FMTLIBNAME= is not specified, all of the format libraries that are known to SAS are listed.

SCOPE=BOTH | SESSION | GLOBAL
specifies the scope.

BOTH — lists both SESSION and GLOBAL format libraries that are known to SAS.

SESSION — lists the format libraries that are known to SAS.

GLOBAL — lists the format libraries that are known globally to all SAS sessions.

Default BOTH

MEMBERS
lists the names of the members in each format library.

See *SAS Cloud Analytic Services: User-Defined Formats*

Example "Example 13: List the Session-Scope Formats in a Session" on page 49

LISTHISTORY <*history_count* | _ALL_>
prints the log of actions that were generated by statements and procedures and submitted to the CAS session. For information about actions, see "Programming with CAS Actions" in *An Introduction to SAS Viya Programming*.

history_count
specifies the number of the most recent actions that are to be listed.

Range 0–1999999999

Note 0 is equivalent to _ALL_

ALL
lists all of the actions that have been executed in the current session.

Default Lists the last 10 actions that were executed in the current session.

Examples Print the last 10 actions that were executed in the current session:

```
cas casauto listhistory;
```

Print the last 5 actions that were executed in the current session:

```
cas casauto listhistory 5;
```

Print all of the actions that were executed in the current session:

```
cas casauto listhistory _all_;
```

LISTSERVERSTARTOPTS
lists the SAS Cloud Analytic Services options and their current values. For each option, a note containing information about the option is written to the SAS log. The information includes the option name, option value type, current option value, value range (when applicable), and value source, as shown in the following example.

```
NOTE: Name = errors
        Type = INT RANGE
        Value = 20
        Minimum = 0
        Maximum = 2147483647
        Source = default
```

Alias LISTSSO

Restriction You must have administration privileges to use this option.

LISTSESSIONS
writes to the SAS log information about all of the current user's sessions that are known to SAS Cloud Analytic Services. The information includes the session name, the session UUID, the session state, the method that was used for user authentication, and the user ID.

Requirement A connection to a session is required to use the LISTSESSIONS option.

See “Example 3: List Information about All of Your Sessions” on page 42

Example Use session Casauto to list all of your sessions.

```
cas casauto listsessions;
```

For each session that is found, a note containing information about that session is written to the SAS log.

LISTSESSOPTS
lists the session options.

See Chapter 13, “Session Options,” on page 281 for a list of the session options.

“Example 4: List the Properties for a Session” on page 42

Example List the session option settings for session Casauto:

```
cas casauto LISTSESSOPTS;
```

LOADFORMATS PATH="*path-to-item-store-file*" <FMTLIBNAME=*format-library-name*> <PROMOTE> <REPLACE> <FMTSEARCH=APPEND | INSERT | REPLACE | NONE>

loads user-defined formats from a format item store file that was created with the FMTC2ITM procedure.

Note: In order to use the LOADFORMATS option, your SAS client must be running on the same platform as the CAS server. Otherwise, an error might occur when you use the LOADFORMATS option. In that case, use the ADDFMTLIB on page 21 option to load the format library instead.

PATH="*path-to-item-store-file*"

specifies the name of the file where the item store is saved.

Requirement The specified path must be readable from the control node of the server.

FMTLIBNAME=*format-library-name*

specifies the name of the format library that is being added to the session. If FMTLIBNAME= is not specified, a name is generated.

Note A generated name cannot be promoted.

PROMOTE

promotes the format library to global scope.

REPLACE

if *format-library-name* already exists in the active session, replaces it with the new *format-library-name*.

Tip If the format library is already in the format search list, specify FMTSEARCH=NONE to avoid a duplicate name warning message in the SAS log.

FMTSEARCH=APPEND | INSERT | REPLACE | NONE

specifies how the new format library impacts the format library search order.

APPEND	appends the format library name to the end of the list of format libraries to be used when searching for a format.
INSERT	inserts the format library name at the beginning of the list of format libraries to be used when searching for a format.
REPLACE	clears the list of format libraries to search and replaces it with the format library that is being added.
NONE	does not modify the format search list.

Default APPEND

See "FMTC2ITM Procedure" in *Base SAS Procedures Guide*

PORT=*number*

specifies the port on which the control node of SAS Cloud Analytic Services listens for client connections.

Alias CASPORT

Default Macro variable _CASPORT_, if set. Otherwise, SAS system option CASPORT.

Range 1-65535

Restriction This option is valid only when starting a new session.

Interaction This option overrides macro variable _CASPORT_ and SAS system option CASPORT.

See "_CASPORT_ Macro Variable" on page 279

"CASPORT= System Option" on page 304

PROMOTEFMTLIB FMTLIBNAME=*format-library-name* <REPLACE>
promotes a session-local format library to a global format library.

FMTLIBNAME=*format-library-name*
specifies the name of the format library. This option is required.

REPLACE
replaces the format library if it is already promoted.

Tip Global format libraries are not automatically included in the format search list for new sessions. Use FMTSEARCH on page 25 to add any global format libraries that you want to use to your new session's format search list.

See *SAS Cloud Analytic Services: User-Defined Formats*

Example "Example 17: Promote a Format Library to Global Scope" on page 51

RECONNECT
reconnects to a session using a session name.

Tip If the session is not known to SAS, connect using the session UUID. Do not specify the RECONNECT keyword in that case. See "UUID="*session-uuid*"" on page 33.

See "Example 8: Connect to an Existing Session" on page 44

SAVEFMTLIB FMTLIBNAME=*format-library-name* < < <TABLE=*table-name*> <CASLIB=*caslib*> <REPLACE> > | <PATH=*path*> > <PROMOTE>
saves a session format library to a CAS table or to a file.

FMTLIBNAME=*format-library-name*
specifies the name of the format library to save. This option is required.

TABLE=*table-name*
specifies the name of the table in which the format library is saved.

Default the name specified in the FMTLIBNAME= option

Notes Do not specify this option with PATH=.

When saving a format library to a table on disk, you can specify the table name with or without the file extension (.sashdat).

If the table already exists in the caslib, specify REPLACE to replace the existing table. Otherwise, an error occurs.

Do not change the name of the format library file after you save it. If you change the filename, the format library that it contains cannot be loaded.

CASLIB=*caslib*
specifies the caslib in which the table is stored.

Default the active caslib for the session

Requirement You must have Write access to the specified caslib.

Note Do not specify this option with PATH=.

Tip Contact your SAS Cloud Analytic Services system administrator if you want to store your format library in the FORMATS caslib.

REPLACE
replaces the table if it already exists.

Alias REPLACE

Note This option is ignored when PATH= is specified.

PATH=*path*
specifies the name of the file to which the format library is to be saved.

Note Do not specify this option with TABLE= or CASLIB=.

PROMOTE
promotes the format library to global scope so that it is available to all sessions.

See *SAS Cloud Analytic Services: User-Defined Formats*

Example "Example 10: Create a Format Library and Save It to a SASHDAT File" on page 45

SESSOPTS=(*session-option(s)*)
specifies one or more session option settings to apply during or after session start up.

See Chapter 13, "Session Options," on page 281 for a list of the options that you can specify for *session-option(s)*.

"Example 5: Change a Property for a Session" on page 43

"Program: Start a Session with Custom Properties" on page 40

TERMINATE
terminates the SAS Cloud Analytic Services session.

Alias CLEAR

Note When you terminate the active session, SAS system option SESSREF= continues to reference the terminated session. A note is written to the SAS log indicating that the session identified by SAS system option SESSREF= was terminated. To access the server in that case, you must start a new

session or set system option SESSREF= to an existing connected session. See "Example 1: Start a Session" on page 40 and "SESSREF= System Option" on page 309.

Tip Use **_ALL_** instead of *session-name* to terminate all of the sessions in your SAS client.

See "Example 9: Terminate a Session" on page 45

USER=*user-ID*
specifies the user ID to use for connecting to SAS Cloud Analytic Services.

Alias CASUSER

Default SAS system option CASUSER=, if set.

Restriction This option is valid only when starting a new session.

Requirement *User-ID* must match a user ID in your authinfo file. See *SAS Viya Administration: Authentication*.

Interaction This option overrides SAS system option CASUSER=.

Note When you use SAS Studio, the user credentials that you used to sign on are used to authenticate your connection to CAS. The USER= option is not needed in that case. The USER= option or SAS system option CASUSER= is used when submitting code to CAS from the command line, in batch mode.

See "CASUSER= System Option" on page 307

UUID="*session-uuid*"
specifies the UUID of an existing session to which you want to connect.

Requirements *session-uuid* must be 36 characters in length and must be enclosed in quotation marks.

You must also specify the HOST= and PORT= options to connect to a session.

Tip You can view the UUID for a session with the LIST command option.

See "LIST" on page 26

"Program: Connect to an Existing Session Using the Session Name and UUID" on page 44

UUIDMAC=*macro-variable-name*
specifies a SAS macro variable name into which the UUID of the session is stored.

Tip The UUIDMAC= option is useful if you want subsequent SAS steps to connect to the session by specifying the UUID.

See "Program: Start a Session and Store the UUID in a Macro Variable" on page 40

Details

What Can I Do with the CAS Statement?

You can do the following tasks with the CAS statement:

- list information about a specific SAS Cloud Analytic Services session or all of your sessions
- list the properties of a session
- manage format libraries in a session
- change one or more session properties
- disconnect a session
- connect to an existing session
- start a session
- terminate a session

Starting Your Initial Session

After you sign in to SAS Studio, you must start a session in order to connect to SAS Cloud Analytic Services. After you start your session, you can use it to complete your tasks. Code snippet **New Session** in SAS Studio provides the SAS code that is needed to start a session. Alternatively, you can submit your own CAS statement to start a customized session. See "Example 1: Start a Session" on page 40 for examples. Use the CAS statement to perform management tasks on your session, such as listing or changing properties, managing format libraries, and so on.

CAS Statement Status

When you execute a CAS statement for the first time in your SAS session, global macro variable CASSTMTERR is created. It is set to 0 if the CAS statement was successful, 1 if an error occurred, or 2 if a warning was issued. The CASSTMTERR macro variable is updated each time you execute a CAS statement. You can use the CASSTMTERR macro variable in your SAS program to test the status of your last CAS statement and proceed accordingly. For an example, see "Example 7: Disconnect from a Session" on page 44.

Maximum Number of Sessions on the Server

The maximum number of concurrent sessions is determined by SAS Cloud Analytic Services configuration option MAXSESSIONS. The default is 5000. Administrative users are exempt from this limit. When the limit is reached, non-administrative users are denied access until the number of sessions drops below the maximum. In that case, an error message is written to the SAS log indicating that the limit is reached, and the requested session is not started. For information about configuration option MAXSESSIONS, see Configuration File Options in *SAS Viya Administration: SAS Cloud Analytic Services*.

Managing Sessions Using the CAS Server Monitor

You can also use the CAS Server Monitor to manage your sessions. Using the Server Monitor, you can perform the following session tasks:

- list information about your sessions
- cancel the action that is currently running in a session
- cancel a session

- terminate a session

Note: Some tasks in the CAS Server Monitor might require administration privileges.

Use the GETCASURL function to get the CAS Server Monitor URL. See "GETCASURL Function" on page 274.

For information about the CAS Server Monitor, see SAS Viya Administration: Using CAS Server Monitor.

Managing User-Defined Formats Using the CAS Statement

The CAS statement provides several options that enable you to perform common tasks related to user-defined format libraries. The following table maps common user-defined format library tasks to the applicable CAS statement options and examples.

Table 2.1 *Performing Common Tasks for User-Defined Formats Using the CAS Statement*

Task	What to Use	Example(s)
Migrate your user-defined formats stored in SAS format catalogs to CAS.	The IFTC2ITM procedure and the CAS statement LOADFORMATS option	"Add Formats in Format Catalogs to a CAS Session" in *SAS Cloud Analytic Services: User-Defined Formats*
Add a user-defined format library stored in a SASHDAT file to your session.	The CAS statement ADDFMTLIB on page 21 option	"Example 11: Add a Format Library That Is Stored in a SASHDAT File" (p. 47)
Promote a session-scope user-defined format library in your session to global scope.	The CAS statement PROMOTEFMTLIB on page 31 option	"Example 17: Promote a Format Library to Global Scope" (p. 51)
List the format libraries and formats that are available in your session.	The CAS statement LISTFORMATS on page 28 option	"Example 13: List the Session-Scope Formats in a Session" (p. 49)
List the ranges in a user-defined format in your session.	The CAS statement LISTFMTRANGES on page 27 option	"Example 16: List the Ranges in a Format" (p. 51)
Display the user-defined format library search list for your session.	The CAS statement LISTFMTSEARCH on page 28 option	"Example 14: Add Global Format Libraries to a Session's Format Search List" (p. 49) "Example 15: Change the Format Library Search List" (p. 50)
Modify your session's user-defined format library search list.	The CAS statement FMTSEARCH on page 25 option	"Example 14: Add Global Format Libraries to a Session's Format Search List" (p. 49) "Example 15: Change the Format Library Search List" (p. 50)

Task	What to Use	Example(s)
Save a format library in your session to a SASHDAT file.	The CAS statement SAVEFMTLIB on page 31 option	"Example 10: Create a Format Library and Save It to a SASHDAT File" (p. 45)
Drop a session-scope or global-scope user-defined format library.	The CAS statement DROPFMTLIB on page 24 option	"Example 18: Drop a Format Library from Session Scope" (p. 52)

For information about user-defined formats in SAS Cloud Analytic Services, see *SAS Cloud Analytic Services: User-Defined Formats*.

Troubleshooting Session Errors

What Session-Related Errors Does This Section Cover?

This section covers the following CAS session-related errors:

- A Connection to a Cloud Analytic Services Session Could Not Be Made
- Could Not Find Netrc or Authinfo File
- Missing Session Information
- Request to Connect Failed for UUID
- Server Has Met the Maximum Number of Concurrent Sessions
- Session Cannot Be Resolved
- Session Is Not Recognized
- Session Connection for a Session Is Not Active
- Unable to Connect to Cloud Analytic Services

A Connection to a Cloud Analytic Services Session Could Not Be Made

```
WARNING: Session session-name is disconnected.

ERROR: A connection to the Cloud Analytic Services session
session-name could not be made. Make sure that the session name is
correctly specified and that it is an active session.
```

Note: The warning message might not appear.

Cause	Solution
A reference to session *session-name* was made in the CASUTIL procedure SESSREF= option but the specified session is disconnected, does not exist, or is not known to SAS.	• If a warning message indicates that *session-name* is disconnected, reconnect to session *session-name*. See "Example 8: Connect to an Existing Session" on page 44. • Verify that session *session-name* exists. See "Example 3: List Information about All of Your Sessions" on page 42. If session *session-name* does not exist, specify a different session or start session *session-name*. See "Example 1: Start a Session" on page 40.

Could Not Find Netrc or Authinfo File

```
ERROR: Could not find netrc or authinfo file.
```

Cause	Solution
Your netrc or .authinfo file that is required for authentication could not be found.	Verify that a valid .authinfo file exists for your user ID and that the file permissions grant Read and Write access to you only. See "Create an Authinfo File" in *Client Authentication Using an Authinfo File*.

Missing Session Information

```
WARNING: Session session-name is disconnected.

ERROR: Missing session information.
```

Note: The warning message might not appear.

Cause	Solution
A reference to session *session-name* was made in a LIBNAME statement, but the specified session is disconnected, does not exist, or is not known to SAS.	• If a warning message indicates that *session-name* is disconnected, reconnect to session *session-name*. See "Example 8: Connect to an Existing Session" on page 44. • Verify that session *session-name* exists. See "Example 3: List Information about All of Your Sessions" on page 42. If session *session-name* does not exist, specify a different session or start session *session-name*. See "Example 1: Start a Session" on page 40.

Request to Connect Failed for UUID

```
ERROR: Request to CONNECT failed for UUID session-uuid.
Failure occurs when a disconnected CAS session exceeds the timeout
value and terminates, when the specified UUID is not correct or
designates a session that is already connected, or when
authentication fails.

ERROR: Connection failed. Server returned: Authentication failed: Access denied.
```

Note: The authentication error message might not appear.

Cause	Solution
An attempt was made to connect to a session using its name and UUID, but the session was not found or a connection could not be established.	• If authentication failed, verify that your credentials are correct. For information about authentication in SAS Cloud Analytic Services, see SAS Viya Administration: Authentication. • Verify that session *session-uuid* exists. See "Example 3: List Information about All of Your Sessions" on page 42. If session *session-uuid* does not exist, specify a different session UUID or start a new session. See "Example 1: Start a Session" on page 40.

Server Has Met the Maximum Number of Concurrent Sessions

The following error messages appear:

```
ERROR: Unable to connect to Cloud Analytic Services host-name on port host-port

ERROR: Connection failed. The server has met the maximum number of concurrent
sessions. Contact your site administrator.
```

Cause	Solution
The maximum number of concurrent sessions specified by the SAS Cloud Analytic Services MAXSESSIONS configuration parameter has been reached.	Contact your SAS Cloud Analytic Services system administrator. For more information, see "Maximum Number of Sessions on the Server" on page 34.

Session Cannot Be Resolved

```
WARNING: Session session-name is disconnected.

ERROR:  Session reference 'session-name' cannot be resolved
```

Note: The warning message might not appear.

Cause	Solution
A reference to session *session-name* was made in the CAS procedure SESSION statement, but the specified session is disconnected, does not exist, or is not known to SAS.	• If a warning message indicates that *session-name* is disconnected, reconnect to session *session-name*. See "Example 8: Connect to an Existing Session" on page 44. • Verify that session *session-name* exists. See "Example 3: List Information about All of Your Sessions" on page 42. If session *session-name* does not exist, specify a different session or start session *session-name*. See "Example 1: Start a Session" on page 40.

Session Is Not Recognized

```
WARNING: Session session-name is disconnected.

ERROR: Request failed. Session session-name not recognized.
```

Note: The warning message might not appear.

Cause	Solution
A request was made on session *session-name*, but session *session-name* is disconnected, does not exist, or is not known to SAS.	• If a warning message indicates that *session-name* is disconnected, reconnect to session *session-name*. See "Example 8: Connect to an Existing Session" on page 44. • Verify that session *session-name* exists. See "Example 3: List Information about All of Your Sessions" on page 42. If session *session-name* does not exist, specify a different session or start session *session-name*. See "Example 1: Start a Session" on page 40.

Session Connection for a Session Is Not Active

An error message similar to the following appears:

```
ERROR: Request to LISTSESSOPTS failed. The session
connection for session-name is not active.
```

Cause	Solution
A request such as LISTSESSOPTS or LISTFORMATS was made on session *session-name*, but session *session-name* is disconnected.	Reconnect to session *session-name*. See "Example 8: Connect to an Existing Session" on page 44.

Unable to Connect to Cloud Analytic Services

One of the following error messages appears:

```
ERROR: Unable to connect to Cloud Analytic Services host-name
on port host-port Verify connection parameters and retry.

ERROR: Connection failed. Server returned: Authentication
failed: Access denied.
```

Cause	Solution
The specified *host-name* or *host-port* is invalid, or authentication failed.	• If authentication failed, verify that your credentials are correct. For information about authentication in SAS Cloud Analytic Services, see SAS Viya Administration: Authentication. • Verify that the specified *host-name* and *host-port* are correct. • If the connection parameters are correct, verify that *host-name* is available.

Examples

Example 1: Start a Session

Program: Start a Session with Default Properties

This example starts a session that is named Casauto with default session properties. If necessary, set system options CASHOST= and CASPORT= to a host and port that are valid for your site.

```
/* options cashost="cloud.example.com" casport=5570; */
cas casauto;
```

SAS Log

Notes similar to the following are written to the SAS log:

```
NOTE: The session CASAUTO connected successfully to Cloud Analytic Services
      cloud.example.com using port 5570. The UUID is session-UUID.
      The user is sasdemo and the default CASLIB is
      CASUSERHDFS(sasdemo).
NOTE: The SAS option SESSREF was updated with the value CASAUTO.
NOTE: The SAS macro _SESSREF_ was updated with the value CASAUTO.
NOTE: The session is using nnn workers.
```

Program: Start a Session with Custom Properties

This example starts a session that is named Casauto. Session options are specified to set the active caslib to CASUSER and to enable metrics reporting:

```
/* options cashost="cloud.example.com" casport=5570; */
cas casauto sessopts=(caslib=casuser metrics=True);
```

SAS Log

Notes similar to the following are written to the SAS log:

```
NOTE: The session CASAUTO connected successfully to Cloud Analytic Services
      cloud.example.com using port 5570. The UUID is session-UUID.
      The user is sasdemo and the default CASLIB is
      CASUSER(sasdemo).
NOTE: The SAS option SESSREF was updated with the value CASAUTO.
NOTE: The SAS macro _SESSREF_ was updated with the value CASAUTO.
NOTE: The session is using nnn workers.
NOTE: 'CASUSER(sasdemo)' is now the active caslib.
NOTE: Action 'setsessopt' used (Total process time):
NOTE:       real time               0.181410 seconds
NOTE:       cpu time                0.446927 seconds (246.36%)
NOTE:       total nodes             142 (4536 cores)
NOTE:       total memory            35.47T
NOTE:       memory                  13.09M (0.00%)
NOTE: The CAS server request to update one or more session options
      for session CASAUTO completed.
```

Program: Start a Session and Store the UUID in a Macro Variable

This example starts a session that is named Casauto and stores the session UUID in macro variable CASAUTO_UUID for later use:

```
/* options cashost="cloud.example.com" casport=5570; */
cas casauto uuidmac=casauto_uuid;
```

```
%put Session casauto UUID: &casauto_uuid;
```

SAS Log

The %PUT statement writes the session UUID to the SAS log. Here is an example.

```
Session casauto UUID: 55c7425b-e383-794a-830a-055731b4e211
```

Additional Information

Here are some best practices to follow when creating your own session:

- Always specify the name of your session in CAS statements that provide a SESSREF= option such as the CASLIB and LIBNAME statements. This helps ensure that the caslib or library reference is associated with the correct session. See "CASLIB Statement" on page 91 and "CAS LIBNAME Statement" on page 66.
- If you need to disconnect from your session, be sure to set the TIMEOUT property for the session to an appropriate value before you disconnect. If no other client connections exist and no actions are running when you disconnect, your session is automatically terminated if you do not reconnect within the time-out period. See TIMEOUT= on page 295.
- To help conserve system resources, always terminate your session when you are finished with it.

Example 2: List Information about the Sessions in Your SAS Client

Program: List a Specific Session in Your SAS Client

This example lists information about SAS client session Casauto:

```
cas casauto list;
```

SAS Log

A note similar to the following is written to the SAS log:

```
NOTE: Session CASAUTO is ACTIVE using port 5570 and host
cloud.example.com for user sasdemo. The session UUID is
session-UUID.
```

Program: List All of the Sessions in Your SAS Client

This example lists information about all of the sessions that are in a SAS client, which are Casauto, Mysess1, and Mysess2, in this example:

```
cas _all_ list;
```

SAS Log

A note is written to the SAS log for each of the three sessions as shown in the following example:

```
NOTE: Session CASAUTO is ACTIVE using port 5570 and host
cloud.example.com for user sasdemo. The session UUID is
session-UUID.
NOTE: Session MYSESS1 is ACTIVE using port 5570 and host
cloud.example.com for user sasdemo. The session UUID is
session-UUID.
NOTE: Session MYSESS2 is ACTIVE using port 5570 and host
cloud.example.com for user sasdemo. The session UUID is
session-UUID.
```

Example 3: List Information about All of Your Sessions

This example uses existing session Casauto to list information about all of the current user's sessions that are known to SAS Cloud Analytic Services. You must be connected to an existing session in order to use the LISTSESSIONS option.

```
cas casauto listsessions;
```

SAS Log

For each of the current user's sessions, a note is written to the SAS log that contains information about that session. Here is an example that shows information for sessions Casauto, Mysess1, and Mysess2.

```
NOTE: SessionName = CASAUTO:Mon Feb  8 12:49:00 2016
         UUID= session-UUID
         State = Connected
         Authentication = Active Directory
         Userid = sasdemo
NOTE: SessionName = MYSESS1:Mon Feb  8 12:49:03 2016
         UUID= session-UUID
         State = Connected
         Authentication = Active Directory
         Userid = sasdemo
NOTE: SessionName = MYSESS2:Mon Feb  8 12:49:05 2016
         UUID= session-UUID
         State = Connected
         Authentication = Active Directory
         Userid = sasdemo
NOTE: Request to LISTSESSIONS completed for session CASAUTO.
```

Example 4: List the Properties for a Session

Program

This example lists the properties for session Casauto:

```
cas casauto listsessopts;
```

SAS Log

A note containing property information is written to the SAS log for each session property. Here is a partial example.

```
NOTE: Name = appTag
         UsageType = Session
         Type = String
         Value =
         Default Value =
         Group = Action
         Min = 0
         Max = 0
         Description = specifies the string to prefix to log messages.
NOTE: Name = caslib
         UsageType = Session
         Type = String
         Value = CASUSER(sasdemo)
         Default Value =
         Group = Caslib
         Min = 0
         Max = 0
         Description = specifies the caslib name to set as the active caslib.
...
```

Additional Information

For information about the session properties, see Chapter 13, "Session Options," on page 281. The GETSESSOPT function enables you to get the value of a single property. See "GETSESSOPT Function" on page 276.

Example 5: Change a Property for a Session

Program

Change the time-out to 60 minutes for session Casauto:

```
cas casauto sessopts=(timeout=3600);
```

Additional Information

For information about the session properties that you can change, see Chapter 13, "Session Options," on page 281. To change one or more properties for all of your generated SAS Cloud Analytic Services sessions, specify the property settings in an OPTIONS statement. See "CAS Statement" on page 19.

Example 6: Display a Session Property Value

Program

Display the current TIMEOUT property value for session Casauto:

```
%put TIMEOUT for session CASAUTO: %sysfunc(getsessopt(casauto, timeout)) minutes.;
```

SAS Log

The following is written to the SAS log:

```
TIMEOUT for session CASAUTO: 60 minutes.
```

Example 7: Disconnect from a Session

Before You Disconnect from Your Session

If no other client connection exists and no actions are running when you disconnect, you must reconnect to the session before the connection time-out expires (60 seconds by default). Otherwise, the session is automatically terminated. Before you disconnect from a session, be sure to set session option TIMEOUT= for that session to an appropriate value. Use the RECONNECT option to reconnect to the session. See "Example 8: Connect to an Existing Session" on page 44.

Program

This example sets the time-out for session Casauto to 90 minutes, and then, if the time-out was successfully set, disconnects from session Casauto:

```
cas casauto sessopts=(timeout=5400);
%if &CASSTMTERR eq 0 %then %do;
   cas casauto disconnect;
%end;
```

SAS Log

The following notes are written to the SAS log:

```
NOTE: The CAS server request to update one or more session
      options for session CASAUTO completed.
NOTE: Request to DISCONNECT completed for session CASAUTO.
```

Example 8: Connect to an Existing Session

Program: Reconnect to an Existing Session Using the Session Name

If you want to reconnect to a session that you started or connected to previously in your current SAS session, you must specify the session name to reconnect. This example reconnects to session Casauto, which was started previously in the current SAS session:

```
cas casauto reconnect;
```

SAS Log

The following note is written to the SAS log:

```
NOTE: Request to RECONNECT completed for session CASAUTO.
```

Program: Connect to an Existing Session Using the Session Name and UUID

If you want to connect to a session that you started in a different SAS session, you must specify the session name and UUID to connect. If you do not know the UUID of the session, use the LISTSESSIONS= option as described in "Example 3: List Information about All of Your Sessions" on page 42. This example uses session name Mysess and the UUID option to connect to session Mysess, which was started in a different SAS session.

```
cas mysess uuid="ca683ddf-fe18-3c48-a04e-45718220976d";
```

SAS Log

The following notes are written to the SAS log.

```
NOTE: The session MYSESS connected successfully to Cloud Analytic
      Services cloud.example.com using port 5570. The UUID is
      ca683ddf-fe18-3c48-a04e-45718220976d. The user is sasdemo and the
      default CASLIB is CASUSER(sasdemo).
NOTE: The session is using nnn workers.
```

Example 9: Terminate a Session

Program

Terminate session Casauto:

```
cas casauto terminate;
```

SAS Log

The following notes are written to the SAS log:

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

Example 10: Create a Format Library and Save It to a SASHDAT File

Program

Create format library MyFmtLib, populate it with character format $Codes, numeric format Response, and numeric range format MPGrating, and then store MyFmtLib in file myFmtLib.sashdat in caslib CASUSER.

```
/* options cashost="cloud.example.com" casport=5570; */
cas casauto;                                          /* 1 */

proc format sessref=casauto casfmtlib="myFmtLib";  /* 2 */
   value $codes
      "A" = "Alpha"
      "B" = "Beta"
      "C" = "Charlie"
      "D" = "Delta";
run;

proc format sessref=casauto casfmtlib="myFmtLib";  /* 3 */
   value response
      1 = "Yes"
      2 = "No"
      3 = "Undecided"
      4 = "No response";
run;

proc format sessref=casauto casfmtlib="myFmtLib";  /* 4 */
   value MPGrating
      34  -  HIGH = "Excellent"
      24  -< 34   = "Good"
      19  -< 24   = "Fair"
```

```
      LOW -< 19   = "Poor";
run;

cas casauto savefmtlib fmtlibname=myFmtLib         /* 5 */
   table="myFmtLib.sashdat" caslib=casuser replace;
```

1 If you have not already done so, start your CAS session.

2 Use the FORMAT procedure to create character format $Codes. Option CASFMTLIB= specifies the name of the format library in CAS. The FORMAT procedure stores character format $Codes in the default SAS format catalog Work.Formats and in the CAS in-memory format library MyFmtLib.

Note: Use the FORMAT procedure option LIBRARY= to store the format in a different catalog in SAS. For an example, see "Add and Save a User-Defined Format" in *SAS Cloud Analytic Services: User-Defined Formats*.

Note: Use the FORMAT procedure to add existing SAS format catalogs to your CAS session. See "Add Formats in Format Catalogs to a CAS Session" in *SAS Cloud Analytic Services: User-Defined Formats*.

3 Create numeric format Response. Numeric format Response is stored in the SAS format catalog Work.Formats and in the CAS in-memory format library MyFmtLib.

4 Create range format MPGrating. Numeric range format MPGrating is stored in the SAS format catalog Work.Formats and in the CAS in-memory format library MyFmtLib.

5 Save the CAS in-memory format library MyFmtLib to file myFmtLib.sashdat in caslib CASUSER for later use. The TABLE= option specifies the name of the file on disk. You can specify the name with or without the .sashdat file extension. If TABLE= is not specified, the name specified in FMTLIBNAME with the .sashdat file extension is used as the filename. The REPLACE option replaces the table file if it already exists in CASUSER. If the table already exists and you do not specify REPLACE, an error occurs.

SAS Log

```
1          cas casauto;
   ...
2
3          proc format sessref=casauto casfmtlib="myFmtLib";
NOTE: Both CAS based formats and catalog-based formats will be written.
      The CAS based formats will be written to the session CASAUTO.
4             value $codes
5                "A" = "Alpha"
6                "B" = "Beta"
7                "C" = "Charlie"
8                "D" = "Delta";
NOTE: Format library MYFMTLIB added. Format search update using
      parameter APPEND completed.
NOTE: Format $CODES has been output.
9          run;
   ...
10
11         proc format sessref=casauto casfmtlib="myFmtLib";
NOTE: Both CAS based formats and catalog-based formats will be written.
      The CAS based formats will be written to the session CASAUTO.
12            value response
13               1 = "Yes"
14               2 = "No"
15               3 = "Undecided"
16               4 = "No response";
NOTE: Format RESPONSE has been output.
17         run;
   ...
18
19         proc format sessref=casauto casfmtlib="myFmtLib";
NOTE: Both CAS based formats and catalog-based formats will be written.
      The CAS based formats will be written to the session CASAUTO.
20            value MPGrating
21               34  -  HIGH = "Excellent"
22               24  -< 34   = "Good"
23               19  -< 24   = "Fair"
24               LOW -< 19   = "Poor";
NOTE: Format MPGRATING has been output.
25         run;
   ...
26
27         cas casauto savefmtlib fmtlibname=myFmtLib
28            table="myFmtLib.sashdat" caslib=casuser replace;
NOTE: Cloud Analytic Services saved the file myFmtLib to HDFS in
      caslib CASUSER(sasdemo).
NOTE: The format library MYFMTLIB save to myFmtLib.sashdat
      completed successfully.
NOTE: Request to SAVEFMTLIB MYFMTLIB completed for session CASAUTO.
29
30         cas casauto terminate;
NOTE: Deletion of the session CASAUTO was successful.
NOTE: Request to TERMINATE completed for session CASAUTO.
```

Example 11: Add a Format Library That Is Stored in a SASHDAT File

Program

Add to session Casauto format library MyFmtLib, which was stored in file myFmtLib.sashdat in “Example 10: Create a Format Library and Save It to a SASHDAT File” on page 45.

```
/* options cashost="cloud.example.com" casport=5570; */
cas casauto;                                    /* 1 */
```

```
cas casauto addfmtlib fmtlibname=myFmtLib     /* 2 */
   table="myFmtLib.sashdat" caslib=casuser replacefmtlib;

cas casauto listformats fmtlibname=myFmtLib;  /* 3 */
```

1 If you have not already done so, start your CAS session.

2 Use the FMTLIBNAME option to specify a name for the format library in session Casauto. The TABLE= option specifies the name of the format library file in caslib CASUSER, which is myFmtLib.sashdat. The name can be specified with or without the .sashdat file extension. If you do not specify TABLE=, the filename is assumed to be the name specified in FMTLIBNAME= with the .sashdat file extension. The REPLACEFMTLIB option replaces format library MyFmtLib if it already exists in session Casauto. If MyFmtLib already exists in session Casauto and you do not specify REPLACEFMTLIB, an error occurs.

   By default, format library MyFmtLib is placed at the end of the format library search list (APPEND). Use the POSITION= option to specify a different position in the search list.

3 Use LISTFORMATS to verify that format library MyFmtLib was added to session Casauto.

SAS Log

```
1          cas casauto;
   ...
2
3          cas casauto addfmtlib fmtlibname=myFmtLib
4             table="myFmtLib.sashdat" caslib=casuser replacefmtlib;
NOTE: Format library MYFMTLIB added. Format search update using
      parameter APPEND completed.
NOTE: Request to ADDFMTLIB MYFMTLIB completed for session CASAUTO.
5
6          cas casauto listformats fmtlibname=myFmtLib;
NOTE: Fmtlib = MYFMTLIB
         Scope = Session
         Fmtsearch = YES
NOTE: Request to LISTFORMAT completed for session CASAUTO.
```

Example 12: List the Format Libraries That Are Available to a Session

Program

List all of the format libraries that are available to session Casauto.

```
cas casauto listformats;
```

SAS Log

Here is an example of what is written to the SAS log.

```
     cas casauto listformats;
NOTE: Fmtlib = COMPANYFORMATS
         Scope = Global
         Fmtsearch = NA
NOTE: Fmtlib = DEPTFORMATS
         Scope = Global
         Fmtsearch = NA
NOTE: Fmtlib = MYFMTLIB
         Scope = Session
         Fmtsearch = YES
NOTE: Request to LISTFORMAT completed for session CASAUTO.
```

Example 13: List the Session-Scope Formats in a Session

Program

List the session-scope format libraries and their formats in session Casauto.

```
cas casauto listformats scope=session members;
```

SAS Log

Here is an example of what is written to the SAS log.

```
     cas casauto listformats scope=session members;
NOTE: Fmtlib = MYFMTLIB
         Scope = Session
         Fmtsearch = YES
         Format = $codes
         Format = mpgrating
         Format = response
NOTE: Request to LISTFORMAT completed for session CASAUTO.
```

Example 14: Add Global Format Libraries to a Session's Format Search List

Program

Add global format libraries CompanyFormats and DeptFormats to the format search list for session Casauto. This example assumes that format libraries CompanyFormats and DeptFormats exist in global scope, and format library MyFmtLib exists in session scope. (See "Example 11: Add a Format Library That Is Stored in a SASHDAT File" on page 47.)

```
cas casauto listformats;                    /* 1 */
cas casauto                                 /* 2 */
   fmtsearch=(companyFormats deptFormats)
   position=insert;
cas casauto listfmtsearch;                  /* 3 */
```

1 List all of the format libraries that are available to session Casauto. A note is written to the SAS log for each format library.

2 Insert format libraries CompanyFormats and DeptFormats at the beginning of the search list.

3 Use LISTFMTSEARCH to verify that format libraries CompanyFormats and DeptFormats are now included in the format search list.

SAS Log

```
1          cas casauto listformats;
 NOTE: Fmtlib = COMPANYFORMATS
          Scope = Global
          Fmtsearch = NA
 NOTE: Fmtlib = DEPTFORMATS
          Scope = Global
          Fmtsearch = NA
 NOTE: Fmtlib = MYFMTLIB
          Scope = Session
          Fmtsearch = YES
 NOTE: Request to LISTFORMAT completed for session CASAUTO.
 2
 3          cas casauto
 4             fmtsearch=(companyFormats deptFormats)
 5             position=insert;
 NOTE: Request to FMTSEARCH completed for session CASAUTO.
 6          cas casauto listfmtsearch;
 NOTE: FmtLibName = COMPANYFORMATS
          Scope = Both
 NOTE: FmtLibName = DEPTFORMATS
          Scope = Both
 NOTE: FmtLibName = MYFMTLIB
          Scope = Session
 NOTE: Request to LISTFMTSEARCH completed for session CASAUTO.
```

Example 15: Change the Format Library Search List

Program

Change the format library search list shown in "Example 14: Add Global Format Libraries to a Session's Format Search List" on page 49 to CompanyFormats, MyFmtLib, DeptFormats so that MyFmtLib can override formats in DeptFormats.

```
cas casauto
   fmtsearch=(companyformats, myFmtLib, deptformats)  /* 1 */
   position=replace;

cas casauto listfmtsearch;                            /* 2 */
```

1 Change the order to CompanyFormats, MyFmtLib, DeptFormats. POSITION=REPLACE replaces the current search list with the new search list.

2 Use LISTFMTSEARCH to verify the new search list.

SAS Log

```
     cas casauto
              fmtsearch=(companyformats, myFmtLib, deptformats)
              position=replace;
NOTE: Request to FMTSEARCH completed for session CASAUTO.

     cas casauto listfmtsearch;
NOTE: FmtLibName = COMPANYFORMATS
         Scope = Session
NOTE: FmtLibName = MYFMTLIB
         Scope = Session
NOTE: FmtLibName = DEPTFORMATS
         Scope = Session
NOTE: Request to LISTFMTSEARCH completed for session CASAUTO.
```

Example 16: List the Ranges in a Format

Program

List the ranges in formats Response and MPGrating in format library MyFmtLib, which was added to session Casauto in "Example 11: Add a Format Library That Is Stored in a SASHDAT File" on page 47.

```
cas casauto listfmtranges fmtname=response;
cas casauto listfmtranges fmtname=MPGrating;
```

SAS Log

```
cas casauto listfmtranges fmtname=response;
Format Name         Range
RESPONSE            1=Yes
                    2=No
                    3=Undecided
                    4=No response

NOTE: Request to LISTFMTRANGES RESPONSE completed for session CASAUTO.
cas casauto listfmtranges fmtname=MPGrating;
Format Name         Range
MPGRATING           LOW-<19=Poor
                    19-<24=Fair
                    24-<34=Good
                    34-HIGH=Excellent
```

Example 17: Promote a Format Library to Global Scope

Program

Promote format library MyFmtLib, which was added in "Example 11: Add a Format Library That Is Stored in a SASHDAT File" on page 47, to global scope.

```
cas casauto promotefmtlib fmtlibname=myFmtLib;  /* 1 */

cas casauto listformats;                        /* 2 */
```

1 Promote format library MyFmtLib to global scope.

Note: If the format library that you are promoting already exists in global scope, specify REPLACE to replace it. Otherwise, an error occurs.

2 Use LISTFORMATS to verify that format library MyFmtLib now exists in global scope.

SAS Log

```
     cas casauto promotefmtlib fmtlibname=myFmtLib;
NOTE: Request to PROMOTEFMTLIB MYFMTLIB completed for
      session CASAUTO.

     cas casauto listformats;
NOTE: Fmtlib = MYFMTLIB
         Scope = Session
         Fmtsearch = YES
NOTE: Fmtlib = MYFMTLIB
         Scope = Global
         Fmtsearch = NA
NOTE: Request to LISTFORMAT completed for session CASAUTO.
```

Example 18: Drop a Format Library from Session Scope

Program

Drop format library MyFmtLib, which was promoted in "Example 17: Promote a Format Library to Global Scope" on page 51, from session scope.

```
cas casauto dropfmtlib fmtlibname=myFmtLib;  /* 1 */

cas casauto listformats;                     /* 2 */
```

1 Drop format library MyFmtLib from session scope. Format library MyFmtLib remains in the Casauto session format library search list.

2 Use LISTFORMATS to verify that format library MyFmtLib is no longer in session scope.

SAS Log

```
     cas casauto dropfmtlib fmtlibname=myFmtLib;
NOTE: Request to DROPFMTLIB MYFMTLIB completed for
      session CASAUTO.

     cas casauto listformats;
NOTE: Fmtlib = MYFMTLIB
         Scope = Global
         Fmtsearch = NA
NOTE: Request to LISTFORMAT completed for session CASAUTO.
```

Chapter 3

CAS LIBNAME Engine

CAS LIBNAME Engine Overview

SAS Cloud Analytic Services (CAS) is the analytic server and associated cloud services in SAS Viya. The CAS engine performs client/server communication. The engine is part of the SAS client and connects to the CAS server.

The CAS LIBNAME engine connects a SAS session to a CAS session. If there is not a pre-existing CAS session to which you can connect, the CAS engine creates a new session when you specify the CAS LIBNAME statement.

The libref is your link between SAS and the in-memory tables on the CAS server. When you assign a CAS engine libref, you are associating the libref with a CAS session in order to work with in-memory tables.

A libref is associated with the session that you specify in the SESSREF= statement option. If you do not specify the option, then the libref is associated with the session in the SESSREF= system option. The system option is updated when you start a session with the CAS statement.

If you start multiple CAS sessions, the SESSREF= option allows you to specify a session to associate with the libref. Otherwise, the libref is associated with a session according to the SESSREF= system option. You can view the setting with the following code:

```
proc options option=sessref;
    run;
```

By default, the libref uses the active caslib, which can change as caslibs are added and dropped. However, you can specify the CASLIB= LIBNAME option to bind the libref to a specific caslib. In this case, adding and dropping caslibs has no effect on the libref unless the bound caslib is dropped.

You can specify library options in the CAS LIBNAME statement or data set option in a DATA step or procedure step. When you specify a value for a data set option that has a corresponding CAS LIBNAME statement option (such as PROMOTE=) or a GLOBAL option, the data set option value takes precedence over the value for the CAS LIBNAME statement option.

To run SAS 9.4 procedures and the DATA step that use CAS processing or transfer data from CAS, you must do the following:

1. Connect to a server and start a CAS session.
2. Specify the CAS engine LIBNAME statement and use the libref with the input table name. When you assign a CAS libref, the CAS LIBNAME engine associates the libref with the active caslib unless you specify the LIBNAME statement CASLIB= option. If you use the CASLIB= option, the CAS libref is bound to the caslib.

   Here are two examples of the CAS LIBNAME statement:

   ```
   /* The libref Mycas is associated with the active caslib.*/
   /* If a new caslib is added, Mycas is associated with the new caslib.*/

   libname mycas cas;

   /* In the following statement, the libref Mycas is bound to the testTables caslib,
   /* even if the active caslib changes to a caslib other than testTables.  */
   ```

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

3. Execute a procedure or DATA step.
4. At the end of your program, when you no longer need to access data in CAS, you can use the following statement to terminate your session to preserve resources:

```
cas casauto terminate;
```

Note: See *Base SAS Procedures Guide* for specific information about working with procedures in CAS.

Use of WHERE with the CAS LIBNAME Engine

WHERE processing enables you to conditionally select a subset of rows so that the software processes only the rows that meet specified conditions. If you use the WHERE= data set option or WHERE statement with the CAS LIBNAME engine, the engine will attempt to resolve the WHERE expression on the CAS server. Performance is often improved with large tables when the WHERE expression qualifies only a relatively small subset.

There are WHERE clause elements that cannot be handled by CAS, so the WHERE evaluation happens in SAS instead.

Note: You can set system option MSGLEVEL=i to view messages about CAS processing in the SAS log.

Working with Temporary Columns

The CAS LIBNAME engine enables users to create temporary columns. Temporary columns are useful when preparing and exploring data because you can rapidly revise the expressions that create the temporary columns. The alternative is to create in-memory tables with permanent columns. In this case you might be creating additional in-memory tables that consume resources. These tables might require additional programming statements to drop them if they do not lead to better analytical insights.

The TEMPNAMES= option and the SCRIPT= option are required arguments for working with temporary columns. The following SAS code defines three variables in a simple script and captures it in the NewCols file reference. The names of the variables that you can add to the mycas.cars table are then listed in the TEMPNAMES= data set option. It appears that there is some duplication of information, because t1 through t3 are defined in the SAS script and are listed in the TEMPNAMES= option. However, this is only because it is a very simple example. Scripts can be very complicated, using hundreds of variables in assignments and expressions. Only the variables listed in the TEMPNAMES= option are added to the input table, which conserves resources in the server.

If you want to add a temporary character column to the input table, you must follow the name of the variable in the TEMPNAMES= option with a dollar sign ($) and the variable length in bytes.

```
cas casauto host="cloud.example.com" port=5570;
proc casutil;
  load data=sashelp.cars(where=(type ne 'Hybrid')) replace;
```

```
quit;

filename newcols temp;                                          /*1*/

data _null_;                                                    /*2*/
   file newcols;
   put "t1 = round(mpg_highway/mpg_city,0.5);";
   put "t2 = round(cylinders/enginesize,0.2);";
   put "t3 = msrp / invoice;"

run;

libname mycas cas sessref=casauto;

proc mdsummary data=mycas.cars(tempnames=(t1 t2 t3)    /*3*/
  script=newcols);
   output out=mycas.carssum;
   var mpg_city mpg_highway t1 t2 t3;
run;

proc print data=mycas.carssum;
  var _column_ -- _sum_;
run;
```

1 Instead of naming the location of the file, the code shows the use of TEMP, which means that the values are stored in a temporary file. A path can be specified instead of using TEMP.

2 A NULL DATA step is used to write three expressions to the temporary file. As an alternative, you can enter the expressions in a text file and specify the path instead of using the TEMP device with the FILENAME statement.

3 The MDSUMMARY procedure produces summary statistics for the MPG_CITY and MPG_HIGHWAY variables in addition to the t1, t2, and t3 variables listed in the TEMPNAMES= Data Set option. The OUTPUT statement creates an in-memory table named mycas.carssum. The table includes the variables from the TEMPNAMES= Data Set option.

The "ONDEMAND= Data Set Option" on page 81 is optional for working with temporary columns. ONDEMAND= enables users to customize how their data is evaluated: one row at a time or collectively at the outset.

Getting Started

The CAS engine makes it possible to run SAS 9.4 procedures, SAS Viya procedures, and an in-memory DATA step. You must start a session with the CAS server and create a CAS LIBNAME engine libref before the following types of data transfers can occur between SAS and the CAS server:

- reading SAS data sets from a loaded table and writing to a caslib as an in-memory table
- transferring CAS table data or table metadata as input to a DATA step or SAS procedure when the DATA step or procedure executes in SAS.

The example below demonstrates a few key points about the engine:

- The CAS engine is the key method for performing an in-memory DATA step.
- The CAS engine is required when using a SAS procedure to transfer data from the CAS server.
- SAS Viya analytic procedures process in-memory tables with CAS actions, so the engine provides the way to identify the tables to use for analysis.
- The engine provides one way to perform data transfer between SAS and the server and is appropriate for smaller data sets. For larger data sets, the CASUTIL procedure can be used to perform a server-side load.

```
option casport=5570 cashost="cloud.example.com";                    /*1*/

cas casauto sessopts=(caslib=casuser);                              /*2*/

libname mycas cas;                                                  /*3*/

proc casutil;                                                       /*4*/
   load data=sashelp.cars casout="cars";
quit;

data mycas.bopc;                                              /*5*/
   set mycas.cars;
   where make in ("Buick", "Oldsmobile", "Pontiac", "Cadillac");
      wthp_ratio = round((weight / horsepower), .0001);
   drop msrp invoice;
run;

proc print data=mycas.bopc(obs=5);                                  /*6*/
run;

proc treesplit data=mycas.bopc;                                     /*7*/
   model mpg_highway = cylinders enginesize wheelbase length;
   class type drivetrain;
run;
```

1. Specify the connection information. The CASHOST= and CASPORT= system options are used to specify the connection information. As an alternative, your administrator might have set the default host and port for your server in a configuration file.
2. Start a session and set the active caslib to Casuser. You can start a session manually with the CAS statement. In this example, the CAS statement starts the CAS session named CASAUTO.
3. The LIBNAME statement assigns a CAS engine libref named Mycas, that you use to connect to the session CASAUTO. In subsequent procedure steps, input table names must begin with the Mycas libref.

   Some procedures require output table names to also begin with the CAS libref. See the documentation for your procedure for more information. SAS DATA step programs require a CAS engine libref on both the input and output table names.
4. Add the Sashelp.cars data set as an in-memory table.
5. Run an in-memory DATA step to create a new table with a subset of the rows and columns and add a new column. The input and output tables must begin with the CAS engine libref Mycas.

6. View five rows from the new data. The engine performs data transfer from the CAS server to SAS.
7. Use the Mycas libref to identify the input table for a SAS Viya procedure.

Data Types

Data Type Definitions

The CAS engine supports the storage of three data types. The following table provides information for each type:

Data Type	Description	Example	Missing Value

Fixed Length Character
CHAR(*n*)
	Stores a fixed-length character string. *n* is the maximum number of bytes to store. The maximum number of bytes is required to store each value regardless of the actual size of the value.
	Range: 1 - 32767 bytes
		`data mycas.greets;`
		`  length greet1 $16;`
		`  greet1 = "Good day";`
		`run;`
		`cas data greets;`
			all blanks

Varying Length Character
VARCHAR(*n*) **
	Stores a varying-length character string. *n* is the maximum number of characters to store.
	Range 1 - 536,870,911 characters (UTF-8 encoding)
		`data mycas.info;`
		`  length lastname varchar(40);`
		`  lastname="Adams";`
		`run;`
			all blanks, or zero length

DOUBLE (numeric)	Stores a numeric value, including dates and times, as a floating-point number.
	Range: 8 bytes for the CAS Engine*
		`data mycas.info;`
		`  length count 8;`
		`  count = 25;`
		`run;`
			Any one of the following forms:
			`.`
			`.A-.Z`
			`._`

* If a shorter length is specified, then 8 bytes are used and a note is printed to the log.

** The CAS engine is required for storing VARCHAR variables in the output table of a DATA step. The DATA step can *read* CAS tables containing VARCHAR variables, but it cannot store them unless a CAS engine libref is specified on the output table.

The SAS V9 engine supports only the CHAR and NUMERIC types. For more information about data types supported by the SAS V9 engine, see "SAS Variables" in *SAS Language Reference: Concepts*.

Support for Implicit Declaration of Data Types

The CAS engine supports implicit data type declaration for DOUBLE (NUMERIC) and CHAR type variables.

Implicit declaration means that you do not have to explicitly declare a variable's type or length before using it. You can create a new variable and use it for the first time in an assignment statement without having to explicitly declare its type or length. When you

create a variable this way, SAS determines the type based on the values that you assign to the variable.

- Variables that are assigned a character string value are implicitly defined as a CHAR types with a default length of 8 bytes.
- Variables that are assigned an integer value are implicitly defined as DOUBLE types with a default length of 8 bytes.

  Note: This is different from the V9 engine, which supports a length range of 1 - 8 bytes for NUMERIC types.
- Implicit type declaration is not supported for VARCHAR variables. VARCHAR variables must be explicitly declared in either a LENGTH statement or an ARRAY statement.

In the following DATA step, the type and length for variables **x** and **y** are set implicitly by SAS:

```
data test;
   x=1;
   y='hello';
run;
```

For complete information about data types supported by the SAS V9 engine, see "SAS Variables" in *SAS Language Reference: Concepts*.

VARCHAR Data Type

Definition

VARCHAR(*length* | *)

Stores a varying-length character string. *length* is the user-defined maximum length of the character string, and * indicates that the maximum storage size is 536,870,911 characters.

Syntax

```
LENGTH variable-naame) VARCHAR(length|*);
ARRAY arrayname[N] VARCHAR(length|*);
ARRAY arrayname[*] VARCHAR-variables;
```

variable-name
specifies one or more variables that are assigned the type VARCHAR.

length
specifies a numeric constant that is the user-defined maximum number of characters stored in the VARCHAR variable. This value can be up to 536,870,911 characters in length. Uninitialized VARCHAR variables are given a length of 1 by default. This value is based on the defined range.

```
length xyz varchar(32);
```

Uninitialized VARCHAR variables are given a length of 1 by default. 1 is the minimum length of a VARCHAR variable.

Range 1 to 536,870,911 (UTF-8) characters (2^{31} bytes)

See "PROC CONTENTS Output for Actual and Defined Length" on page 61

*

specifies the maximum length allowed, which is 536,870,911 characters. When assigning a character constant to a VARCHAR variable, the character constant is limited to 32767 bytes.

```
length xyz varchar(*);
```

Uninitialized VARCHAR variables are given a length of 1 by default. This value is based on the defined range for VARCHAR variables, in which 1 is the minimum length a VARCHAR variable can be.

See "PROC CONTENTS Output for Actual and Defined Length" on page 61

array-name

specifies the name of the array. Defines the elements in an array as a list of VARCHAR variables.

When using a list of VARCHAR variables with the ARRAY statement, you can use the hyphen (–), colon / prefix, and double-dash lists:

```
array arr1[*] v1-v5;
array arr2[*] v:;
array arr3[*] v1--v5;
```

You cannot use VARCHAR character lists specified as _CHARACTER_.

N

describes the number and arrangement of elements in the array

*

specifies the maximum length allowed, 536,870,911 characters. When assigning a character constant to a VARCHAR variable, the character constant is limited to 32767 bytes.

```
array myArray{*} varchar(*) a1 a2 a3 ('a','b','c');
```

Requirement requires a CAS engine libref on the output table

Engine CAS engine only

Note If you have sites that support multiple languages, consider running your SAS session using UTF-8 encoding and using the VARCHAR data type to minimize character conversion issues.

Details

The VARCHAR type is a varying length character data type whose length represents the maximum number of characters you want to store. VARCHAR variables have the following characteristics:

- their length is measured in terms of characters rather than bytes
- their length varies depending on the values present

These characteristics are in contrast to those of the CHAR data type whose length is measured in terms of bytes and whose length is fixed.

For example, a VARCHAR(20) can store *up to* 20 characters, but the actual storage used depends on the lengths of the individual values in the column.

For example, if a VARCHAR variable, **columnName**, is defined as a VARCHAR(10) variable, this means that it can store up to 10 characters. But because the value "hello" contains only 5 characters and each character uses one byte, only 5 bytes of memory are

allocated for the variable. A fixed-length CHAR column, on the other hand, takes up the defined number of bytes regardless of the actual size of the column.

PROC CONTENTS Output for Actual and Defined Length

When PROC CONTENTS is run against a CAS table that contains a VARCHAR variable, additional length information is displayed in the output. This information is displayed only for CAS tables that contain VARCHAR variables.

Figure 3.1 *A CAS Table Showing Detailed Length Information and Max Bytes Used*

Alphabetic List of Variables and Attributes

#	Variable	Type	Len		Max Bytes Used
			Bytes	Chars	
1	vc	Varchar	.	.	1
2	vc2	Varchar	100	100	8

The **Max Bytes Used** column displays the *actual length* of the data entered, or, the actual width of the column based on the values that it contains. The *actual length* of a VARCHAR variable is the storage size of the data entered, or, the actual width of the column based on the data that it contains. The column data in each row is adjusted to fit the width of the data that is in that individual row. If a VARCHAR variable is uninitialized, this column displays the default minimum length of 1.

The **Length** column displays the *defined maximum length* of a VARCHAR variable in both bytes and in characters. The *defined maximum length* of a VARCHAR variable is defined by the user using the VARCHAR(*n*) syntax or it is defined by SAS when you specify the VARCHAR(*) syntax.

For VARCHAR variables that are created using the system-defined maximum length, VARCHAR(*), the maximum allowable length is expressed as (.) in the **Len** column for both the **Bytes** and **Chars** sub-columns.

Here is the code that produces the PROC CONTENTS output that is shown above:

```
data mycas.new;
   length vc varchar(*);   /* uninitialized */
   length vc2 varchar(100);
   vc2 = "12345678";
run;

proc contents data=mycas.new; run;
```

Examples: Creating VARCHAR Variables

- **Example 1:** Create a VARCHAR Using the LENGTH Statement

```
libname mycas cas;
data mycas.roman;
   length vc32 varchar(32);
      do i = 1 to 10;
         vc32 = put(i, ROMAN.);
         output;
      end;
run;
```

- **Example 2:** Create a VARCHAR Using the ARRAY Statement

```
data mycas.test;
   array test{*} varchar(*) a1 a2 a3 ('a','b','c');
   put test[1]; put test[2]; put test[3];
run;
```

When to Use a VARCHAR Data Type

The VARCHAR data type is useful because it can save space when the lengths of the column values vary. With fixed-width data types, any space that is not used by the value in the column is padded with trailing blanks, which wastes space. The entire space is blocked out in memory whether the value needs the space or not. With varying-length data types, such as VARCHAR, only the space that is needed is used (there are no trailing blanks).

- CHAR – use a fixed-width CHAR when the sizes of the column data entries are similar. Fixed-width columns are usually accessed faster.
- VARCHAR – use when the sizes of the column data vary considerably but you are reasonably certain they will not exceed a certain width.
- VARCHAR(*) – use when the sizes of the column data vary considerably and the column width might exceed any limits you might place on it.

In most cases, you should take advantage of VARCHAR support. However, if values are consistently short, such as an ID column of airport codes, then a fixed-width CHAR variable uses less memory and runs faster. This is because VARCHAR values require 16 bytes plus the memory needed to store the VARCHAR value. So, if your values are always smaller than 16 bytes, you can save memory and processing time by using a CHAR type variable instead.

Range

SAS defines the length of a VARCHAR data type in terms of characters rather than bytes. The maximum length of a VARCHAR variable is 536,870,911 Unicode characters, or, 2^{31} bytes. This means that up to 536,870,911 characters, or 2,147,483,644 bytes of data or can be stored in a VARCHAR variable. The maximum length in bytes is calculated by multiplying 536,870,911 by the maximum length that any one character in the UTF-8 character set can be, which is 4 bytes.

Setting Missing VARCHAR Variables

You can use the CALL MISSING routine to set a VARCHAR variable to missing:

```
if var1 = "abc" then call missing(var1);
```

You can use the MISSING function to test whether a VARCHAR variable is missing.

```
if missing(var2) then var2 = "missing";
```

VARCHAR Support for Implicit and Explicit Data Type Conversion

Type conversion happens when a SAS DATA step or procedure moves data from a CAS table into a SAS data set. The data must be converted from the data type supported by the CAS engine to a data type supported by the SAS V9 engine.

The DATA step supports the processing of VARCHAR data. However, only the CAS engine supports the VARCHAR data type. This means that the DATA step can read in and process VARCHAR data, but the data is converted to a CHAR when it is stored as a SAS data set.

When you convert between CAS data and SAS data, the supported data type conversions are defined by the engine.

If character strings declared as VARCHAR data types are converted to the CHAR data type, values that are too long for the CHAR data type are truncated.

Data types can be converted from one type to another either implicitly or explicitly.

In *implicit conversions* SAS automatically converts data from one type to another and the conversions are not visible to the user. An example is when you save a CAS table containing a VARCHAR as a SAS data set. The VARCHAR is implicitly converted to a CHAR in the output data set.

In *explicit conversions*, users deliberately convert one type to another using programming statements.

SAS language elements that can explicitly convert one data type to another are the PUT function and the INPUT function. The following converts the numeric value of a VARCHAR to a DOUBLE data type and writes the output to a CAS table.

```
data mycas.new;
   length vc varchar(40);
   vc = '5000';
   num = input(vc,8.);
run;
proc contents data=mycas.new; run;
```

Output 3.1 *PROC CONTENTS Output Showing Explicit Data Type Conversion*

Alphabetic List of Variables and Attributes

#	Variable	Type	Len		Max Bytes Used
			Bytes	Chars	
2	num	Num	8		
1	vc	Varchar	40	40	4

VARCHAR Length with Implicit Type Conversion

VARCHAR variables are not supported by the SAS V9 engine. SAS automatically converts VARCHAR variables to CHAR variables if you try to store them in anything other than a CAS table.

When converting a variable from a VARCHAR to a CHAR, the length of the CHAR depends on how the VARCHAR is originally defined.

- VARCHAR(*) – If a table that contains a VARCHAR(*) definition is saved as a SAS data set, the VARCHAR is automatically converted to a CHAR with an assigned length of 32767.
- VARCHAR(*n*) – If a table that contains a VARCHAR(*n*) definition is converted to a SAS data set, then the length of the variable depends on the local SAS session encoding. The length is calculated as follows: SAS multiplies the current length of the VARCHAR by the maximum value that a character's length can be in the local SAS session encoding.
  - If the local SAS session encoding uses single-byte characters, then the VARCHAR is converted to a CHAR with a length of (*n* x 1). *n* is the length of the original VARCHAR and 1 is the largest number of bytes required to store any character in the character set.
  - If the local SAS session encoding uses double-byte characters, then the VARCHAR is converted to a CHAR with length (*n* x 2). *n* is the length of the

original VARCHAR and 2 is the largest number of bytes required to store any character in the character set.

- If the local SAS session encoding uses UTF-8 encoding, then the VARCHAR is converted to a CHAR with length (*n* x 4). *n* is the length of the original VARCHAR and 4 is the largest number of bytes required to store any character in the character set.

Note: When assigning a character constant to a VARCHAR variable, the character constant is limited to 32767 bytes.

See "Example 3" in *SAS Cloud Analytic Services: DATA Step Programming* for more information.

Restrictions for the VARCHAR Data Type in the CAS Engine

Not all SAS language elements support the VARCHAR data type, even with the CAS engine. There are also differences in how some SAS language elements behave with VARCHAR variables. These limitations and behavior differences are listed in the table below.

Table 3.1 *Restrictions and Notable Behaviors for the VARCHAR Data Type in the CAS Engine*

Feature	Description
ATTRIB	You cannot use the ATTRIB statement to create VARCHAR variables.
BY statement	The BY statement uses a fixed width for VARCHAR variables. Using a VARCHAR(*) type in the BY statement might cause unexpected results.
Formats	The width of VARCHAR formats is measured in bytes rather than characters.
Functions	When passing a character value to a function, numbers indicating a length or position that are passed to the function or returned by the function are in units of bytes. When passing a VARCHAR variable to functions, these numbers are in units of characters. Of note, this includes the INDEX and SUBSTR functions. See "Index CHAR and VARCHAR Character Strings" in *SAS Cloud Analytic Services: DATA Step Programming* for a related example.
KEY= on SET and MODIFY statements	VARCHAR variables are not supported by the KEY= option in either the SET or MODIFY statements.
PUT statement (to ODS output)	VARCHAR variables are not supported with the PUT statement when the DATA step writes output using ODS.

VARCHAR	The VARCHAR data type is supported by the CAS engine but not by the V9 engine. This means that to create or store a VARCHAR variable, you must use the CAS engine. The SAS DATA step (with the V9 engine) can read data containing VARCHAR variables but it converts and stores them as CHAR data types.
Variable Lists	Selecting a character variable range for character variable lists (for example, *a-character-f*) is not supported for VARCHAR variables because VARCHAR variables are not fixed-width character variables. For example, you cannot specify VARCHAR variables using the following shorthand forms: _CHARACTER_ *var1*–CHARACTER-*varN*

Concatenating variables	When concatenating character values, the result is a character value that is limited to 32767 bytes. In this example, the character result of the concatenation is assigned to a VARCHAR variable and the result value is limited to 32767 bytes.
	`length vc varchar(*);`
	`vc = "a string that is 32000 bytes long"`
	`  || "another string that is 32000 bytes long";`
	To go beyond the 32767 byte limit, include a VARCHAR variable in the concatenation. The concatenation result is a VARCHAR that can go beyond 32767 bytes.
	`length vc1 vc2 varchar(*);`
	`vc1 = "a string that is 32000 bytes long";`
	`vc2 = vc1 || "another string that is 32000 bytes long";`

Variable Names and Data Set Names in CAS Engine

The rules for naming variables and tables when running the DATA step in SAS and in CAS are as follows:

- Names can be up to 32 bytes in length.
- Names can contain alphanumeric characters and the underscore (_) character.
- Names can contain special characters **/ \ * ? " < > | : -** if they are specified as name literals (in quotation marks followed by n).

Here is an example showing these rules:

```
data _null_ / sessref=casauto;
      greater_than_8_but_less_than_32 = 100;
      'blanks & special chars'n = 'hello';
      put greater_than_8_but_less_than_32
      put 'blanks & special chars'n;
run;
```

Note: These rules are identical to the rules for names in SAS 9.4 when system option VALIDVARNAME=ANY and system option VALIDMEMNAME=EXTEND.

Dictionary

CAS LIBNAME Statement

Associates a SAS libref with in-memory tables on the CAS server.

Category: Data Access

Data type: CHAR, VARCHAR, and DOUBLE

Tip: If you do not use the default, these connections are valid: HOST= and PORT= (together), SESSREF=, UUID=, or UUIDMAC=.

Syntax

Form 1: **LIBNAME** *libref* CAS <*options*>;

Form 2: **LIBNAME** *libref* CAS **HOST=** "*controller-host-name*" **PORT=***network-port-number* <*options*>;

Form 3: **LIBNAME** *libref* CAS **SESSREF=***session-name* <*options*>;

Required Argument

CAS
specifies the CAS LIBNAME engine.

Optional Arguments

CASLIB=*caslib-reference-name*
specifies the name of a caslib to bind to the libref. The active caslib for your session can change as a result of adding or dropping caslibs. When you specify this option, an engine operation using the libref uses the caslib that you specify, regardless of the active caslib for the session. You can override this option with the CASLIB= data set option.

HOST="*controller-host-name*"
specifies the host name of the CAS controller. By default, if you started a CAS session, the host name is determined from the session. If you have not started a session, the host name is determined as follows:

1. the value of the CASHOST= system option
2. the _CASHOST_ macro variable
3. the server uses localhost

libref
specifies a valid SAS name that serves as a shortcut name to associate with the tables on the CAS server. The name must conform with SAS naming conventions. It can be up to eight bytes long.

PORT=*network-port-number*
specifies an integer for the port where the CAS controller listens for client connections. Otherwise, the engine checks for a value in the CASPORT= system option and then in the _CASPORT_ macro variable.

SESSREF=*session-name*
specifies the name of a CAS session to which you want to connect. Using this option is a preferred alternative to HOST= and PORT=.

UUID="*session-uuid*"
specifies the universally unique identifier (UUID) of an existing session. You must obtain the UUID from the existing session before you can specify it in this option. The engine connects to the session that is identified in the UUID.

Tip Most people prefer to use SESSREF= instead of UUID=.

Example This example starts a session with the CAS statement and saves the UUID in the MyUUID macro variableA libref is assigned and connects to that session.

```
cas casauto host="cloud.example.com" port=5570 uuid=MyUUID;

< Perform some additional tasks. >
libname mycas cas uuid="&MyUUID";

data mycas.female;
    set sashelp.class;
    where sex='F';
run;
```

UUIDMAC=*macro-variable-name*
specifies a SAS macro variable. The engine saves the UUID of the CAS session that is started or connected to. This option is useful if you want a subsequent SAS step to connect to a session that the engine started or modified. If you do not specify this option, the UUID is stored in the _IOCASUUID_macro variable value.

Example: Execute a DATA step

Once you have assigned a CAS engine libref, you can run a DATA step as you would for any other data source. The DATA step reads data and loads it into a CAS table.

In the example below, the DATA step reads data from a table in a Base engine library and writes data to a CAS table using the CAS engine library.

```
/*Start a CAS session.*/
cas casauto host="cloud.example.com" port=5570;

/*Assign a libref. THe engine finds the CASAUTO session from the SESSREF system option.*
libname mycas cas;

/*Use the engine to transfer data from SAS to the server.*/
 data mycas.cars;
    set sashelp.cars;
 run;

/*Run Data step in CAS to subset data.*/
data mycas.somecars;
     set mycas.cars;
     by make model;
```

```
     keep make model weight origin;
run;

/* Print output from in-memory table. */
proc print data=mycas.cars2;
run;
```

See Also

- CASLIB statement,
- COMPRESS= LIBNAME option

CASLIB= LIBNAME Option

Specifies the name of the caslib to use for engine operations in the LIBNAME statement.

Category: Data Access

Default: the active caslib

Syntax

CASLIB=*caslib-reference-name*

Required Argument

caslib-reference-name
specifies the name of the caslib. Caslib names must be unique within the session.

Details

The Active Caslib

The CASLIB= LIBNAME option binds the specified caslib to the libref. If you do not specify this option, then the libref uses the active caslib for your session. The active caslib can change as caslibs are added and dropped.

Comparisons

The CASLIB= data set option overrides the CASLIB= LIBNAME statement option.

The CASLIB= LIBNAME statement option overrides the CASLIB= session option.

See Also

- “CASLIB Statement”
- “CASLIB= Data Set Option”
- “CASLIB= Session Option”
- “CASLIB= System Option”

COMPRESS= LIBNAME Option

Requests that the CAS table is compressed.

Category: Data Access

Default: NO

Restrictions: When using this option with the CAS server, use YES and NO values only.
Use with output data sets only.

Syntax

COMPRESS=YES | NO

Required Arguments

YES
specifies that the CAS table is compressed.

NO
specifies that the CAS table is not compressed.

Comparisons

The COMPRESS= data set option overrides the COMPRESS= LIBNAME statement option.

See Also

"COMPRESS= Data Set Option"

DATALIMIT= LIBNAME Option

Specifies the maximum number of bytes of data that can be transferred between the CAS server and SAS during a single-table read.

Category: Data Access

Default: 100M

Restriction: This option affects only Read access.

Tip: This option can prevent you from accidentally transferring a large amount of data from the server to the client.

Syntax

DATALIMIT=*integer* | *integer*K | *integer*M | *integer*G | ALL

Optional Arguments

integer
specifies the maximum number of bytes to read.

***integer*K**
specifies the maximum number of kilobytes to read.

***integer*M**
specifies the maximum number of megabytes to read.

***integer*G**
specifies the maximum number of gigabytes to read.

ALL
specifies that the entire file can be read, no matter how large it is.

Comparisons

The DATALIMIT= data set option overrides the DATALIMIT= LIBNAME statement option.

The DATALIMIT= LIBNAME option overrides the CASDATALIMIT= system option.

See Also

- "CASDATALIMIT= System Option"
- "DATALIMIT= Data Set Option"
- "READTRANSFERSIZE= LIBNAME Option"
- "READTRANSFERSIZE= Data Set Option"
- "WRITETRANSFERSIZE= LIBNAME Option"
- "WRITETRANSFERSIZE= Data Set Option"

NCHARMULTIPLIER= LIBNAME Option

Specifies a multiplication factor that is used to increase the number of bytes for a fixed character variable when data is transcoded to the UTF-8 encoding in order to run in the CAS server.

Category: Data Access

Defaults: 1 for SBCS environments
1.5 for DBCS environments

Example: NCHARMULTIPLIER=1.5 expands an 8–byte field to 12 bytes

Syntax

NCHARMULTIPLIER=*n*

Required Argument

n
specifies the multiplication factor to use when transcoding

Range $0 < n \leq 4$

Comparisons

The NCHARMULTIPLIER= data set option overrides the NCHARMULTIPLIER= LIBNAME statement option.

The NCHARMULTIPLIER= LIBNAME statement option overrides the CASNCHARMULTIPLIER= system option.

See Also

- "CASNCHARMULTIPLIER= System Option"
- "NCHARMULTIPLIER= Data Set Option"

MAXTABLEMEM= LIBNAME Option

Specifies the maximum amount of memory, in bytes, to allocate for an in-memory table.

Valid in: CAS LIBNAME statement

Category: Data Access

Default: 16M

Syntax

MAXTABLEMEM=*integer* | *integer*K | *integer*M | *integer*G

Required Arguments

integer
specifies the total number of bytes to allocate.

***integer*K**
specifies the total number of kilobytes to allocate.

***integer*M**
specifies the total number of megabytes to allocate.

***integer*G**
specifies the total number of gigabytes to allocate.

Details

Integer values are always converted to the nearest whole megabyte but not less than 1 megabyte. Specifying 0 indicates to use the value from the MAXTABLEMEM= session option.

Comparisons

The MAXTABLEMEM= data set option overrides the MAXTABLEMEM= LIBNAME statement option.

Note: The value for either the data set option or the LIBNAME statement option is limited to the value for the MAXTABLEMEM= session option.

See Also

- "MAXTABLEMEM= Data Set Option"
- "MAXTABLEMEM= Session Option"

READTRANSFERSIZE= LIBNAME Option

Specifies the maximum number of bytes that the CAS engine reads when transferring data from the CAS server.

Category: Data Access

Alias: RTS

Default: 500MB

Restriction: This option affects only Read access.

Syntax

READTRANSFERSIZE=*integer* | *integer*K | *integer*M | *integer*G

Required Arguments

integer
specifies the total number of bytes to read.

***integer*K**
specifies the total number of kilobytes to read.

***integer*M**
specifies the total number of megabytes to read.

***integer*G**
specifies the total number of gigabytes to read.

Details

If the entire result of the read request is smaller than the value of the READTRANSFERSIZE= option, only the necessary number of bytes are transferred. This situation can occur if either the table size or the value of the DATALIMIT= option is smaller than the value of the READTRANSFERSIZE= option.

The following scenarios demonstrate different use cases for the READTRANSFERSIZE= option:

- Table size=45MB, DATALIMIT=100MB, READTRANSFERSIZE=500MB. The entire table is handled in a single read request because the table size is less than or equal to DATALIMIT= and READTRANSFERSIZE=.
- Table size=110MB, DATALIMIT=100MB, READTRANSFERSIZE=500MB. Only 100MB of the table is read and handled in a single read request. Because DATALIMIT= is smaller than the table size and smaller than READTRANSFERSIZE=, this error results:

```
ERROR:  The maximum allowed bytes of data have been fetched
        from SAS Cloud Analytic Services.
```

If you encounter this error, you can summarize the data in the server first and then transfer the smaller summarized data set to SAS. Or you can increase this setting so that the large data transfer can proceed without error.

- Table size=2GB, DATALIMIT="ALL", READTRANSFERSIZE=500MB. The entire table is handled in four read requests because the table size is equal to DATALIMIT= and greater than READTRANSFERSIZE=.

Comparisons

The READTRANSFERSIZE= data set option overrides the READTRANSFERSIZE= LIBNAME statement option.

See Also

- "DATALIMIT= LIBNAME Option" on page 69
- "DATALIMIT= Data Set Option" on page 78
- "READTRANSFERSIZE= Data Set Option" on page 85
- "WRITETRANSFERSIZE= LIBNAME Option" on page 74
- "WRITETRANSFERSIZE= Data Set Option" on page 88

TAG= LIBNAME Option

Specifies a multi-level prefix for a CAS table that is used to identify the table to both the CAS server and SAS.

Category: Data Access

Syntax

TAG=*tag-name*

Required Argument

tag-name
specifies a user-defined string to use for constructing table names.

Details

The TAG= option allows a SAS session to access CAS tables that have multi-level names without requiring the global VALIDMEMNAME=EXTEND option. A SAS table is identified by its libref and a table name (for example, mycas.mytab). Table names in SAS Cloud Analytic Services can have more than two levels (for example, lev1.lev2.mytab).

If you use this option, SAS can access only the in-memory tables that are prefixed with the value that you specified.

The value of the engine's TAG option is the CAS table name up to a delimiting period. The CAS server knows the table by the full name, 'lev1.lev2.mytab', but the SAS session knows the table name only as 'mytab'. When the TAG option is used, the CAS engine produces the correct name for both the SAS session and the CAS server session.

Example

This example uses the TAG= option to specify that the tag name, 'lev1.lev2', prefixes to the table name, 'mytab'. The engine uses the table name "lev1.lev2.mytab" in the CAS actions that are sent to the server.

```
libname mycas cas tag="lev1.lev2";
proc print data=mycas.mytab;
run;
```

See Also

"TAG= Data Set Option" on page 86

TRANSCODE_FAIL= LIBNAME Option

Specifies how the engine handles transcoding errors.

Valid in: DATA and PROC steps

Category: Data Access

Default: Error

Syntax

TRANSCODE_FAIL=ERROR | WARN | SILENT

Required Arguments

ERROR
Writes an error message to the SAS log. Processing stops.

WARN
Writes a warning message to the SAS log. Processing continues.

SILENT
Transcoding errors are ignored. Processing continues.

Comparisons

The TRANSCODE_FAIL= data set option overrides the TRANSCODE_FAIL= LIBNAME statement option.

See Also

"TRANSCODE_FAIL= Data Set Option" on page 87

WRITETRANSFERSIZE= LIBNAME Option

Specifies the maximum number of bytes that the CAS engine writes when transferring data to the CAS server.

Category: Data Access

Alias: WTS

Default: 512KB

Restriction: This option affects only Write access.

Syntax

WRITETRANSFERSIZE=*integer* | *integer*K | *integer*M | *integer*G

Required Arguments

integer
specifies the total number of bytes to read.

***integer*K**
specifies the total number of kilobytes to read.

***integer*M**
specifies the total number of megabytes to read.

***integer*G**
specifies the total number of gigabytes to read.

Comparisons

The WRITETRANSFERSIZE= data set option overrides the WRITETRANSFERSIZE= LIBNAME statement option.

See Also

- "DATALIMIT= LIBNAME Option" on page 69
- "DATALIMIT= Data Set Option" on page 78
- "READTRANSFERSIZE= LIBNAME Option" on page 72
- "READTRANSFERSIZE= Data Set Option" on page 85
- "WRITETRANSFERSIZE= LIBNAME Option" on page 74
- "WRITETRANSFERSIZE= Data Set Option" on page 88

APPEND= Data Set Option

Specifies whether to append rows from the DATA step and the PROC step output to the in-memory table.

Valid in: DATA step and PROC step

Category: Data Access

Default: NO

Restriction: You can set either the PROMOTE= on page 84 data set option or the APPEND= data set option to YES. An error results if you set both to YES at the same time.

Note: New observations are appended using the size of CHAR variables in the DATA step output table. If the DATA step output table does not exist, the rules for the NCHARMULTIPLIER= data set option are followed.

Syntax

APPEND=YES | NO | FORCE

Required Arguments

YES
specifies that the rows are appended to the CAS table.

NO
specifies that the rows are not appended to the CAS table.

FORCE
uses the CHAR length of the existing table to determine the length of newly appended observations.

CASLIB= Data Set Option

Specifies the name of the caslib to use for engine operations for the table.

Valid in: DATA step and PROC step

Category: Data Access

Default: the active caslib

Syntax

CASLIB=*caslib-reference-name*

Required Argument

caslib-reference-name
specifies the name of the caslib. Caslib names must be unique within the session.

Comparisons

The CASLIB= data set option overrides the CASLIB= LIBNAME statement option.

The CASLIB= LIBNAME statement option overrides the CASLIB= session option.

The CASLIB= session statement option overrides the CASLIB= system option.

See Also

- “CASLIB Statement” on page 91
- “CASLIB= LIBNAME Option” on page 68
- “CASLIB= Session Option” on page 285
- “CASLIB= System Option” on page 301

COMPRESS= Data Set Option

Requests that the CAS table is compressed.

Valid in: DATA step and PROC step

Category: Data Access

Default: NO

Restrictions: When using this option with the CAS server, use YES and NO values only.

Use with output data sets only.

Tip: If you specify both the COMPRESS= LIBNAME and data set option, the data set option has precedence.

Syntax

COMPRESS=YES | NO

Required Arguments

YES
specifies that the CAS table is compressed.

NO
specifies that the CAS table is not compressed.

Comparisons

The COMPRESS= data set option overrides the COMPRESS= LIBNAME statement option.

See Also

"COMPRESS= LIBNAME Option"

COPIES= Data Set Option

Specifies the number of copies for a redundant table.

Valid in: DATA step and PROC step

Category: Data Access

Default: 1

Restrictions: This option applies only to distributed servers.

The number of copies is limited to one less than the number of worker nodes in the server.

Syntax

COPIES=*integer*

Required Argument

integer
specifies the number of redundant copies for each block.

Details

SAS Cloud Analytic Services organizes data from tables in blocks. Redundant copies of blocks are stored in-memory across nodes. In the event of node failure, a surviving node accesses the data from the redundant block.

A large number of redundant block copies results in greater fault tolerance for node failures because redundant tables are distributed in memory on a greater number of nodes. COPIES=0 results in no fault tolerance in the event of a node failure.

DATALIMIT= Data Set Option

Specifies the maximum number of bytes of data that can be transferred between the CAS server and SAS during a single-table read.

Valid in: DATA step and PROC step

Category: Data Access

Default: 100M

Restriction: This option affects only Read access.

Tip: This option can prevent you from accidentally transferring a large amount of data from the server to the client.

Syntax

DATALIMIT=*integer* | *integer*K | *integer*M | *integer*G | ALL

Required Arguments

integer
specifies the maximum number of bytes to read.

***integer*K**
specifies the maximum number of kilobytes to read.

***integer*M**
specifies the maximum number of megabytes to read.

***integer*G**
specifies the maximum number of gigabytes to read.

ALL
specifies that the entire file can be read, no matter how large it is.

Comparisons

The DATALIMIT= data set option overrides the DATALIMIT= LIBNAME statement option.

The DATALIMIT= LIBNAME option overrides the CASDATALIMIT= system option.

See Also

- "CASDATALIMIT= System Option"
- "DATALIMIT= LIBNAME Option"
- "READTRANSFERSIZE= LIBNAME Option"

- "READTRANSFERSIZE= Data Set Option"
- "WRITETRANSFERSIZE= LIBNAME Option"
- "WRITETRANSFERSIZE= Data Set Option"

DUPLICATE= Data Set Option

Specifies whether the output table in the SAS Cloud Analytic Services is duplicated on all nodes.

Valid in: DATA step and PROC step

Category: Data Access

Default: NO

Restriction: This option applies only to distributed servers.

Interaction: The value for the COPIES= data set option is ignored if you specify the DUPLICATE= data set option.

Syntax

DUPLICATE=YES | NO

Required Arguments

YES
specifies that the output table is duplicated on all nodes.

NO
specifies that the output table is not duplicated on all notes

Details

The DUPLICATE= data set option results in a repeated table. (See "Data" in *SAS Cloud Analytic Services: Fundamentals* for more information.) A repeated table is different from a table with replicate copies of blocks for fault tolerance. You can enable fault tolerance by specifying a nonzero value in the COPIES= data set option. In a repeated table, all nodes have all rows of the table, and these rows are active everywhere.

MAXTABLEMEM= Data Set Option

Specifies the maximum amount of memory, in bytes, to allocate for an in-memory table.

Valid in: DATA step and PROC step

Category: Data Access

Default: 16M

Syntax

MAXTABLEMEM=*integer* | *integer*K | *integer*M | *integer*G

Required Arguments

integer
specifies the total number of bytes to allocate.

***integer*K**
specifies the total number of kilobytes to allocate.

***integer*M**
specifies the total number of megabytes to allocate.

***integer*G**
specifies the total number of gigabytes to allocate.

Details

Integer values are always converted to the nearest whole megabyte but not less than 1 megabyte. Specifying 0 indicates to use the value from the MAXTABLEMEM= session option.

Comparisons

The MAXTABLEMEM= data set option overrides the MAXTABLEMEM= LIBNAME statement option.

Note: The value for either the data set option or the LIBNAME statement option is limited to the value for the MAXTABLEMEM= session option.

See Also

- "MAXTABLEMEM= Data Set Option"
- "MAXTABLEMEM= Session Option"

NCHARMULTIPLIER= Data Set Option

Specifies a multiplication factor that is used to increase the number of bytes for a fixed character variable when data is transcoded to the UTF-8 encoding in order to run in the CAS server.

Valid in: DATA step and PROC step

Category: Data Access

Defaults: 1 for SBCS environments
1.5 for DBCS environments

Example: NCHARMULTIPLIER=1.5 expands an 8–byte field to 12 bytes

Syntax

NCHARMULTIPLIER=*n*

Required Argument

n
specifies the multiplication factor to use when transcoding

Range $0 < n \leq 4$

Comparisons

The NCHARMULTIPLIER= data set option overrides the NCHARMULTIPLIER= LIBNAME statement option.

The NCHARMULTIPLIER= LIBNAME statement option overrides the CASNCHARMULTIPLIER= system option.

See Also

- "CASNCHARMULTIPLIER= System Option"
- "NCHARMULTIPLIER= LIBNAME Option"

ONDEMAND= Data Set Option

Specifies how to evaluate temporary computed columns.

Valid in: DATA step and PROC step

Category: Data Access

Default: NO

Tip: On-demand execution is recommended when you transfer data from SAS Cloud Analytic Services to the SAS session, such as when the in-memory table is the input data of a DATA step or a procedure.

Syntax

ONDEMAND=YES | NO

Required Arguments

YES
specifies that temporary computed columns are evaluated one row at a time.

NO
specifies that temporary computed columns are evaluated collectively at the outset.

See Also

"Working with Temporary Columns" on page 55

ORDERBY= Data Set Option

Orders data hierarchically within a partition according to the specified variables.

Valid in: DATA step and PROC step

Category: Data Access

Default: Ascending order

Syntax

ORDERBY=(<DESCENDING> *variable-name-1* <,<DESCENDING> *variable-name-2 …*>)

Required Arguments

variable-name
specifies the variables by which to order the data within a partition.

DESCENDING
specifies that data values are arranged in descending order.

Note: Ascending order is the default.

Details

Ordering is hierarchical. For example, **`ORDERBY=(A B)`** indicates that the values of variable B are ordered within the ordered values of variable A. One or more specified variables must exist and cannot be partitioning variables. Order is determined based on the raw value of the variables and uses locale-sensitive collation for character variables.

Example

In this example, a SAS data set from the Sashelp library is loaded into a distributed CAS server using PROC CASUTIL. In the DATA step, the table is partitioned and then ordered within each partition using the PARTITION= and ORDERBY= data set options.

```
cas casauto sessopts=(caslib='casuser');                  /* 1 */
libname mycas cas;                                              /* 2 */

proc casutil outcaslib='casuser';                               /* 3 */
   load data=sashelp.baseball replace;
run;

data mycas.baseball (partition=(team) orderby=(nHome name))     /* 4 */
                    / sessref=casauto;
   set mycas.baseball;
   keep team name nHome;
run;
proc print data=mycas.baseball(where=(team eq 'Seattle'));      /*5*/
 run;
```

1. Create the session, Casauto. Specify the SESSOPTS= option in the CAS statement with the CASLIB= session option to ensure that the CASUSER personal caslib is set as the active caslib. Caslibs provide a way to access in-memory tables and an associated data source. In this example, the personal caslib CASUSER is being used, so no CASLIB statement is needed. For more information about the CASUSER and CASUSERHDFS personal caslibs, see Personal, Predefined, and Custom Caslibs in SAS Cloud Analytic Services: Fundamentals
2. Create a CAS engine libref.
3. Load the Sashelp.Baseball data set to a distributed CAS table.
4. Use the PARTITION= data set option to partition the table by team. This groups the rows containing the same value for Team together, onto the same server node. The ORDERBY= option then orders the rows within each partition by nHome and Name.

5. Use PROC PRINT with a simple WHERE expression that uses a partitioned column. The server efficiently uses the information because it can narrow the results to the partitioned column rather than evaluate every row with a full-table scan.

See Also

"PARTITION= Data Set Option"

PARTITION= Data Set Option

Partitions the output table according to the formatted values of the specified variables.

Valid in: DATA step and PROC step

Category: Data Access

Tip: You can specify `partition=()` to redistribute data equally across all worker nodes.

Syntax

PARTITION=(*variable-list*) | ( )

Required Argument

variable-list
specifies the list of partitioning variables for the output table.

Details

- Partitioning information is ignored when tables are opened for input.
- Errors result from partitioning by a variable that does not exist on output.
- Partition keys are derived based on formatted values as to how variable names are ordered in the variable list. Key construction is not hierarchical, so PARTITION=(A B) indicates that any unique combination of formatted values for A and B variables forms a partition of the data.
- Observations that share the same partition key are arranged together on the same worker node in SAS Cloud Analytic Services.

Example

In this example, a SAS data set from the Sashelp library is loaded into a distributed CAS server using PROC CASUTIL. In the DATA step, the table is partitioned and then ordered within each partition using the PARTITION= and ORDERBY= data set options.

```
cas casauto sessopts=(caslib='casuser');                /* 1 */
libname mycas cas;                                            /* 2 */

proc casutil outcaslib='casuser';                             /* 3 */
   load data=sashelp.baseball replace;
run;

data mycas.baseball (partition=(team) orderby=(nHome name))   /* 4 */
                    / sessref=casauto;
   set mycas.baseball;
```

```
   keep team name nHome;
run;
proc print data=mycas.baseball(where=(team eq 'Seattle'));   /*5*/
 run;
```

1. Create the session, Casauto. Specify the SESSOPTS= option in the CAS statement with the CASLIB= session option to ensure that the CASUSER personal caslib is set as the active caslib. Caslibs provide a way to access in-memory tables and an associated data source. In this example, the personal caslib CASUSER is being used, so no CASLIB statement is needed. For more information about the CASUSER and CASUSERHDFS personal caslibs, see Personal, Predefined, and Custom Caslibs in SAS Cloud Analytic Services: Fundamentals .
2. Create a CAS engine libref.
3. Load the Sashelp.Baseball data set to a distributed CAS table.
4. Use the PARTITION= data set option to partition the table by team. This groups the rows containing the same value for Team together, onto the same server node. The ORDERBY= option then orders the rows within each partition by nHome and Name.
5. Use PROC PRINT with a simple WHERE expression that uses a partitioned column. The server efficiently uses the information because it can narrow the results to the partitioned column rather than evaluate every row with a full-table scan.

See Also

ORDERBY= on page 81

PROMOTE= Data Set Option

Requests that the CAS table is added with global scope.

Valid in: DATA step and PROC step

Category: Data Access

Default: NO

Restrictions: You can set either PROMOTE= or APPEND= to YES. An error results if you set both to YES at the same time.

An error results if a global-scope table with the same name already exists. Existing global scope tables must be dropped or deleted before creating a new global table.

Requirement: The caslib target must also have global-scope .

Note: Global scope lets other sessions access the table, subject to access controls.

Syntax

PROMOTE=YES | NO

Required Arguments

YES
specifies that the table is added with global scope.

NO
specifies that the table is added with a session scope.

READTRANSFERSIZE= Data Set Option

Specifies the maximum number of bytes that the CAS engine reads when transferring data from the CAS server.

Valid in: DATA step and PROC step

Category: Data Access

Alias: RTS

Default: 500M

Restriction: This option affects only Read access.

Syntax

READTRANSFERSIZE=*integer* | *integer*K | *integer*M | *integer*G

Required Arguments

integer
specifies the total number of bytes to read.

***integer*K**
specifies the total number of kilobytes to read.

***integer*M**
specifies the total number of megabytes to read.

***integer*G**
specifies the total number of gigabytes to read.

Details

If the entire result of the read request is smaller than the value of the READTRANSFERSIZE= option, only the necessary number of bytes are transferred. This situation can occur if either the table size or the value of the DATALIMIT= option is smaller than the value of the READTRANSFERSIZE= option.

The following scenarios demonstrate different use cases for the READTRANSFERSIZE= option:

- Table size=45MB, DATALIMIT=100MB, READTRANSFERSIZE=500MB. The entire table is handled in a single read request because the table size is less than or equal to DATALIMIT= and READTRANSFERSIZE=.
- Table size=110MB, DATALIMIT=100MB, READTRANSFERSIZE=500MB. Only 100MB of the table is read and handled in a single read request. Because DATALIMIT= is smaller than the table size and smaller than READTRANSFERSIZE=, this error results:

  ```
  ERROR:  The maximum allowed bytes of data have been fetched
          from SAS Cloud Analytic Services.
  ```

  If you encounter this error, you can summarize the data in the server first and then transfer the smaller summarized data set to SAS. Or you can increase this setting so that the large data transfer can proceed without error.

- Table size=2GB, DATALIMIT="ALL", READTRANSFERSIZE=500MB. The entire table is handled in four read requests because the table size is equal to DATALIMIT= and greater than READTRANSFERSIZE=.

Comparisons

The READTRANSFERSIZE= data set option overrides the READTRANSFERSIZE= LIBNAME statement option.

See Also

- DATALIMIT= LIBNAME option,
- DATALIMIT= data set option,
- READTRANSFERSIZE= LIBNAME option,
- WRITETRANSFERSIZE= LIBNAME option,
- WRITETRANSFERSIZE= data set option

SCRIPT= Data Set Option

Specifies the file reference for the SAS script that defines the temporary computed columns.

Valid in: DATA step and PROC step

Category: Data Access

Alias: TEMPEXPRESS=

Syntax

SCRIPT=*fileref*

Required Argument

fileref
specifies the file reference for the SAS script that defines the temporary computed columns.

See Also

"Working with Temporary Columns" on page 55

TAG= Data Set Option

Specifies a multi-level prefix for a CAS table that is used to identify the table to both the CAS server and SAS.

Valid in: DATA step and PROC step

Category: Data Access

Syntax

TAG=*tag-name*

Required Argument

tag-name
specifies a user-defined string to use for constructing table names.

See Also

"TAG= LIBNAME Option" on page 73

TEMPNAMES= Data Set Option

Adds temporary computed column names to the input table.

Valid in: DATA step and PROC step

Category: Data Access

Length: Temporary columns can be character or numeric (8-byte doubles).

Restriction: Temporary computed columns are supported only for tables that are opened for input.

Syntax

TEMPNAMES=*variable-list*

Required Argument

variable-list
lists the names of the temporary columns that are added to the input table.

See Also

"Working with Temporary Columns" on page 55

TRANSCODE_FAIL= Data Set Option

Specifies how the engine handles transcoding errors.

Valid in: DATA step and PROC step

Category: Data Access

Default: Error

Syntax

TRANSCODE_FAIL=ERROR | WARN | SILENT

Required Arguments

ERROR
writes an error message to the SAS log. Processing stops.

WARN
writes a warning message to the SAS log. Processing continues.

SILENT
transcoding errors are ignored. Processing continues.

Comparisons

The TRANSCODE_FAIL= data set option overrides the TRANSCODE_FAIL= LIBNAME statement option.

See Also

"TRANSCODE_FAIL= LIBNAME Option" on page 74

WRITETRANSFERSIZE= Data Set Option

Specifies the maximum number of bytes that the CAS engine writes when transferring data to the CAS server.

Valid in: DATA step and PROC step
Category: Data Access
Alias: WTS
Default: 512K
Restriction: This option only affects Write access.

Syntax

WRITETRANSFERSIZE=*integer* | *integer*K | *integer*M | *integer*G

Required Arguments

integer
specifies the total number of bytes to read.

***integer*K**
specifies the total number of kilobytes to read.

***integer*M**
specifies the total number of megabytes to read.

***integer*G**
specifies the total number of gigabytes to read.

Comparisons

The WRITETRANSFERSIZE= data set option overrides the WRITETRANSFERSIZE= LIBNAME statement option.

See Also

- "DATALIMIT= LIBNAME Option" on page 69
- "DATALIMIT= Data Set Option" on page 78
- "READTRANSFERSIZE= LIBNAME Option" on page 72
- "READTRANSFERSIZE= Data Set Option" on page 85
- "WRITETRANSFERSIZE= LIBNAME Option" on page 74

Chapter 4

CASLIB Statement

Dictionary

CASLIB Statement

Adds and manages caslibs in a SAS Cloud Analytic Services session.

Restriction: You can add caslibs only if you are authorized to do so. See Adjust Caslib Management Privileges

See: For more examples of accessing data, see *SAS Cloud Analytic Services: User's Guide*.

For conceptual information about caslibs, see "Caslibs" in *SAS Cloud Analytic Services: Fundamentals*.

Syntax

Form 1: **CASLIB** *caslib-reference-name* <SESSREF=*session-reference*> DATASOURCE=(SRCTYPE="*type*" <*data-source-options*>) <PATH=*directory-path*><*option(s)*>;

Form 2: **CASLIB** *caslib-reference-name* LIST <SESSREF=*session-reference*>;

Form 3: **CASLIB** _ALL_ ASSIGN | _ALL_ LIST

Required Arguments

caslib-reference-name
specifies the name of the caslib. Names of session caslibs must be unique within the session. Names of global caslibs must be unique across all sessions within a server.

Range 1 to 256 characters

Tips Caslib names are case insensitive. If a session-scope caslib and a global-scope caslib have the same name, the session caslib is searched first.

ALL is a valid name with the LIST option and the ASSIGN option.

For naming rules see "Rules for Words and Names in the SAS Language" in *SAS Language Reference: Concepts*.

DATASOURCE= (SRCTYPE="*type*", <*data-source-options*> , <ENCRYPTIONPASSWORD="*string*">)

specifies data source options to use when connecting to a data source. The SRCTYPE="*type*" option specifies the data source type. Data sources can be either databases or path-based. The *data-source-options* syntax depends on the data source.

ENCRYPTIONDOMAIN="*string*"

specifies the name for a collection of data that is stored with a common encryption password.

Restrictions The ENCRYPTIONDOMAIN= option can be specified only with the DATASOURCE= option.

When you add a path-based caslib that supports the encryptionDomain and encryptionPassword parameters, you can specify one or the other. Use the encryptionDomain parameter to retrieve credentials automatically from an encryption domain that is managed with SAS Environment Manager.

ENCRYPTIONPASSWORD="*string*"

specifies a password for encrypting or decrypting stored data.

Restrictions The ENCRYPTIONPASSWORD= option can be specified only with the DATASOURCE= option.

When you add a path-based caslib that supports the encryptionDomain and encryptionPassword parameters, you can specify one or the other. Use the encryptionDomain parameter to retrieve credentials automatically from an encryption domain that is managed with SAS Environment Manager.

The following table shows the syntax, supported file types (if applicable), and an example for path-based data sources. For the data source options syntax for databases such as Hadoop and Oracle, see Chapter 9, "Data Connectors," on page 167.

Table 4.1 *Path-Based Data Source Types and Options*

SRCTYPE= Type	Option Syntax	Example	Supported File Types
PATH	DATASOURCE=(SRCTYPE =PATH, <ENCRYPTIONDOMAIN= "*string*>) PATH="*file-path*";	`caslib mycsvs path="/data/Myxlsxfiles/" datasource=(srctype="path", encryptiondomain=string);`	SASHDAT SAS7BDAT CSV XLS, XLSX
DNFS	DATASOURCE=(SRCTYPE =DNFS, <ENCRYPTIONDOMAIN= "*string*>) PATH="*file-path*";	`caslib mycsvs datasource=(srctype="dnfs") path="/data/Mycsvfiles/";`	SASHDAT CSV

SRCTYPE= Type	Option Syntax	Example	Supported File Types
HDFS	DATASOURCE=(SRCTYPE =HDFS, <ENCRYPTIONDOMAIN= "*string*">) PATH="*file-path*";	`caslib mycsvs path="/data/Mycsvfiles/" datasource=(srctype="hdfs", encryptiondomain=`*`string`*`);`	SASHDAT CSV

See For information about the PATH= option, see "PATH="*directory-path*"" on page 94.

Examples "Example 1: Add a Global Caslib" on page 96

"Example 5: Encrypt Tables in a Caslib" on page 104

Optional Arguments

ASSIGN

used with the _ALL_ option to assign SAS librefs for existing caslibs so that they are visible in the SAS Studio Libraries tree.

Restriction The CASLIB _ALL_ ASSIGN statement only assigns a libref for caslibs that follow SAS libref naming rules. For example, if the caslib has more than eight characters, then no libref is assigned for that caslib.

Requirement The ASSIGN option must be used with the _ALL_ option.

Tip When caslibs are dropped, the corresponding SAS library is not automatically cleared.

Examples The following statement assigns SAS librefs for all existing caslibs:

```
caslib _all_ assign;
```

The following statement assigns SAS librefs for all existing caslibs in the session named MySess:

```
caslib _all_ assign sessref=mysess;
```

ALL

specifies that the ASSIGN, or LIST argument applies to all currently added caslibs.

Example "Program 2: List All Caslibs" on page 98

CREATEDIRECTORY

creates a subdirectory in the caslib's data source. The path, up to the last directory, must already exist.

Alias CREATEDIR

Restriction Directory creation is available only for global caslibs.

Example The following CASLIB statement creates a new directory named Programs in the **/examples/caslibs/** directory:

```
caslib TestDir path="/examples/caslibs/programs"
```

```
datasource=(srctype="path") createdir global;
```

DESCRIPTION="*description*"
specifies a description of the data source.

Alias DESC=

DROP
drops a session scope caslib.

Alias CLEAR

Example
```
caslib caslibA drop;
```

```
NOTE: Cloud Analytic Services removed the caslib 'CASLIBA'.
NOTE: Action to DROP caslib CASLIBA completed for session CASAUTO.
```

GLOBAL
adds the caslib so that it has global scope. A global-scope caslib can have access controls set so that it is accessible from all sessions and can be a way to share data. Other connections to the server that get their own sessions have access to the caslib, subject to access controls. If you do not specify GLOBAL, the caslib is created with session scope. You must also grant access to the caslib in the CAS Server Monitor. You can add caslibs only if you are authorized to do so. See Adjust Caslib Management Privileges

See For information about caslibs scope, see "Caslibs Scope" in *SAS Cloud Analytic Services: Fundamentals*.

LIBREF=
creates a libref and associates it with a caslib. The libref can then be used to access data in SAS Cloud Analytic Services.

Example The following CASLIB statement creates the caslib Casdata and binds it to the libref Mycas.
```
caslib casdata path='/cas/mycasdata/' libref=mycas;
```

LIST
displays caslib names and their specifications. To display all caslibs, specify _ALL_ for the *caslib-reference-name*.

Tip Specify *caslib-reference-name* to list the settings for a single caslib. Specify _ALL_ to list all caslibs and their caslib settings. The specifications displayed by the LIST _ALL_ option are a subset. For full specification information, use the LIST option with a specific caslib specified.

Example "Example 2: List Caslib Settings" on page 97

NOTACTIVE
specifies that the caslib being added does not become the active caslib for the session.

PATH="*directory-path*"
specifies the fully qualified path to a directory to use as a data source.

Example The following CASLIB statement adds a caslib that accesses a path-based directory:

```
caslib mylib path="/local/data" description="Local data" ;
```

SESSION
adds the caslib so that it is session-scope. Other connections to the server that get their own sessions do not have access to the caslib. The lifetime of the caslib is the lifetime of the session. When you add caslibs, SESSION is the default.

Alias LOCAL

See "Caslibs Scope" on page 95

SESSREF=*session-reference*
specifies the name of the session to associate the caslib with. By default, the most recently started session is used.

SUBDIRS
specifies that subdirectories of the specified PATH= directory can be accessed with the caslib.

Tip You do not need to use the SUBDIRS option if the full path to the subdirectory is specified.

Details

What are Caslibs?

Caslibs are the mechanism for accessing data with SAS Cloud Analytic Services (CAS). They provide a volatile, in-memory space to hold tables, access controls, and data source information. Caslibs provide a way to organize in-memory tables and an associated data source. They also provide a way to apply access controls to data. A table within a caslib is a temporary, in-memory copy of the original data. All operations in SAS Cloud Analytic Services that use data are performed on tables within a caslib. Use the SAVE statement in the CASUTIL procedure to permanently save tables to a data source. For more information about the CASUTIL procedure, see Chapter 5, "CASUTIL Procedure," on page 109.

What Can I Do with the CASLIB Statement

You can do the following tasks with the CASLIB statement:

- add a caslib with access to files from the data source and access to in-memory tables that are read from the data source.
- specify the options to use when connecting to a data source.
- list the caslibs that are available to your session.
- view the caslib settings for one or more caslibs.

For conceptual information about caslibs, see "Caslibs" in *SAS Cloud Analytic Services: Fundamentals*.

Caslibs Scope

A caslib can have session scope or global scope. Session-scope caslibs make data available to the session that added the caslib. By default, when you add a caslib with the CASLIB statement, the caslib is session scope. You cannot change the scope of a caslib once it has been added.

Global-scope caslibs make data available to all sessions. By default, your personal caslib is global scope, but restricted to the sessions that you start. You can promote tables to

global-scope caslibs only. Global-scope caslibs are useful for data sources that all programmers need to access or in cases when you want to share data with other users. An administrator might restrict your ability to add a global-scope caslib. Use the GLOBAL CASLIB statement option to add a global-scope caslib. You can add caslibs only if you are authorized to do so. See Adjust Caslib Management Privileges

Session-scope caslibs are useful for ad hoc data analysis and in cases where you do not want to share data with other users.

For more information about caslib's scope, see "Caslibs Scope" in *SAS Cloud Analytic Services: Fundamentals*.

You can also hide caslibs. A hidden caslib is omitted from most lists of caslibs. Tables in a hidden caslib are omitted from most lists of tables. For more information about hidden caslibs, see Reduced Visibility: Hidden Caslibs.

The Active Caslib

When you start a session, your personal caslib is added by default. Initially, it is the active caslib if your server is configured with personal caslibs. If not, then the first defined global caslib is the active caslib. When you use the CASLIB statement to add a caslib, 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 in the CASUTIL procedure or as a data set option for a CAS engine libref. Because the active caslib is used as a default data source, only one caslib can be active at a time. If you use another CASLIB statement to add a caslib, the previous caslib becomes inactive, and the new caslib becomes active. To add a caslib without making it the active caslib, use the NOTACTIVE CASLIB statement option. You can also set the active caslib with the CASLIB= session option. For information about the CASLIB= session option, see "CASLIB= Session Option" on page 285.

Troubleshooting

The following list identifies several messages that you might encounter. In most cases, the messages can be experienced from any programming interface. For messages that are produced in SAS Studio only, an indication is provided.

You Do Not Have Permission to Create a Caslib

ERROR: You do not have permission to create global caslibs or ERROR: You do not have permission to create session caslibs.

You can add caslibs only if you are authorized to do so. See Adjust Caslib Management Privileges

Examples

Example 1: Add a Global Caslib

Program

The following example adds the caslibs Vapublic and Hadooplib. They each contain connection information to the data sources. After running this program, Hadooplib is the active caslib.

```
/*If not already done, create session Casauto.*/
/*Specify a host and port that are valid for your site.*/
/*options cashost="cloud.example.com" casport=5570;*/
/*cas casauto;*/
```

```
caslib Vapublic path="/vapublic"
                          datasource=(srctype="hdfs") global ;        /*1*/

caslib Hadooplib desc="Hadoop Caslib"
               datasource=(srctype="hadoop",                          /*2*/
               dataTransferMode="parallel",
               hadoopjarpath="Hadoo-jar-file-path",
               hadoopconfigdir="Hadoop-config-files-path",
               username="user-id",
               server="Hadoop-server-hostname",
               schema="schema-name") global;
```

1 The first CASLIB statement adds a global-scope caslib named Vapublic. The DATASOURCE= option and the PATH= option provide connection information to the Vapublic directory. The GLOBAL option enables you to promote tables to the caslib. You must also set the caslib to Global in CAS Server Monitor. Vapublic is now the active caslib.

2 The second CASLIB statement adds a global-scope caslib named Hadooplib, which provides access to a Hadoop database. The DATASOURCE= option specifies the option to use when connecting to the database. The GLOBAL option enables you to promote tables to the caslib. Hadooplib is now the active caslib.

SAS Log

The notes in the SAS log verify that the caslibs Vapublic and Hadooplib were added. Note that Hadooplib is the active caslib, because it was added last.

```
 28         caslib Vapublic path="/vapublic"
 29                                datasource=(srctype="hdfs") global ;
 NOTE: 'VAPUBLIC' is now the active caslib.
 NOTE: Cloud Analytic Services added the caslib 'VAPUBLIC'.
 NOTE: Action to ADD caslib VAPUBLIC completed for session CASAUTO.
 30
 31
 32         caslib Hadooplib desc="Hadoop Caslib"
 33
datasource=(srctype="hadoop",                                    /*2*/
 34                        dataTransferMode="parallel",
 35                        hadoopjarpath="Hadoo-jar-file-path",
 36                        hadoopconfigdir="Hadoop-config-files-path",
 37                        username="user-id",
 38                        server="Hadoop-server-hostname",
 39                        schema="schema-name") global;
 NOTE: 'HADOOPLIB' is now the active caslib.
 NOTE: Cloud Analytic Services added the caslib 'HADOOPLIB'.
 NOTE: Action to ADD caslib HADOOPLIB completed for session CASAUTO.
```

Additional Information

- For information about specifying Hadoop data source options, see “SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop” on page 187.
- You can add caslibs only if you are authorized to do so. See Adjust Caslib Management Privileges

Example 2: List Caslib Settings

You can list the caslib properties for one or more caslibs with the LIST statement.

Program 1: List the Settings for a Specific Caslib

The CASLIB statement with the LIST option and a caslib name displays the caslib settings for a specific caslib.

```
caslib CASUSER list;
caslib Hadooplib list;
```

SAS Log Showing Data Connector Settings

The settings for uid, server, hadoopConfigDir, hadoopJarPath, and schema only appear when listing the caslib explicitly. The settings that are displayed correspond to data connector options and are specific to the data connector.

```
NOTE: Session = CASAUTO Name = HADOOPLIB
         Type = hadoop
         Description = Hadoop Caslib
         Path =
         Definition = dataTransferMode = 'Parallel'
         Subdirs = No
         Local = No
         Active = Yes
         Personal = No
         Definition = uid = 'hiveuser'
         Definition = server = 'hive01.example.com'
         Definition = hadoopConfigDir = '/data/cdh54/sdm/conf'
         Definition = hadoopJarPath = '/data/cdh54/sdm/lib'
         Definition = schema = 'default'
```

Program 2: List All Caslibs

The CASLIB statement with the LIST option and the _ALL_ option displays all of the caslibs that are available and the caslib settings for each one.

```
caslib _all_ list;
```

SAS Log Showing Settings for All Caslibs

Caslibs 1 and 2 are the personal caslibs that were automatically created when the session started. Caslibs 3 and 4 were added explicitly in the code above. Caslib 5 was created by another session, but has global scope, so it is available to all sessions.

```
NOTE: Session = CASAUTO Name = CASUSER(casdemo)                    1
         Type = PATH
         Description = Personal File System Caslib
         Path = /u/casdemo/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
         Personal = Yes
NOTE: Session = CASAUTO Name = CASUSERHDFS(casdemo)                2
         Type = HDFS
         Description = Personal HDFS Caslib
         Path = /user/casdemo/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
         Personal = Yes
NOTE: Session = CASAUTO Name = HADOOPLIB                           3
         Type = hadoop
         Description = 'Hadoop Caslib'
         Path =
         Definition = ui
         Subdirs = No
         Local = No
         Active = No
         Personal = No
NOTE: Session = CASAUTO Name = VAPUBLIC                            4
         Type = HDFS
         Description =
         Path = /vapublic/
         Definition =
         Subdirs = No
         Local = No
         Active = No
         Personal = No
NOTE: Session = CASAUTO Name = Formats                             5
         Type = PATH
         Description = Format Caslib
         Path = /casdemo/formats/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
```

Additional Information

When you list caslibs with the **CASLIB _ALL_ LIST;** statement, the following caslib setting information is displayed for all caslibs:

Type=string	indicates the caslib type specified by the SRCTYPE= option.
Description=string	displays the description specified by the DESCRIPTION= option.
Path=string	displays the path specified in the PATH= option.
Definition=string	displays the data source options specified by the DATASOURCE= option.
Subdirs=Yes | No	indicates whether the caslib can access subdirectories. Subdirs= is specified by the SUBDIRS option.

Local=Yes | No	indicates whether the caslib is session scope. **Local=yes** indicates session scope. Local= corresponds to the SESSION option in the CASLIB statement.
Active=Yes | No	indicates whether the caslib is the active caslib.
Personal=Yes | No	indicates whether the caslib is a personal caslib or not.

When you list a specific data connector caslib, additional information about the connection settings is displayed, as shown in "SAS Log Showing Data Connector Settings" on page 98. The settings that are displayed correspond to data connector options and are specific to the data connector. For the data source options syntax for databases such as Hadoop and Oracle, see Chapter 9, "Data Connectors," on page 167.

- You can also use the CASUTIL procedure to view information about caslibs. For documentation about the CASUTIL procedure, see Chapter 5, "CASUTIL Procedure," on page 109.
- For the Hadoop connect string syntax, see "SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop" on page 187.

Example 3: Load and Save a Table

Program

The following program adds the caslib Myvapublic, which provides a place to copy an in-memory version of the Cars data set. The in-memory table, named carsWght, is then saved to the HDFS data source.

```
caslib Myvapublic path="/vapublic"
                         datasource=(srctype="hdfs");          /*1*/

libname mycas cas caslib=myvapublic;                           /*2*/

data mycas.carsWght;                                           /*3*/
   set sashelp.cars;
   if weight<5500 then delete;
   keep make model type weight MPG_City;
run;

proc casutil incaslib="Myvapublic" outcaslib="Myvapublic";     /*4*/
   list tables incaslib="Myvapublic";
   save casdata="CarsWght";
run;
```

1 The first CASLIB statement adds a global-scope caslib named Myvapublic. The DATASOURCE= option and the PATH= option provide connection information to the Myvapublic directory. Myvapublic is now the active caslib.

2 The LIBNAME statement assigns the libref Mycas. To run a DATA step in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names. The CASLIB= option binds the libref to the Myvapublic caslib.

3 This DATA step creates an in-memory table named Mycas.CarsWght in the Myvapublic caslib. There is no on-disk representation and it does not persist in the Myvapublic directory unless you save it.

4 The CASUTIL procedure saves the table to Myvapublic. The INCASLIB= option specifies the caslib that contains the file, and the OUTCASLIB= option specifies the

caslib that the file is being made available to. Use the PROC CASUTIL LIST statement to make sure that the table exists in the Myvapublic caslib.

SAS Log

The note in the SAS log shows that the table was saved.

```
60         save casdata="CarsWght";
NOTE: Cloud Analytic Services saved the file CarsWght.sashdat to HDFS in caslib
Myvapublic.
```

Results

The following results of the PROC CASUTIL LIST TABLES statement show the tables that are in the Myvapublic caslib.

The CASUTIL Procedure

Caslib Information	
Library	MYVAPUBLIC
Source Type	HDFS
Path	/vapublic/
Session local	Yes
Active	Yes
Personal	No
Hidden	No

The SAS System

The CASUTIL Procedure

Table Information for Caslib MYVAPUBLIC									
Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
CARSWGHT	7	5	utf-8	31Jan2017:14:31:55	31Jan2017:14:31:55	No	No	No	No

Additional Information

- Caslibs that use SRCTYPE="HDFS" are for distributed servers only. They use a Hadoop instance that is co-located with SAS Cloud Analytic Services.
- Caslibs provide a way to organize in-memory tables and an associated data source. They also provide a way to apply access controls to data. Once the caslib is dropped, the in-memory tables are dropped, too. Files in the caslib's data source are not removed or modified. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- Tables that are saved from a caslib are saved in SASHDAT format by default.
- For information about using the DATA step in CAS, see *SAS Cloud Analytic Services: DATA Step Programming*.
- For documentation about the CASUTIL procedure syntax, see Chapter 5, "CASUTIL Procedure," on page 109.
- For more examples of using the CASUTIL procedure to access and save data, see Chapter 1, "Accessing Data," on page 1.

Example 4: Copy Data from One Data Source to Another

```
caslib ldbeta datasource=(srctype="path")
                    path="path-to-directory"
                    description="imported files";

proc casutil incaslib="LDbeta" outcaslib="hps";
   contents casdata="donations.csv";
   load casdata="donations.csv" casout="donations";
   list tables incaslib="hps";
   save casdata="donations" incaslib="hps";
run;
```

1 The first CASLIB statement adds a session-scope caslib named Ldbeta. The DATASOURCE= option provides connection information to a path-based directory.

2 The CASUTIL procedure loads and saves the table to the data source. The INCASLIB= option specifies the caslib that contains the file, and the OUTCASLIB= option specifies the caslib that the file is being made available to.

3 The CONTENTS statement reads the on-disk file, Donations.csv, and displays the table metadata. This enables you to learn if the file has column names in the first row and the data types.

4 The LOAD CASDATA= statement reads the CSV file into memory and explicitly names the table Donations. The table is now available for analytics.

5 The LIST TABLES statement confirms that the in-memory table named Donations is available in the HPS caslib.

6 The SAVE statement saves the table as a SASHDAT file so that it can be loaded from the HPS caslib in the future rather than imported from the CSV file.

Results

The following results show table metadata displayed by the PROC CASUTIL CONTENTS statement.

Output 4.1 *Table Metadata*

Summary of Presentation and Taste Scores

The CASUTIL Procedure

CAS File Information

Name	Permission	Owner	Group	File Size	Last Modified
donations.csv	-rwxr-xr-x			809.8MB	14May2014:12:32:46

Column Information for donations.csv in Caslib LDBETA

Column	Type	Length	Formatted Length
donationid	varchar	0	0
projectid	varchar	0	0
donor_acctid	varchar	0	0
donor_city	varchar	0	0
donor_state	varchar	0	0
donor_zip	double	8	12
is_teacher_acct	varchar	0	0
donation_timestamp	varchar	0	0
donation_to_project	double	8	12
donation_optional_support	double	8	12
donation_total	double	8	12
dollar_amount	varchar	0	0
donation_included_optional_support	varchar	0	0
payment_method	varchar	0	0
payment_included_acct_credit	varchar	0	0
payment_included_campaign_gift_card	varchar	0	0
payment_included_web_purchased_gift_card	varchar	0	0
payment_was_promo_matched	varchar	0	0
via_giving_page	varchar	0	0
for_honoree	varchar	0	0
donation_message	varchar	0	0

The following results show the files available in the HPS caslib, displayed by the PROC CASUTIL LIST statement.

Output 4.2 *Partial Display of Files Available in the HPS Caslib*

Table Information for Caslib HPS

Label	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table
	279979	64	utf-8	18Mar2016:14:14:13	18Mar2016:14:14:13	Yes
	2018	24	utf-8	18Mar2016:14:14:26	18Mar2016:14:14:26	Yes
All Data Types	20	8	latin1	18Mar2016:14:14:39	18Mar2016:14:14:39	Yes
QUERY_FOR_actual_plan_2010_2011.sas7bdat	55320	25	latin1	18Mar2016:14:14:51	18Mar2016:14:14:51	Yes
Amazon Reviews	1830	12	latin1	18Mar2016:14:15:04	18Mar2016:14:15:04	Yes
Asia cities map coordinates	193723	26	latin1	18Mar2016:14:15:17	18Mar2016:14:15:17	Yes
Big organics	111115	13	latin1	18Mar2016:14:15:54	18Mar2016:14:15:54	Yes
	399999	8	utf-8	18Mar2016:14:16:08	18Mar2016:14:16:08	Yes
Bodyfat2	252	19	latin1	18Mar2016:14:16:21	18Mar2016:14:16:21	Yes
Campaign for Investments	1660	33	latin1	18Mar2016:14:16:34	18Mar2016:14:16:34	Yes
Car Sales	20	9	latin1	18Mar2016:14:16:47	18Mar2016:14:16:47	Yes
Car Sales data from sashelp library	428	15	latin1	18Mar2016:14:17:01	18Mar2016:14:17:01	Yes
Census	129605	172	latin1	18Mar2016:14:17:14	18Mar2016:14:17:14	Yes
	3476	4	utf-8	18Mar2016:14:17:26	18Mar2016:14:17:26	Yes
Claim History	10302	36	latin1	18Mar2016:14:17:39	18Mar2016:14:17:39	Yes

Additional Information

- Caslibs provide a way to organize in-memory tables and an associated data source. They also provide a way to apply access controls to data. After a caslib is dropped, the in-memory tables are dropped, too. Files in the caslib's data source are not removed or modified in any way. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- Tables that are saved from a caslib are saved in SASHDAT format by default.
- For information about using the DATA step in CAS, see *SAS Cloud Analytic Services: DATA Step Programming*.
- For documentation about the CASUTIL procedure syntax, see Chapter 5, "CASUTIL Procedure," on page 109.
- For more examples of using the CASUTIL procedure to access and save data, see Chapter 1, "Accessing Data," on page 1.

Example 5: Encrypt Tables in a Caslib

```
caslib Encr datasource=(srctype="path", encryptionPassword="your-password")
                                        path="your-file-path";
run;

libname mycas cas;

proc casutil;
   load data=sashelp.cars groupby=("make") outcaslib="Encr";
   contents casdata="cars";
   list files incaslib="encr";

proc mdsummary data=mycas.cars;
   var msrp invoice;
   output out=mycas.mdsumstatEncr;
   groupby make;
run;
```

```
options obs=15;
proc print data=mycas.mdsumstatEncr;
   var Make _Column_ _NObs_ _Mean_ _Max_ _Min_ _Std_;
   title "Summary of MSRP and Invoice, Grouped by Make";
run;

proc casutil incaslib="Encr" outcaslib="Encr";
   save casdata="mdsumstatEncr";
   list files incaslib="encr";
run;
```

1 The ENCRYPTIONPASSWORD= option in the CASLIB statement specifies a password for encrypting or decrypting tables.

2 Assign a CAS engine libref with the LIBNAME statement. To run PROC MDSUMMARY and PROC PRINT in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names.

3 The LOAD CASDATA= statement reads the file into memory. The table is now available for analytics.

4 The CONTENTS statement reads the on-disk file, Cars, and displays the table metadata. This enables you to determine whether the file has column names in the first row and the data types.

5 The LIST TABLES statement confirms that the in-memory table named Cars is available in the Encr caslib.

6 The MDSUMMARY procedure computes the descriptive statistics and groups them by Make.

7 The PRINT procedure prints the output.

8 The CASUTIL procedure saves the table to the data source. The INCASLIB= option specifies the caslib that contains the file, and the OUTCASLIB= option specifies the caslib that the file is being made available to.

9 The SAVE statement stored the data as UTF-8 when it created the SASHDAT file.

10 The LIST FILES statement confirms that the in-memory table named MdsumstatEncr is saved in the data source that the Encr caslib is associated with.

Results: LIST Statement

The following partial results of the PROC CASUTIL LIST statement show that the caslib is encrypted and a password is specified.

Output 4.3 *ENCR Caslib Information*

The CASUTIL Procedure

Caslib Information	
Library	ENCRT
Source Type	PATH
Path	[redacted]
EncryptionPassword	*****
Session local	Yes
Active	Yes
Personal	No
Hidden	No

Results: CONTENTS Statement

Output 4.4 *Column Information for the Cars Data Set*

Column Information for CARS in Caslib ENCRT

Column	Label	Type	Length	Formatted Length	Format
Make		char	13	13	
Model		char	40	40	
Type		char	8	8	
Origin		char	6	6	
DriveTrain		char	5	5	
MSRP		double	8	8	DOLLAR
Invoice		double	8	8	DOLLAR
EngineSize	Engine Size (L)	double	8	12	
Cylinders		double	8	12	
Horsepower		double	8	12	
MPG_City	MPG (City)	double	8	12	
MPG_Highway	MPG (Highway)	double	8	12	
Weight	Weight (LBS)	double	8	12	
Wheelbase	Wheelbase (IN)	double	8	12	
Length	Length (IN)	double	8	12	

Results: LIST Statement Information for the Data Source

Output 4.5 *List of Files in the Data Source*

The CASUTIL Procedure

CAS File Information

Name	Permission	Owner	Group	Encryption Method	File Size	Last Modified
mdsumstatEncr.sashdat	-rwxr-xr-x	[redacted]	[redacted]	AES	124.6KB	31Jan2017:15:12:21

Additional Information

- Caslibs provide a way to organize in-memory tables and an associated data source. They also provide a way to apply access controls to data. After a caslib is dropped, the in-memory tables are dropped too. Files in the caslib's data source are not removed or modified in any way. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- Tables that are saved from a caslib are saved in SASHDAT format by default.
- For information about using the DATA step in CAS, see *SAS Cloud Analytic Services: DATA Step Programming*.

- For documentation about the CASUTIL procedure syntax, see Chapter 5, "CASUTIL Procedure," on page 109.
- For more examples of using the CASUTIL procedure to access and save data, see Chapter 1, "Accessing Data," on page 1.
- For documentation about the MDSUMMARY procedure, see Chapter 6, "MDSUMMARY Procedure," on page 135.

Chapter 5

CASUTIL Procedure

Overview: CASUTIL Procedure

What Does the CASUTIL Procedure Do?

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

- data transfer
- table and file information
- drops tables and deletes files

In the area of data transfer, you can perform the following operations:

- load a data set from a SAS library into a memory on SAS Cloud Analytic Services.
- save in-memory tables in a caslib to the data source that is associated with the caslib.
- load files from the data source that is associated with a caslib into memory on SAS Cloud Analytic Services.

For file and table information, you can perform the following operations:

- view column names, data types, and other column information.
- list the in-memory tables in a caslib.
- list the files in a caslib's data source.

In the area of table and file management, the procedure enables you to drop in-memory tables. Dropping a table frees resources in the server but leaves the file in the data source that is associated with the caslib untouched. The procedure also enables you to delete files from the data source that is associated with the caslib.

The CASUTIL procedure executes without using the RUN statement. After you submit the PROC CASUTIL statement, you can submit additional procedure statements without submitting the PROC statement again. Use the QUIT statement to terminate the procedure.

Terminology

The following terms are used throughout the CASUTIL procedure documentation:

file
: is used to refer to the source data that is in a caslib's data source. For a caslib that uses a path-based data source, this is natural. For a caslib that uses a database as a data source, the tables in the database are referred to as files.

table
: is used to refer to in-memory data. After a file (using the preceding definition) is loaded into the server, it is referred to as a table.

Syntax: CASUTIL Procedure

PROC CASUTIL <*option(s)*>;
CONTENTS CASDATA="*table-name*" <INCASLIB="*caslib*"> <*option(s)*>;
DELETESOURCE CASDATA="*file-name*" <INCASLIB="*caslib*"> <QUIET>;
DROPTABLE CASDATA="*table-name*" <INCASLIB="*caslib*"> <QUIET>;
LIST <FILES | TABLES> <*options(s)*>;
LOAD CASDATA="*file-name*" | DATA=*SAS-data-set* | FILE="*SAS-file*" <*option(s)*>;
PROMOTE CASDATA="*table-name*" <INCASLIB="*caslib*"> <CASOUT="*table-name*"> <OUTCASLIB="*caslib*"> <DROP | KEEP>;
SAVE CASDATA="*table-name* " <INCASLIB="*caslib*"> <CASOUT="*file-name*"> <OUTCASLIB="*caslib*"> <*option(s)*>;
QUIT;

PROC CASUTIL Statement

Manages tables and files in SAS Cloud Analytic Services.

See: For examples of the CASUTIL procedure in real-life scenarios, see *SAS Cloud Analytic Services: User's Guide*.

Syntax

PROC CASUTIL <*option(s)*>;

Optional Arguments

INCASLIB=*caslib*
specifies the input caslib for the procedure. This option does not change the active caslib for your session. If you do not specify this option here or in a statement, such as LOAD, then the active caslib is used.

Specifying the caslib to use is a best practice until you develop experience working with the active caslib with this procedure, the CAS LIBNAME engine, and other procedures.

OUTCASLIB=*caslib*
specifies the output caslib for the procedure. This option does not change the active caslib for your session. If you do not specify this option here or in a statement, such as LOAD, then the active caslib is used.

SESSREF=*session-name*
specifies the session to use with the procedure. If you omit SESSREF=, then procedure uses the session that specified in the &_SESSREF_ macro variable.

Alias SESSION=

CONTENTS Statement

The CONTENTS statement displays table metadata such as column names and data types for files or in-memory tables.

Examples: "Example 3: Promote a Table" on page 130
"Example 4: Saving and Loading Encrypted SASHDAT Files" on page 131

Syntax

CONTENTS CASDATA="*table-name*" <INCASLIB="*caslib*"> <*option(s)*>;

Required Argument

CASDATA="*table-name*"
specifies the name of the file or table.

Optional Arguments

IMPORTOPTIONS=(FILETYPE="*file-type*" <*file-type-options*>)
specifies the file format and options. Specify this option only if you specify a filename in the CASDATA= option. If you want to display metadata for an in-memory table (the result of a LOAD statement), then do not specify this option.

For information about *file-type* and *file-type-options*, see "IMPORTOPTIONS=(FILETYPE="*file-type*" *file-type-options*)" on page 117.

INCASLIB=*caslib*
specifies the caslib that is associated with the file or table. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

DATASOURCEOPTIONS=(*data-source-options*)
specifies overrides for the DATASOURCE= options for the caslib. For more information, see Chapter 7, "Platform Data Sources," on page 153 and Chapter 9, "Data Connectors," on page 167.

Alias OPTIONS=

ROWCOUNT
specifies to include the number of rows in the results. The data source for the input caslib must be HDFS and you must include the filename suffix in the CASDATA= option.

VARS=((*casinvardesc-1*) < (*casinvardesc-2*) …>)
specifies the variables to include. If you do not include this option, all variables are included.

The value for *casinvardesc* is described in "VARS=((*casinvardesc-1*) (*casinvardesc-2*) …)" on page 118.

Alias VARLIST=

Example: Viewing Column Names from a CSV File

```
contents casdata="somefile.csv" importoptions=(filetype="csv");
```

DELETESOURCE Statement

The DELETESOURCE statement removes a file from the data source that is associated with a caslib. You do not need to drop an in-memory table with the same name before using this statement.

Note: You can delete files from path-based caslibs. These are caslibs with a data source type of DNFS, HDFS, or PATH.

Syntax

DELETESOURCE CASDATA="*file-name*" <INCASLIB="*caslib*"> <QUIET>;

Required Argument

CASDATA="*file-name*"
specifies the name of the file to remove.

Optional Arguments

INCASLIB="*caslib*"
specifies the caslib that is associated with the file to remove. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

QUIET
suppresses error messages and avoids setting the SYSERR automatic macro variable when the specified table or file is not found.

DROPTABLE Statement

The DROPTABLE statement removes a table from memory on SAS Cloud Analytic Services.

Example: "Example 4: Saving and Loading Encrypted SASHDAT Files" on page 131

Syntax

DROPTABLE CASDATA="*table-name*" <INCASLIB="*caslib*"> <QUIET>;

Required Argument

CASDATA="*table-name*"
specifies the name of the table to remove from memory.

Optional Arguments

INCASLIB="*caslib*"
specifies the caslib that is associated with the table to remove. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

QUIET
suppresses error messages and avoids setting the SYSERR automatic macro variable when the specified table is not found.

LIST Statement

The LIST statement lists files from a caslib's data source or in-memory tables in a caslib.

Example: "Example 1: Load a CSV File into CAS" on page 127

Syntax

LIST <FILES | TABLES> <*option(s)*>;

Required Argument

FILES | TABLES
specifies whether to list the files from a caslib's data source or in-memory tables.

FILES
lists the files that are available in the caslib's data source.

TABLES
lists the in-memory tables in a caslib.

Default TABLES

Optional Arguments

INCASLIB="*caslib*"
specifies the caslib that is associated with the tables or files to list. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

NOSUBDIRS
specifies to exclude subdirectories from the results.

Applies to LIST FILES and path-based caslibs

DATASOURCEOPTIONS=(*data-source-options*)
specifies overrides for the DATASOURCE= options for the caslib. For more information, see Chapter 7, "Platform Data Sources," on page 153 and Chapter 9, "Data Connectors," on page 167.

Alias OPTIONS=

Applies to LIST FILES

ROWCOUNT
specifies to include the number of rows in the results. The data source for the input caslib must be HDFS.

Applies to LIST FILES

SUBDIR="*path*"
specifies to list the files in the specified subdirectory.

Applies to LIST FILES and path-based caslibs

LOAD Statement

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

Syntax

Form 1: **LOAD** CASDATA="*file-name*" <INCASLIB="*caslib*"> CASOUT="*table-name*"
<OUTCASLIB="*caslib*">
<IMPORTOPTIONS=(FILETYPE="*file-type*" <*file-type-options*>)>
<GROUPBY=(*group-by-variable-1* <*group-by-variable-2 …*>)>
<ORDERBY=(*variable-1*
<*variable-2 …*>)>
<LABEL="*table-description*">
<DATASOURCEOPTIONS=(*data-source-options*)>
<PROMOTE | REPLACE>
<COPIES=*integer*>
<VARS=((*casinvardesc-1*) <, (*casinvardesc-2*), …>)>
<WHERE="*where-expression-1 <logical-operator where-expression-2*>">;

Form 2: **LOAD** DATA=*SAS-data-set*<(*data-set-options*)>
<CASOUT="*table-name*">
<OUTCASLIB="*caslib*">
<APPEND | COMPRESS>
<GROUPBY=(*group-by-variable-1 <group-by-variable-2 …*>) >
<ORDERBY=(*variable-1*
<*variable-2 …*>)>
<LABEL="*table-description*">
<DATASOURCEOPTIONS=(*data-source-options*)>
<PROMOTE | REPLACE>
<REPEAT>
<COPIES=*integer*>;

Form 3: **LOAD** FILE="*SAS-file*" CASOUT="*table-name*"
<OUTCASLIB="*caslib*">
<COMPRESS>
<IMPORTOPTIONS=(FILETYPE=*file-type <file-type-options>*)>
<LABEL="*table-description*">
<COPIES=*integer*>;

Required Arguments

CASDATA="*file-name*"
specifies the name of the file to load from the server-side data source that is associated with the INCASLIB= option or the active caslib.

Requirement You must specify CASOUT=.

CASOUT="*table-name*"
specifies the name to use for the in-memory table.

Interaction This argument is required when you use the LOAD CASDATA= or LOAD FILE= forms.

Note This argument does not follow or enforce SAS naming rules such as the name literal syntax.

Tip Some data sources support table names that exceed 32 bytes. Use this option to limit table names to 32 bytes so that you can access the table with the CAS LIBNAME engine.

DATA=*SAS-data-set*
specifies the libref and data set name to use.

FILE="*SAS-file*"
specifies an external file that is accessible to the SAS client host. Use this option to upload a file to the server and import the data. Do not use this option to import a SAS data set, use the DATA= option.

Requirement You must specify CASOUT=.

Optional Arguments

APPEND
adds the rows from the SAS data set in the DATA= argument to the end of an in-memory table. This option is supported with the DATA= argument only.

CASNCHARMULTIPLIER=*n*

n
specifies a number that is used as a multiplication factor to control the byte size for fixed character data.

Range $0 < n \leq 4$

Defaults For SBCS environments, the default is 1.

For DBCS environments, the default is 1.5.

COMPRESS
specifies to compress the output table. This option is supported with the DATA= argument only.

COPIES=*integer*
specifies the number of replicate copies of the rows to make for fault tolerance. Larger values use more memory and can result in slower performance, but provide high availability for data in the event of a node failure.

Alias REPLICATION=

Default 1

Interactions This option is ignored if the REPEAT option is also specified.

This option is ignored if the data source is a data connector or data connect accelerator. The SAS Data Connector to Oracle and SAS Data Connector to Hadoop are examples of these products.

DATASOURCEOPTIONS=(*data-source-options*)
specifies overrides for the DATASOURCE= options for the input caslib. For more information, see Chapter 7, “Platform Data Sources,” on page 153 and Chapter 9, “Data Connectors,” on page 167.

Alias OPTIONS=

GROUPBY=(*group-by-variable-1 <group-by-variable-2…>*)
specifies the names of the variables to use for grouping results.

Alias PARTITIONBY=

IMPORTOPTIONS=(FILETYPE="*file-type*" *<file-type-options>*)
specifies the file format and options.

FILETYPE="AUTO" | "BASESAS" | "CSV" | "DTA" | "EXCEL" | "HDAT" | "LASR" | "XLS"
specifies the file format. AUTO attempts to determine the file type based on the filename suffix, such as .sashdat, .csv, and so on. Files with a .txt suffix are imported as a CSV file.

Default AUTO

file-type-options
specifies options for importing the data. For more information, see Chapter 8, "Platform File Types," on page 161 and Chapter 9, "Data Connectors," on page 167.

INCASLIB="*caslib*"
specifies the caslib that is associated with the file to load. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

LABEL="*string*"
specifies a descriptive label for the table. The label can be up to 256 characters. If the label text contains single quotation marks, enclose the label in double quotation marks. To remove a label from a table, assign a blank space that is enclosed in quotation marks.

NCHARMULTIPLIER=*n*

n
specifies the multiplication factor used when transcoding.

Range $0 < n \leq 4$

Defaults For SBCS environments the default is 1

For DBCS environments, the default is 1.5.

ORDERBY=(*variable-1 <variable-2…>*)
specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables.

OUTCASLIB="*caslib*"
specifies an alternative caslib to use for the in-memory table. If specified, this option overrides the OUTCASLIB= value in the procedure statement or the active caslib.

PROMOTE
specifies to load the table with global scope. This makes the table available to all sessions that use the caslib, subject to access controls. The caslib must also have global scope.

REPEAT
specifies to duplicate the rows for the table on every machine of a distributed server. Making duplicate copies of tables can be useful in cases like a dimension table that is

used in a join. This option is supported with the DATA= argument only and cannot be combined with the GROUPBY= option.

REPLACE
specifies that an in-memory table with a given name replaces an existing in-memory table with the same name.

VARS=((*casinvardesc-1*) < (*casinvardesc-2*) …>)
specifies the variables to load into the table. If you do not specify this option, then all variables are loaded into the table.

The value can be one or more of the following:

FORMAT="*string*"
specifies the format to apply to the variable.

FORMATTEDLENGTH=*integer*
specifies the format field length plus the format precision length.

LABEL="*string*"
specifies the descriptive label for the variable.

NAME="*string*"
specifies the name for the variable.

NFD=*integer*
specifies the format precision length.

NFL=*integer*
specifies the format field length.

Alias VARLIST=

WHERE="*where-expression-1 <logical-operatorwhere-expression-2>* "
specifies conditions for selecting observations from the data.

where-expression
is an arithmetic or logical expression that consists of a sequence of operators, operands, and SAS functions. An operand is a variable, a SAS function, or a constant. An operator is a symbol that requests a comparison, logical operation, or arithmetic calculation. The expression must be enclosed in single or double quotation marks.

logical-operator
can be AND, AND NOT, OR, or OR NOT.

Details

Summary of Options

Table 5.1 *Summary of LOAD Statement Options*

LOAD Statement Option	LOAD Statement Form
APPEND	LOAD DATA=
CASNCHARMULTIPLIER	CASNCHARMULTIPLIER=
COMPRESS	LOAD DATA=

LOAD Statement Option	LOAD Statement Form
COPIES=	All LOAD statement forms. For more information, see "Data Redundancy" on page 153.
DATASOURCEOPTIONS=	LOAD CASDATA= and LOAD DATA=
GROUPBY=	LOAD CASDATA= and LOAD DATA=
IMPORTOPTIONS=	LOAD CASDATA= and LOAD FILE=
INCASLIB=	LOAD CASDATA=
LABEL=	All LOAD statement forms
NCHARMULTIPLIER	NCHARMULTIPLIER=
ORDERBY=	LOAD CASDATA= and LOAD DATA= *Note:* ORDERBY= requires GROUPBY=
OUTCASLIB=	All LOAD statement forms
PROMOTE	All LOAD statement forms
REPEAT	LOAD DATA=
REPLACE	All LOAD statement forms *Note:* Keep in mind that global-scope tables cannot be replaced. Use the DROPTABLE statement before the LOAD statement.
VARS=	LOAD CASDATA=
WHERE=	LOAD CASDATA= *Note:* You can specify WHERE= as a data set option when you use the LOAD DATA= form.

PROMOTE Statement

The PROMOTE statement copies a session-scope table to global scope.

Note: The PROMOTE statement does not include a REPLACE option. The server does not support promoting a session-scope table and replacing a global-scope table in one operation. You must drop the global-scope table first.

Example: "Example 3: Promote a Table" on page 130

Syntax

PROMOTE CASDATA="*table-name*" <INCASLIB="*caslib*">

<CASOUT="*table-name*">
<OUTCASLIB="*caslib*">
<DROP | KEEP>;

Required Argument

CASDATA="*table-name*"
specifies the name of the in-memory table to promote.

Optional Arguments

CASOUT="*table-name*"
specifies the name to use for the promoted table.

Note This argument does not follow or enforce SAS naming rules such as the name literal syntax.

Tip Some data sources support table names that exceed 32 bytes. Use this option to limit table names to 32 bytes so that you can access the table with the CAS LIBNAME engine.

DROP
specifies to drop the session-scope table after promoting it to global scope, which is the default behavior and a best practice.

INCASLIB="*caslib*"
specifies the caslib with the in-memory table to promote. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

KEEP
specifies to keep the session-scope table after promoting it to global scope. This workflow is uncommon. The table precedence rules are that the session-local table is accessed before a global-scope table is accessed.

OUTCASLIB="*caslib*"
specifies an alternative caslib to use for the promoted table. If specified, this option overrides the OUTCASLIB= value in the procedure statement or the active caslib.

SAVE Statement

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

Example: "Example 4: Saving and Loading Encrypted SASHDAT Files" on page 131

Syntax

SAVE CASDATA="*file-name*" <INCASLIB="*caslib*">
<CASOUT="*table-name*"> <OUTCASLIB="*caslib*">
<*option(s)*>;

Required Argument

CASDATA="*table-name*"
specifies the name of the in-memory table to save.

Optional Arguments

CASOUT="*file-name*"
specifies an alternative name for the file. A file is created in the data source that is associated with the caslib from the OUTCASLIB= option. By default, a .sashdat file suffix is added when you save to a path-based caslib. If you specify a .csv suffix, a CSV file is saved.

Note This argument does not follow or enforce SAS naming rules such as the name literal syntax.

Tip Some data sources support table names that exceed 32 bytes. Use this option to limit table names to 32 bytes so that you can access the table with the CAS LIBNAME engine when you load it again.

INCASLIB="*caslib*"
specifies the caslib with the in-memory table to save. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

OUTCASLIB="*caslib*"
specifies an alternative caslib to use for the saved table. If specified, this option overrides the OUTCASLIB= value in the procedure statement or the active caslib.

SAVE Statement Options

COMPRESS
specifies to compress the data in the saved file.

COPIES=*integer*
specifies the number of replicate copies of the rows to make for fault tolerance. This value is ignored unless the output caslib data source is HDFS and you save to the SASHDAT file format.

Alias REPLICATION=

Default 1

GROUPBY=(*group-by-variable-1 <group-by-variable-2…>*)
specifies the names of the variables to use for partitioning the SASHDAT file.

Alias PARTITIONBY=

IMPORTOPTIONS=(FILETYPE="*file-type*" *<file-type-options>*)
specifies the input file format and options. Specify this option only if you want to read a file from the input caslib's data source and save it without loading it to memory. For more information, see IMPORTOPTIONS= on page 117 for the LOAD statement.

DATASOURCEOPTIONS=(*data-source-options*)
specifies overrides for the DATASOURCE= options for the caslib. For more information, see Chapter 7, “Platform Data Sources,” on page 153 and Chapter 9, “Data Connectors,” on page 167.

Alias OPTIONS=

ORDERBY=(*variable-1 <variable-2…>*)
specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables.

REPLACE
specifies that a new file with a given name replaces an existing file with the same name.

WHERE="*where-expression-1 <logical-operatorwhere-expression-2>* "
The specification for this option is described in WHERE= on page 118 for the LOAD statement.

Enclose Values in Quotation Marks

When you specify a value for one of the following items, you can enclose the value in quotation marks:

- INCASLIB="*caslib*"
- OUTCASLIB="*caslib*"
- CASDATA="*table-name*"
- CASOUT="*table-name*"

For caslibs that use a case-sensitive file system or database as a data source, you control the case that is used.

If you do not enclose the INCASLIB= and OUTCASLIB= values in quotation marks, then SAS naming rules apply.

Subdirectories and Filename Matching

Exclude Subdirectories from the File Information Results

If a path-based caslib has many subdirectories, the results of the LIST FILES statement can be long and can obscure the list of files that can be loaded as data. The NOSUBDIRS option is available to exclude the subdirectories from the results.

```
cas casauto setsessopt=(caslib='casuser');

proc casutil;
   list files nosubdirs;
quit;
```

List the Files in a Subdirectory

This topic is the opposite of the preceding topic. You might have files in a subdirectory of a caslib's data source root that you want to use as data. For example, if your CASUSER personal caslib is assigned to **/home/sasdemo** and you have data in **/user/sasdemo/mydata**, you cannot assign a caslib to the **/home/sasdemo/mydata** path.

To list the files in **/home/sasdemo/mydata**, you can use the SUBDIR= option in the LIST FILES statement.

```
cas casauto setsessopt=(caslib='casuser');
```

```
proc casutil;
   list files subdir='mydata';
quit;
```

The CASUTIL Procedure

CAS File Information

Name	Permission	Owner	Group	File Size	Last Modified
iris.csv	-rwxr-xr-x	[redacted]	[redacted]	3.1KB	08Aug2016:19:38:59
titanic3.xls	-rwxr-xr-x	[redacted]	[redacted]	277.5KB	09Aug2016:09:01:48
jta_free_wifi.csv	-rwxr-xr-x	[redacted]	[redacted]	19.1MB	09Aug2016:09:03:39

Using Wildcard Characters for Filename Matching

Just as the NOSUBDIRS option can help limit the results of the LIST FILES statement, you can use wildcard characters to limit results. The following example lists the files with a CSV suffix.

```
cas casauto setsessopt=(caslib='casuser');

proc casutil;
   list files subdir='mydata/%.csv';
quit;
```

TIP When you specify a wildcard pattern, the results include the matches and any subdirectories (matching or otherwise). To exclude subdirectories, you can specify the NOSUBDIRS option.

- Filename matching is case sensitive. The **`%.csv`** pattern does not match files with an uppercase or mixed case suffix.
- **`%`** matches any number of characters or numbers.
- _ matches a single character or number. You can include more than one underscore in a pattern.
- \ escapes a wildcard character so that it is treated as a literal character instead of a wildcard in a pattern.

The SAS Macro language also uses the '%' character to indicate macro names to resolve during program compilation. When you use the '%' character for pattern matching, include the value in single quotations marks. If you enclose a value in double quotation marks, the SAS Macro parser tries to resolve a macro name. For example, the code **`SUBDIR="mydata/%s.csv"`** results in an attempt to resolve the %S macro name, with the message `WARNING: Apparent invocation of macro S not resolved.` Instead, specify the code as **`SUBDIR='mydata/%s.csv'`**.

Loading Files from a Subdirectory

After you list the files in a caslib's subdirectory, you can load the file into the server with the LOAD CASDATA= statement.

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

```
proc casutil;
   load casdata="mydata/iris.csv" casout="iris";
quit;
```

Saving Files to a Subdirectory

By default, when you use the SAVE statement, a file is created in the directory that is associated with your active caslib, or the OUTCASLIB= option. To save a file in a subdirectory, include the path in the CASOUT= option.

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

proc casutil;
   load casdata="mydata/iris.csv" casout="iris";
   save casdata="iris"
        where="species eq 'Setosa'"
        casout="mydata/setosa.csv"
        replace;

   list files subdir="mydata";
quit;
```

The CASUTIL Procedure

CAS File Information

Name	Permission	Owner	Group	File Size	Last Modified
iris.csv	-rwxr-xr-x	[redacted]	[redacted]	3.1KB	08Aug2016:19:38:59
titanic3.xls	-rwxr-xr-x	[redacted]	[redacted]	277.5KB	09Aug2016:09:01:48
jta_free_wifi.csv	-rwxr-xr-x	[redacted]	[redacted]	19.1MB	09Aug2016:09:03:39
setosa.csv	-rwxr-xr-x	[redacted]	[redacted]	0.9KB	09Aug2016:10:00:23

Limitations and Restrictions

The FORMAT statement in a PROC CASUTIL step applies to the LOAD DATA statements only. The FORMAT statement must be submitted prior to the LOAD DATA statements. Otherwise, it has no effect.

Results: CASUTIL Procedure

Procedure Output

The CONTENTS statement provides detailed information for an in-memory table. The following program generated the results with a distributed server with seven worker nodes.

```
proc casutil;
   load data=sashelp.iris;
   contents casdata="iris";
quit;
```

Table Information for Caslib CASUSER(...)

Table Name	Label	Number of Rows	Number of Columns	Indexed Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
IRIS	Fisher's Iris Data (1936)	150	5	0	utf-8	06Oct2017:14:26:20	06Oct2017:14:26:20	No	No	No	No

Detail Information for iris 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	Index Size	Compressed Size	Compression Ratio
ALL	1	1	150	7200	0	0	0	0	0	1	7200	0	0	0

Column Information for IRIS in Caslib CASUSER(...)

Column	Label	Type	Length	Formatted Length	Format Width	Format Decimal
Species	Iris Species	char	15	15	0	0
SepalLength	Sepal Length (mm)	double	8	12	0	0
SepalWidth	Sepal Width (mm)	double	8	12	0	0
PetalLength	Petal Length (mm)	double	8	12	0	0
PetalWidth	Petal Width (mm)	double	8	12	0	0

The Promoted Table field indicates when a table has global scope. Yes indicates that the table is a global-scope table. No indicates that the table is a session-scope table. For information about scope and repeated tables, see "More About Tables" in *SAS Cloud Analytic Services: Fundamentals*. If you load a table with the LOAD CASDATA= form, then the Source Name field indicates the original filename and the Source Caslib field indicates the original caslib.

For the table details information, see these definitions:

Node
: This field always reports ALL. This procedure provides a summary of the table information for all machines in a distributed server.

Number of Blocks
: The server organizes rows in blocks. For distributed servers, this column shows the sum of the active blocks and any copies of blocks that provide redundancy.

Active Blocks
: The server reads rows from active blocks.

Fixed Data size
: This field shows the number of bytes that are used for numeric columns and fixed-width character columns.

Variable Data size
: This field shows the number of bytes that are used for variable-width character columns.

Blocks Mapped
: This field shows the number of blocks are currently mapped into memory.

Memory Mapped
: This field shows the number of bytes for the blocks that are mapped.

Blocks Unmapped
: This field shows the number of blocks that the server can map into memory. The blocks are mapped into memory when the server handles a request for data from the table. For distributed servers, the redundant blocks that enable fault tolerance are included in this value.

Memory Unmapped
: This field shows the number of bytes for the blocks that the server can map into memory.

Blocks Allocated
: This field shows the number of blocks that do not have an on-disk representation. The blocks can become cached under the following conditions:

  - when you promote a session-scope table to global-scope.
  - you set the MAXTABLEMEM= CAS session option to a lower value. If you append rows and cross the value, the server caches the blocks.

Memory Allocated
: This field shows the number of bytes for the blocks.

Index Size
: This field shows the number of bytes that are used for column indexes. You can index columns when you program with CAS actions.

Compressed Size
: This field shows the number of bytes that are used when the table is compressed. You can use compression by specifying the COMPRESS option when you use one of the forms of the LOAD statement.

Compression Ratio
: This field shows the ratio of the Fixed Data Size value when it is divided by the Compressed Size value. If the remainder from the division is greater than half of the compressed size, then the value is rounded up by 1.

There are two ways for the server to have on-disk blocks that correspond to the Blocks Mapped and Blocks Unmapped values:

- The server can create blocks in a directory that is used for caching. The directories are specified at deployment time by an administrator and corresponds to the CAS_DISK_CACHE environment variable.
- For distributed servers that are co-located with HDFS or use a DNFS caslib, the blocks correspond to the blocks of a SASHDAT file.

ODS Table Names

PROC CASUTIL assigns a name to each table that it creates.

Table 5.2 *ODS Tables Produced by the CASUTIL Procedure*

ODS Table	Description	Statement Used
CaslibInfo	Caslib information	LIST
ColumnInfo	Column information	CONTENTS
FileInfo	CAS file information	LIST with the FILES option
TableDetails	Detailed table information	CONTENTS
TableInfo	CAS table information	LIST with the TABLES option, CONTENTS

Examples: CASUTIL Procedure

Example 1: Load a CSV File into CAS

Program

```
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 to the CASLIB statement specifies a distributed 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 is used to specify the file type and options for reading the data into the server.

Results: LIST FILES Statement for the CSVFILES Caslib

The following display shows the results of the LIST FILES statement. It is a listing of the files that the server can access from the `/data/csv` directory.

The CASUTIL Procedure

File Information for root of caslib CSVFILES.

Name	Permission	Owner	Group	File Size	Last Modified
Spreadsheet10k.xlsx	-rwxrwxrwx	nfsnobody	nfsnobody	563.5KB	08Nov2012:07:21:12
mailorder.csv	-rwxrwxrwx	nfsnobody	nfsnobody	117.2KB	29Sep2015:11:40:14
NST_EST2012_ALLDATA.csv	-rwxrwxrwx	nfsnobody	nfsnobody	20.0KB	15Jan2013:08:59:28
sine.csv	-rwxrwxrwx	nfsnobody	nfsnobody	309.3KB	15Jan2013:13:18:14
SC-EST2011-6RACE-ALL.csv	-rwxrwxrwx	nfsnobody	nfsnobody	138.3KB	15Jan2013:09:02:53
sine10k.csv	-rwxrwxrwx	nfsnobody	nfsnobody	582.9KB	09May2013:07:16:16
UNdata_Export_20130115_092219649.csv	-rwxrwxrwx	nfsnobody	nfsnobody	595.4KB	15Jan2013:09:22:20
2012_nfl_pbp_data_reg_season1.csv	-rwxrwxrwx	nfsnobody	nfsnobody	6.5MB	06Feb2013:11:19:02
County_Population.csv	-rwxrwxrwx	nfsnobody	nfsnobody	1.3MB	14Jan2013:14:15:41

Results: LIST TABLES Statement for the CSVFILES Caslib

The following display shows the results of the LIST TABLES statement. It shows that the County_Population table is the only in-memory table in the caslib.

The CASUTIL Procedure

Caslib Information	
Library	CSVFILES
Source Type	DNFS
Description	"Spreadsheets and CSV source data."
Path	[redacted]
Session local	Yes
Active	Yes
Personal	No

The SAS System

The CASUTIL Procedure

Table Information for Caslib CSVFILES

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Source Caslib	Compressed
COUNTY_POPULATION	3193	97	utf-8	08Apr2016:10:41:18	08Apr2016:10:41:18	No	No	No	County_Population.csv	CSVFILES	No

Example 2: Append Rows to an In-Memory Table

Program

```
proc casutil;
   load data=sashelp.cars(where=(make="Buick"))                    /* 1 */
        casout="some_cars"
        label="Some makes from the Sashelp.Cars sample data."
        promote;

   /* add rows for a few more makes */
```

```
   load data=sashelp.cars(where=(make in ("Ford", "Chrysler")))
        casout="some_cars"
        append;                                                    /* 2 */

   list tables;
quit;

libname mycas cas;

proc cardinality data=mycas.some_cars outcard=mycas.cars_cardinality;
   vars enginesize mpg_highway mpg_city;                           /* 3 */
run;

proc casutil;
  contents casdata="cars_cardinality";                             /* 4 */
run;

proc print data=mycas.cars_cardinality;
  var _varname_ _cardinality_ _nobs_ _nmiss_ _min_--_kurtosis_;
run;
```

1 The first LOAD DATA= statement subsets the Sashelp.Cars data set based on the Make variable. The CASOUT= option specifies the name for the output table, Some_cars. The PROMOTE option sets the output table as a global-scope table.

2 The second LOAD statement uses the APPEND option to append more rows from the Sashelp.Cars data set.

3 The CARDINALITY procedure is used to calculate summary statistics for three variables. The OUTCARD= option specifies an in-memory table to use for storing the summary data.

4 The CONTENTS statement is used to display the table information and column information for the Cars_cardinality table. Notice that the libref is not included with the CASDATA= option. The results of the statement include the column names. Some of the column names are specified in the PRINT procedure.

Results: LIST TABLES Statement for the CASUSER Caslib

The CASUTIL Procedure

Table Information for Caslib [redacted]

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.	47	15	utf-8	08Apr2016:13:27:37	08Apr2016:13:27:37	Yes	No	No	No

Results: CONTENTS Statement for the Cars_Cardinality Table

The CASUTIL Procedure

Table Information for Caslib [redacted]

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	08Apr2016:13:29:19	08Apr2016:13:29:19	No	Yes	No	No

Detail Information for cars_cardinality in Caslib [redacted].

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	7	7	21	7560	0	0	0	7	8792	0	0

Column Information for CARS_CARDINALITY in Caslib [redacted]

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	$

Results: Select Columns from the Cars_Cardinality Table

Obs	_VARNAME_	_CARDINALITY_	_NOBS_	_NMISS_	_MIN_	_MAX_	_MEAN_	_STDDEV_	_SKEWNESS_	_KURTOSIS_
1	EngineSize	17	47	0	2	6.8	3.3957446809	1.0133795046	0.777871628	1.3478938265
2	MPG_Highway	15	47	0	13	36	26.638297872	4.3410354	-0.725567937	1.3662855198
3	MPG_City	12	47	0	10	27	19.382978723	3.4550101711	0.1929445364	0.6283786957

Example 3: Promote a Table

Program

```
caslib hps datasource=(srctype="path") path="/hps" global;

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

libname mycas cas;

proc casutil;
   load data=sashelp.iris casout="irisraw";
quit;

data mycas.irisout;                                    /* 1 */
   set mycas.irisraw;
   sepalratio = sepalwidth / sepallength;
   petalratio = petalwidth / petallength;
run;

/*
 * The purpose for outcaslib= is to show how to
 * work with more than one caslib.
 */
proc casutil outcaslib="hps";                          /* 2 */
```

```
   promote casdata="irisout";
quit;

proc casutil incaslib="hps";
   contents casdata="irisout";
quit;
```

1 The DATA step creates an output table that is named Irisout from an input table named Irisraw. Two columns are added to the table.

2 The OUTCASLIB= option is used to demonstrate how to work with more than one caslib. If you specify OUTCASLIB= when you promote or load a table, then you need to use INCASLIB= with the same name to access the table again.Notice that in the CASDATA= option in the PROMOTE statement on page 119 that follows, the table name is specified without the libref.

Results: CONTENTS Statement for the Irisout Table

The following graphic shows the results of the CONTENTS statement. The Table Information results show that the caslib is HPS and that the table is promoted to global scope. The Column Information results show the two columns that were added with the DATA step.

The CASUTIL Procedure

Table Information for Caslib HPS

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
IRISOUT	150	7	utf-8	08Apr2016:13:47:40	08Apr2016:13:47:40	Yes	No	No	No

Detail Information for irisout in Caslib HPS.

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	14	7	150	9600	0	0	0	14	19200	0	0

Column Information for IRISOUT in Caslib HPS

Column	Label	Type	Length	Formatted Length
Species	Iris Species	char	10	10
SepalLength	Sepal Length (mm)	double	8	12
SepalWidth	Sepal Width (mm)	double	8	12
PetalLength	Petal Length (mm)	double	8	12
PetalWidth	Petal Width (mm)	double	8	12
sepalratio		double	8	12
petalratio		double	8	12

Example 4: Saving and Loading Encrypted SASHDAT Files

Program

The CSV file that is used in this example was downloaded on 10FEB2016 from https://www.hokoukukan.go.jp/download/jta_free_wifi.csv. Your results for the count of WiFi access points by category could be different.

```
options validmemname=extend validvarname=any;                    /* 1 */

options caslib="casuser";
libname mycas cas;

proc casutil incaslib="casuser" outcaslib="casuser";
   load casdata="jta_free_wifi.csv"
        importoptions=(filetype="csv" getnames="yes"
           encoding="sjis")                                      /* 2*/
        casout="jta_free_wifi";

   contents casdata="jta_free_wifi";                             /* 3 */

   save casdata="jta_free_wifi"
        datasourceoptions=(encryptionPassword='changeit');

   droptable casdata="jta_free_wifi";                            /* 4 */

   load casdata="jta_free_wifi.sashdat"                          /* 5 */
        importoptions=(filetype="hdat" encryptionPassword='changeit');
quit;

proc mdsummary data=mycas.jta_free_wifi;                         /* 6  */
  var スポット ID;
  groupby カテゴリー / out=mycas.category;
run;

proc print data=mycas.category;                                  /* 7  */
  var カテゴリー _nobs_;
run;
```

1 This example uses data with column names that do not follow SAS naming conventions. These options provide greater flexibility with table names and column names.

2 The CSV file uses the Shift JIS encoding. If a file does not use UTF-8 or 7-bit ASCII, then specify the encoding.

3 The CONTENTS statement on page 112 displays the table information, table details, and column information. It is included in the example as a reminder that you should confirm that the column names from the file are imported as you expect them to be imported. See “Results: CONTENTS Statement for the Jta_free_wifi Table” on page 133.

4 The DROPTABLE statement is not necessary in most programs. It is included in this example so that the subsequent LOAD CASDATA= statement succeeds without the REPLACE option.

5 The LOAD CASDATA= statement includes the encryption password. Notice also that you do not need to specify an ENCODING= option. The SAVE statement stored the data as UTF-8 when it created the SASHDAT file.

6 The MDSUMMARY procedure is included to show that after the table is loaded into memory, then you can use a CAS engine libref to access the in-memory table. The goal for this example is to identify the different categories of hotspots and the counts. Only one variable is summarized, the WiFi hotspot identifier, and the variable is grouped by values of the hotspot category. The summary is output to an in-memory table on the server that is named Category.

7 The PRINT procedure is used to read the summarized results from the in-memory table on the server. The VAR statement limits the display to the different hotspot categories and the count for each category. See Output 5.1 on page 133.

Results: CONTENTS Statement for the Jta_free_wifi Table

The CASUTIL Procedure

Table Information for Caslib CASUSER([redacted])

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Source Caslib	Compressed
JTA_FREE_WIFI	42259	21	utf-8	08Apr2016:13:56:01	08Apr2016:13:56:01	No	No	No	jta_free_wifi.csv	CASUSER([redacted])	No

Detail Information for jta_free_wifi in Caslib CASUSER([redacted]).

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	84	42	42259	32730247	19883511	0	0	84	65471024	0	0

Column Information for JTA_FREE_WIFI in Caslib CASUSER([redacted])

Column	Type	Length	Formatted Length
スポットID	double	8	12
スポット名（日本語）	varchar	115	55
スポット名（英語）	varchar	255	255
スポットステータス	varchar	20	8

Results: WiFi Hotspot Categories and Counts

Output 5.1 *WiFi Hotspot Categories and Counts*

Obs	カテゴリー	_NObs_
1	その他	31989
2	バス	2
3	ホテル	1741
4	商業施設(百貨店、SC、アウトレットモール等)	46
5	港湾	5
6	移動中の休憩所(サービスエリア、道の駅等)	181
7	空港	6
8	美術館・博物館・寺社仏閣	11
9	観光スポット（景勝地等）	33
10	観光案内所	91
11	鉄道(駅構内)	175
12	飲食・小売店	7979

Chapter 6

MDSUMMARY Procedure

Overview: MDSUMMARY Procedure

What Does the MDSUMMARY Procedure Do?

The MDSUMMARY procedure computes basic descriptive statistics for variables across all observations or within groups of observations in parallel for data tables stored in SAS Cloud Analytic Services (CAS). The MDSUMMARY procedure uses CAS tables and capabilities, ensuring full use of parallel processing.

Syntax: MDSUMMARY Procedure

Restriction: You cannot use a FORMAT statement in a PROC MDSUMMARY step. If you need to format the input table, apply the formats to the table when it is created. For examples, see “Example 4: Using Formats with Group-By Variables” on page 147

and "Add and Save a User-Defined Format" in *SAS Cloud Analytic Services: User-Defined Formats*.

PROC MDSUMMARY DATA=*libref.table-name* <NTHREADS=*integer*>;
  VAR <*variable-list*>;
  OUTPUT OUT=*table-name*;
  GROUPBY *variable-list* </ OUT=*table-name*>;
RUN;

PROC MDSUMMARY Statement

Calculates multidimensional summaries of numeric variables.

Syntax

PROC MDSUMMARY DATA=*libref.table-name*<NTHREADS=*integer*>;

Optional Arguments

DATA=*libref.table-name*
specifies the two-level input table name.

Requirement The table name must be a two-level name where *Libref* is a CAS engine libref.

Alias TABLE=

NTHREADS=*integer*
specifies the number of threads to use within each compute node.

VAR Statement

Specifies the analysis variables and their order in the output.

Syntax

VAR <*variable-list*>

Without Arguments

If no variables are listed, then the summary statistics are computed for all numeric variables.

Optional Argument

<variable(s)>
identifies one or more analysis variables and specifies their order in the results.

GROUPBY Statement

Creates BY groups in terms of the variable value combinations given the variables in the variable list.

Requirement: You must specify either the OUTPUT statement, or at least one GROUPBY statement with the OUT= option specified. You cannot specify both the OUTPUT= statement and a GROUPBY statement with the OUT= option specified.

Tips: Multiple GROUPBY statements can be specified, each having its own output table.

If a variable value is missing, PROC MDSUMMARY includes the observations and rows in the analysis.

Syntax

GROUPBY <*variable(s)*> </ OUT=*table-name*>;

Without Arguments

If no variables are listed, then the statistics are calculated across all input observations

Optional Arguments

OUT=*table-name*
specifies the table name.

Requirement The table name must be a two-level name where *Libref* is a CAS engine libref.

variable(s)
specifies the analysis variables to group by.

OUTPUT Statement

Creates an output table that contains the results of PROC MDSUMMARY.

Restriction: You can specify one OUTPUT statement only.

Requirement: You must specify either the OUTPUT statement or at least one GROUPBY statement that specifies the OUT= option. You cannot specify the OUTPUT= statement with one or more GROUPBY statements that also specify the OUT= option.

Syntax

OUTPUT OUT=*libref.table name*;

Required Argument

OUT=*libref.table name*
specifies the two-level table name.

Requirement The table name must be a two-level name where *Libref* is a CAS engine libref.

PROC MDSUMMARY Output Data Sets

You can create output tables by using one OUTPUT statement or multiple GROUPBY statements with the OUT= option specified. You must specify either the OUTPUT statement, or one or more GROUPBY statements with the OUT= option specified. You cannot specify both an OUTPUT statement and a GROUPBY statement with the OUT= option specified. To produce multiple output tables, use multiple GROUPBY OUT= statements.

PROC MDSUMMARY does not display output. You can use PROC PRINT to display the output table.

Results: MDSUMMARY Procedure

Output Tables

PROC MDSUMMARY produces one or more output tables for each By group, defined by a set of variables listed in a GROUPBY statement.

A PROC MDSUMMARY table contains the following:

- One column for each basic statistic and one row for each combination of group-by level and analysis variable.
- If you are creating By groups, then two columns for each group-by variable are also included in the output table. One column is for the group-by variable itself. The other column is of a character type and has the same name as the group-by variable but with _f appended. The column contains the formatted value of the group-by variable.
- A column named _Column_, containing the name of the analysis variable, is included in the output. The _Column_ column denotes the variable for which statistics have been computed.

Group-by processing collects observations for analysis according to the formatted values of the group-by variables, with each unique combination of formatted group-by variable values forming one group-by level. Groups are not collected or processed in any particular order.

The statistics produced by MDSUMMARY are not configurable but are fixed and include:

Table 6.1 *Table of Statistic Produced by the MDSUMMARY Procedure*

Column Name	Statistic	Description
CSS	CSS	Corrected sum of squares

Column Name	Statistic	Description
_CV_	CV	Coefficient of variation
MAX	MAX	Maximum value
MEAN	MEAN	Arithmetic mean
MIN	MIN	Minimum value
NMISS	NMISS	Number of values that are missing
NOBS	NOBS	Total number of observations
PRT	PRT	p-Value for Student's t statistics
STD	STD	Standard deviation
STDERR	STDERR	Standard error of the mean
SUM	SUM	Sum
_T_	T	Student's t statistic
USS	USS	Uncorrected sum of squares
VAR	VAR	Variance

Examples: MDSUMMARY Procedure

Example 1: Compute Descriptive Statistics

Program

The following example shows you how to access your data with SAS Cloud Analytics Services (CAS), compute all statistics for each variable, and treat the entire input table as one group. The results are written to an output table.

```
caslib MyCasLib datasource=(srctype="path") path='your-file-path';/*1*/

libname mycas cas;/*2*/

proc casutil;/*3*/
   load data=sashelp.cars outcaslib="MyCasLib ";
   contents casdata="cars";
```

```
quit;

proc mdsummary data=mycas.cars;/*4*/
   var MSRP MPG_City;
   output out=mycas.mdsumstat;
run;

proc print data=mycas.mdsumstat;/*5*/
   var _Column_ _NObs_ _Mean_ _Max_ _Min_ _Std_  ;
   title 'Summary of MSRP and City Miles Per Gallon';
run;

proc casutil;/*6*/
   save casdata="mdsumstat" incaslib="MyCasLib" outcaslib="MyCasLib";
   list files incaslib="MyCasLib";
quit;
```

1 The CASLIB statement adds a session-scope caslib named MyCasLib. The caslib provides access to your data source. The DATASOURCE= option and the PATH= option provide connection information to your data source.

2 The LIBNAME statement creates a CAS engine libref. To run PROC MDSUMMARY and PROC PRINT in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names.

3 The CASUTIL procedure loads the data. The LOAD DATA= statement reads the file into memory. The table is now available for analytics. The CONTENTS statement reads the on-disk file, Cars, and displays the table metadata. This enables you to learn if the file has column names in the first row and the data types.

4 The MDSUMMARY procedure produces summary statistics for MPG_CITY and MSRP. The OUTPUT statement creates an in-memory table named Mycas.MdsumStat. The table includes a row of summary statistics for MSRP and MPG_City.

5 The MDSUMMARY procedure does not print output. Use the PRINT procedure to print the table Mycas.MdsumStat.

6 The CASUTIL procedure saves the table to the data source. The INCASLIB= option specifies the caslib that contains the file, and the OUTCASLIB= option specifies the caslib that the file is being made available to. The SAVE statement stored the data as UTF-8 when it created the SASHDAT file. The LIST FILES statement confirms that the in-memory table named Mdsumstat is saved in the data source.

Results

Output 6.1 *Column Information for the Cars Data Set*

The CASUTIL Procedure

Table Information for Caslib MYCASLIB

Table Name	Label	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
CARS	2004 Car Data	428	15	utf-8	05Apr2016:15:23:33	05Apr2016:15:23:33	No	No	No	No

Detail Information for cars in Caslib MYCASLIB.

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	428	68480	0	0	0	0	0	1	68480

Column Information for CARS in Caslib MYCASLIB

Column	Label	Type	Length	Formatted Length	Format
Make		char	13	13	
Model		char	40	40	
Type		char	8	8	
Origin		char	6	6	
DriveTrain		char	5	5	
MSRP		double	8	8	DOLLAR
Invoice		double	8	8	DOLLAR
EngineSize	Engine Size (L)	double	8	12	
Cylinders		double	8	12	
Horsepower		double	8	12	
MPG_City	MPG (City)	double	8	12	
MPG_Highway	MPG (Highway)	double	8	12	
Weight	Weight (LBS)	double	8	12	
Wheelbase	Wheelbase (IN)	double	8	12	
Length	Length (IN)	double	8	12	

Results: MDSUMMARY Procedure

Output 6.2 *Summary of MSRP and City Miles per Gallon*

Summary of MSRP and City Miles Per Gallon

Obs	_Column_	_NObs_	_Mean_	_Max_	_Min_	_Std_
1	MSRP	428	32774.85514	192465	10280	19431.716674
2	MPG_City	428	20.060747664	60	10	5.2382176386

Additional Information

- Caslibs provide a way to organize in-memory tables and an associated data source. They also provide a way to apply access controls to data. After a caslib is dropped, the in-memory tables are dropped too. Files in the caslib's data source are not removed or modified in any way. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- Tables that are saved from a caslib are saved in SASHDAT format by default.
- For documentation about the CASUTIL procedure syntax, see Chapter 5, “CASUTIL Procedure,” on page 109.
- For more examples of using the CASUTIL procedure to access and save data, see Chapter 1, “Accessing Data,” on page 1.

Example 2: Computing Descriptive Statistics with Group-By Variables

Program

The following examples shows you how to access your data with SAS Cloud Analytics Services (CAS), compute all statistics for each variable (treating the entire input table as one group), and computes all statistics for every unique combination of the formatted values of variables Make and Type. The results are written to an output table.

```
libname mycas cas;/*1*/

proc casutil incaslib="casuser" outcaslib="casuser" ;/*2*/
   load data=sashelp.cars;
   contents casdata="cars";
quit;

proc mdsummary data=mycas.cars;/*3*/
   var MPG_City;
   groupby / out=mycas.carsmpgcityall; /*4*/
   groupby make type / out=mycas.carsmaketype; /*5*/
run;

proc print data=mycas.carsmpgcityall;/*6*/
   var _Column_ _NObs_ _Mean_ _Std_ _Min_ _Max_;
   title 'Overall City Mileage';
run;

proc print data=mycas.carsmaketype;/**/
   var make Type  _Column_ _NObs_ _Mean_ _Std_ _Min_ _Max_;
   title 'City Mileage by Make and Type';
run;
```

1 The LIBNAME statement creates a CAS engine libref. To run PROC MDSUMMARY and PROC PRINT in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names.

2 The CASUTIL procedure loads the data into the default caslib, Casuser. The LOAD DATA= statement reads the file into memory. The table is now available for analytics. The CONTENTS statement reads the on-disk file, Cars, and displays the table metadata. This enables you to learn if the file has column names in the first row and the data types.

3 The MDSUMMARY procedure produces summary statistics for MPG_CITY.

4 The first GROUPBY statement with the OUT= option creates an in-memory table named Mycas.CarsMPGCityAll that calculates summaries based on all qualifying rows of the input table

5 The second GROUPBY statement with the OUT= option creates an in-memory table named Mycas,CarsMakeType that calculates summaries for rows grouped by Make and Type.

6 The MDSUMMARY procedure does not print output. The PRINT procedure prints the table Mycas.MdsumStat.

Results: MDSUMMARY Procedure

Output 6.3 *Overall Summary of City Mileage*

Overall City Mileage

Obs	_Column_	_NObs_	_Mean_	_Std_	_Min_	_Max_
1	MPG_City	428	20.060747664	5.2382176386	10	60

Output 6.4 *Summary of City Mileage by Make and Type*

City Mileage by Make and Type

Obs	Make	Type	_Column_	_NObs_	_Mean_	_Std_	_Min_	_Max_
1	Acura	SUV	MPG_City	1	17	.	17	17
2	Acura	Sedan	MPG_City	5	20.4	2.6076809621	18	24
3	Acura	Sports	MPG_City	1	17	.	17	17
4	Audi	Sedan	MPG_City	13	18.615384615	2.3642638579	14	23
5	Audi	Sports	MPG_City	4	19	2.7080128015	15	21
6	Audi	Wagon	MPG_City	2	16.5	2.1213203436	15	18
7	BMW	SUV	MPG_City	2	16	0	16	16
8	BMW	Sedan	MPG_City	13	19.230769231	0.8320502943	18	20
9	BMW	Sports	MPG_City	4	18.25	2.6299556397	16	21
10	BMW	Wagon	MPG_City	1	19	.	19	19
11	Buick	SUV	MPG_City	2	17	2.8284271247	15	19
12	Buick	Sedan	MPG_City	7	19.428571429	0.9759000729	18	20
13	Cadillac	SUV	MPG_City	2	15	1.4142135624	14	16
14	Cadillac	Sedan	MPG_City	4	18	0	18	18
15	Cadillac	Sports	MPG_City	1	17	.	17	17
16	Cadillac	Truck	MPG_City	1	13	.	13	13
17	Chevrolet	SUV	MPG_City	4	15.75	2.3629078131	14	19
18	Chevrolet	Sedan	MPG_City	15	22.266666667	4.0964560758	14	28
19	Chevrolet	Sports	MPG_City	2	18	0	18	18
20	Chevrolet	Truck	MPG_City	5	15.2	1.9235384062	13	18
21	Chevrolet	Wagon	MPG_City	1	22	.	22	22
22	Chrysler	Sedan	MPG_City	13	20.307692308	1.6525039276	18	22
23	Chrysler	Sports	MPG_City	1	17	.	17	17
24	Chrysler	Wagon	MPG_City	1	17	.	17	17
25	Dodge	SUV	MPG_City	1	15	.	15	15

Results: PROC CASUTIL CONTENTS Statement

Output 6.5 *Metadata for the Cars Table*

The CASUTIL Procedure

Table Information for Caslib CASUSER([redacted])

Table Name	Label	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
CARS	2004 Car Data	428	15	utf-8	04Apr2016:17:05:10	04Apr2016:17:05:10	No	No	No	No

Detail Information for cars in Caslib CASUSER([redacted]).

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	428	68480	0	0	0	0	0	1	68480

Column Information for CARS in Caslib CASUSER([redacted])

Column	Label	Type	Length	Formatted Length	Format
Make		char	13	13	
Model		char	40	40	
Type		char	8	8	
Origin		char	6	6	
DriveTrain		char	5	5	
MSRP		double	8	8	DOLLAR
Invoice		double	8	8	DOLLAR
EngineSize	Engine Size (L)	double	8	12	
Cylinders		double	8	12	
Horsepower		double	8	12	
MPG_City	MPG (City)	double	8	12	
MPG_Highway	MPG (Highway)	double	8	12	
Weight	Weight (LBS)	double	8	12	
Wheelbase	Wheelbase (IN)	double	8	12	
Length	Length (IN)	double	8	12	

Additional Information

- Caslibs provide a way to access in-memory tables and an associated data source. They also provide a way to apply access controls to data. In this example, the personal caslib Casuser is being used, so no CASLIB statement is needed.
- Tables that are saved from a caslib are saved in SASHDAT format by default.
- For documentation about the CASUTIL procedure syntax, see Chapter 5, "CASUTIL Procedure," on page 109.
- For more examples of using the CASUTIL procedure to access and save data, see Chapter 1, "Accessing Data," on page 1.

Example 3: Using Multiple Group-By Variables

Program

The following example loads a data set into CAS and computes all statistics for every unique combination of the formatted values of variables Make, Model, Type, and MPG_CITY. The results are written to an output table

```
proc casutil; /*1*/
   load data=sashelp.cars;
   contents casdata="cars";
quit;

libname mycas cas;/*2*/

proc mdsummary data=mycas.cars ;/*3*/
```

```
   var MPG_City;
   groupby make model type / out=mycas.carsmiles;
run;

proc print data=mycas.carsmiles;/*4*/
   var Make Model Type _Column_ _NObs_ _Min_ _Max_;
   title 'City Mileage for Make, Model, and Type';
run;
```

1 The CASUTIL procedure loads the data into the default caslib, Casuser. The LOAD DATA= statement reads the file into memory. The table is now available for analytics. The CONTENTS statement reads the on-disk file, Cars, and displays the table metadata. This enables you to learn if the file has column names in the first row and the data types.

2 The LIBNAME statement for the CAS engine creates a CAS libref. To run PROC MDSUMMARY and PROC PRINT in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names.

3 The MDSUMMARY procedure produces summary statistics for MPG_CITY, grouped by Make, Model, and Type. The OUT= option creates an in-memory table named Mycas.CarsMiles. The table includes a row of summary statistics for each unique combination of MPG_City, Make, Model, and Type.

4 The PRINT procedure prints the table Mycas.CarsMiles.

Results

Output 6.6 *PROC MDSUMMARY Output: City Mileage for Make, Model, and Type*

City Mileage for Make, Model, and Type

Obs	Make	Model	Type	_Column_	_NObs_	_Min_	_Max_
1	Audi	A4 3.0 Quattro 4dr manual	Sedan	MPG_City	1	17	17
2	Audi	A41.8T convertible 2dr	Sedan	MPG_City	1	23	23
3	Audi	A6 2.7 Turbo Quattro 4dr	Sedan	MPG_City	1	18	18
4	Audi	A6 3.0 Quattro 4dr	Sedan	MPG_City	1	18	18
5	Audi	TT 1.8 convertible 2dr (coupe)	Sports	MPG_City	1	20	20
6	Audi	TT 3.2 coupe 2dr (convertible)	Sports	MPG_City	1	21	21
7	BMW	325i 4dr	Sedan	MPG_City	1	20	20
8	BMW	330Ci 2dr	Sedan	MPG_City	1	20	20
9	BMW	330xi 4dr	Sedan	MPG_City	1	20	20
10	BMW	525i 4dr	Sedan	MPG_City	1	19	19
11	BMW	M3 convertible 2dr	Sports	MPG_City	1	16	16
12	BMW	M3 coupe 2dr	Sports	MPG_City	1	16	16
13	BMW	Z4 convertible 2.5i 2dr	Sports	MPG_City	1	20	20
14	Buick	LeSabre Custom 4dr	Sedan	MPG_City	1	20	20
15	Buick	LeSabre Limited 4dr	Sedan	MPG_City	1	20	20
16	Buick	Regal GS 4dr	Sedan	MPG_City	1	18	18
17	Buick	Rendezvous CX	SUV	MPG_City	1	19	19
18	Cadillac	SRX V8	SUV	MPG_City	1	16	16
19	Chevrolet	Cavalier 4dr	Sedan	MPG_City	1	26	26
20	Chevrolet	Malibu LT 4dr	Sedan	MPG_City	1	23	23
21	Chevrolet	Silverado SS	Truck	MPG_City	1	13	13
22	Chevrolet	Tahoe LT	SUV	MPG_City	1	14	14
23	Chrysler	Concorde LX 4dr	Sedan	MPG_City	1	21	21
24	Chrysler	Concorde LXi 4dr	Sedan	MPG_City	1	19	19
25	Chrysler	PT Cruiser 4dr	Sedan	MPG_City	1	22	22

Additional Information

- Caslibs provide a way to access in-memory tables and an associated data source. They also provide a way to apply access controls to data. In this example, the personal caslib Casuser is being used, so no CASLIB statement is needed. The in-memory table Mycas.CarsMiles is temporary, and is dropped when the session is ended. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- For documentation about the CASUTIL procedure syntax, see Chapter 5, “CASUTIL Procedure,” on page 109.
- For more examples of using the CASUTIL procedure to access and save data, see Chapter 1, “Accessing Data,” on page 1.

Example 4: Using Formats with Group-By Variables

Program

The following example defines two value formats, one numeric and the other character, and uploads them to an existing CAS session and applies the formats to two variables.

```
libname mycas cas;/*1*/

proc format casfmtlib='fmtlib' ;/*2*/
   value $flvrfmt
      'Chocolate'='Chocolate'
      'Vanilla'='Vanilla'
      'Rum','Spice'='Other Flavor';
   value agefmt (multilabel)
      15 - 29='below 30 years'
      30 - 50='between 30 and 50'
      51 - high='over 50 years';
run;

data mycas.cake;/*3*/
input LastName $ 1-12 Age 13-14 PresentScore 16-17
TasteScore 19-20 Flavor $ 23-32 Layers 34 ;
format age agefmt. flavor $flvrfmt.;
datalines;
Orlando     27 93 80  Vanilla    1
Ramey       32 84 72  Rum        2
Goldston    46 68 75  Vanilla    1
Roe         38 79 73  Vanilla    2
Larsen      23 77 84  Chocolate  .
Davis       51 86 91  Spice      3
Strickland  19 82 79  Chocolate  1
Nguyen      57 77 84  Vanilla    .
Hildenbrand 33 81 83  Chocolate  1
Byron       62 72 87  Vanilla    2
Sanders     26 56 79  Chocolate  1
Jaeger      43 66 74             1
Davis       28 69 75  Chocolate  2
Conrad      69 85 94  Vanilla    1
Walters     55 67 72  Chocolate  2
Rossburger  28 78 81  Spice      2
Matthew     42 81 92  Chocolate  2
Becker      36 62 83  Spice      2
Anderson    27 87 85  Chocolate  1
Merritt     62 73 84  Chocolate  1
;

proc mdsummary data=mycas.cake; /*4*/
   var TasteScore;
   groupby flavor / out=mycas.flav; /*5*/
   groupby flavor age / out=mycas.flag; /*6*/
run;
```

```
proc print data=mycas.flav; /*7*/
   var flavor _Column_ _NObs_ _Min_ _Max_ _Mean_;
title 'Taste Score for Cake Flavors and Participant''s Age';
title2 'GROUPBY Flavor';
run;

proc print data=mycas.flag;
   var flavor age _Column_ _NObs_ _Min_ _Max_ _Mean_;
title 'Taste Score for Cake Flavors and Participant''s Age';
title2 'GROUPBY Flavor and Age';
run;
```

1 The LIBNAME statement for the CAS engine creates a CAS engine libref. To run PROC MDSUMMARY and PROC PRINT in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names.

2 The FORMAT procedure creates the formats Flvrfmt and Agefmt. The CASFMTLIB= option adds the format library to the CAS session. It associates the format library with the CAS tables.

3 The DATA step creates the input data set. This DATA step runs in the SAS client session and not in CAS. However, the DATA step sends the results to CAS in the form of an in-memory CAS table. The CAS engine libref "Mycas" enables CAS processes to run on the data set.

4 The MDSUMMARY procedure computes summary statistics for cake tasting scores.

5 The first GROUPBY statement with the OUT= option creates an in-memory table named Mycas.Flav that is grouped by Flavor.

6 The second GROUPBY statement with the OUT= option creates an in-memory table named Mycas.Flag that is grouped by Flavor and Age.

7 The PRINT procedure prints the output data sets.

Results

Output 6.7 *PROC PRINT Output: Cake Flavors and Participant's Age Grouped by Flavor*

Taste Score for Cake Flavors and Participant's Age
GROUPBY Flavor

Obs	Flavor	_Column_	_NObs_	_Min_	_Max_	_Mean_
1		TasteScore	1	74	74	74
2	Chocolate	TasteScore	9	72	92	81.444444444
3	Other Flavor	TasteScore	4	72	91	81.75
4	Vanilla	TasteScore	6	73	94	82.166666667

Output 6.8 *PROC PRINT Output: Cake Flavors and Participant's Age Grouped by Flavor*

Taste Score for Cake Flavors and Participant's Age
GROUPBY Flavor and Age

Obs	Flavor	Age	_Column_	_NObs_	_Min_	_Max_	_Mean_
1	Chocolate	below 30 years	TasteScore	5	75	85	80.4
2	Chocolate	between 30 and 50	TasteScore	2	83	92	87.5
3	Other Flavor	below 30 years	TasteScore	1	81	81	81
4	Other Flavor	over 50 years	TasteScore	1	91	91	91
5	Vanilla	below 30 years	TasteScore	1	80	80	80
6	Chocolate	over 50 years	TasteScore	2	72	84	78
7	Vanilla	between 30 and 50	TasteScore	2	73	75	74
8	Vanilla	over 50 years	TasteScore	3	84	94	88.333333333
9		between 30 and 50	TasteScore	1	74	74	74
10	Other Flavor	between 30 and 50	TasteScore	2	72	83	77.5

Additional Information

- Caslibs provide a way to access in-memory tables and an associated data source. They also provide a way to apply access controls to data. In this example, the personal caslib Casuser is being used, so no CASLIB statement is needed. The in-memory tables Mycas.Flav and Mycas.Flag are temporary, and are dropped when the session ends. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- For documentation about the CASUTIL procedure syntax, see Chapter 5, "CASUTIL Procedure," on page 109.
- For more examples of using the CASUTIL procedure to access and save data, see Chapter 1, "Accessing Data," on page 1.

Example 5: Graph Summary Statistics Results Obtained from the MDSUMMARY Procedure

Program

The following example loads data into CAS, and creates a plot from the summarized results of the MDSUMMARY procedure.

```
proc casutil;      /* 1 */
   load data=sashelp.cars;
   contents casdata="cars";
quit;

libname mycas cas;/* 2 */

proc mdsummary data=mycas.cars;                    /* 3 */
   var mpg_highway;
   groupby origin type / out=mycas.mpghw_sum;
run;

ods graphics / width=4in;
title "Summarized Highway MPG";
```

```
proc sgpanel data=mycas.mpghw_sum;                    /* 4 */
   where origin in ("Asia" "USA");
   panelby origin / uniscale=row;
   format _mean_ 2.;
   vbar type / response=_mean_;
   rowaxis label="Summary MPG Values";
run;
title;
ods graphics / reset=all;
```

1 The LIBNAME statement for the CAS engine creates a CAS engine libref. To run PROC MDSUMMARY, PROC SGPANEL, and PROC PRINT in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names.

2 The CASUTIL procedure loads the data into the default caslib, Casuser. The LOAD DATA= statement reads the file into memory. The table is now available for analytics. The CONTENTS statement reads the on-disk file, Cars, and displays the table metadata. This enables you to learn if the file has column names in the first row and the data types.

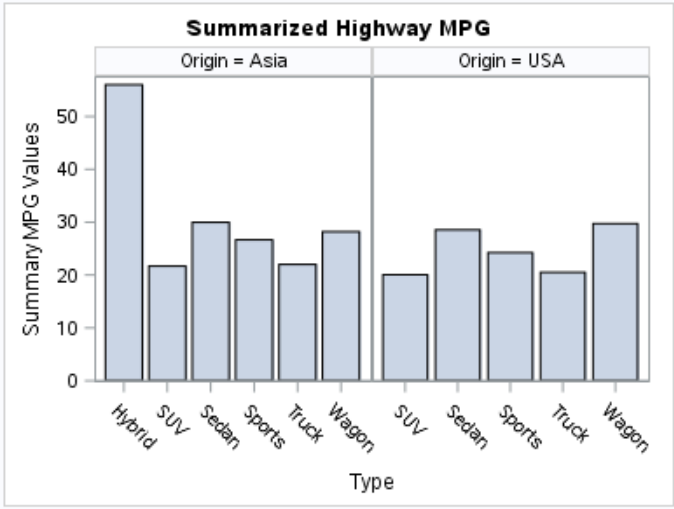
3 The MDSUMMARY procedure produces summary statistics for highway miles-per-gallon. The OUT= option in the GROUPBY statement creates a table in CAS. The table includes a row of summary statistics for each unique combination of origin and type.

4 The SGPANEL procedure plots the summarized results from the MDSUMMARY procedure. The procedure creates a parameterized vertical bar chart that shows the mean statistic for highway miles-per-gallon. The procedure subsets the data, comparing only the cars made in Asia and the U.S.A. The graph is paneled by country of origin.

Results

The SGPANEL procedure generates the following graph:

Output 6.9 *Graph Output*



Additional Information

- Caslibs provide a way to organize in-memory tables and an associated data source. They also provide a way to apply access controls to data. In this example, the default caslib CASUSER is being used, so no CASLIB statement is needed. The in-memory table Mycas.Mpghw_Sum is temporary, and is dropped when the session is ended. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- For documentation about the CASUTIL procedure syntax, see Chapter 5, "CASUTIL Procedure," on page 109.
- For more examples of using the CASUTIL procedure to access and save data, see Chapter 1, "Accessing Data," on page 1.
- For information about the SGPANEL procedure, see "SGPANEL Procedure" in *SAS ODS Graphics: Procedures Guide*.

Chapter 7

Platform Data Sources

Data Redundancy

The following table shows how several factors interact with respect to data redundancy. Data redundancy applies to distributed servers only.

Table 7.1 *Data Redundancy by Data Access Method, Data Source, and File Type*

Data Access Method	Caslib Data Source	Redundancy		File Formats
		SASHDAT files	Other files	

DATA step and
`PROC CASUTIL;`
`LOAD DATA=`
	The caslib data source isn't a factor for these data access methods.	Not applicable.	Based on COPIES= when the file is loaded.	SAS Data Sets

Data Access Method	Caslib Data Source	Redundancy		File Formats
		SASHDAT files	Other files	
PROC CASUTIL; LOAD CASDATA=	DNFS	COPIES= is not needed, surviving nodes reread the file.	Based on COPIES= when the file is loaded.	SASHDAT and CSV files
	HDFS	Based on COPIES= when the file was saved.	Based on COPIES= when the file is loaded.	SASHDAT and CSV files
	PATH	Based on COPIES= when the file is loaded.	Based on COPIES= when the file is loaded.	SASHDAT, CSV, and PC Files
	Data Connectors	Not applicable.	Based on COPIES= when the file is loaded.	Based on the data connector.

Dictionary

HDFS Data Source

Specifies a Hadoop Distributed File System directory for loading and saving files that the SAS Cloud Analytic Servicescontroller can access. SAS Cloud Analytic Services must be co-located with the Hadoop cluster to use this data source type.

Applies to: Distributed servers only, CASLIB statement

Example: Add a caslib to access files from the **/vapublic** directory in HDFS.

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

proc casutil incaslib="public" outcaslib="public";
   list files;
quit;
```

Syntax

Data Source Arguments

ENCRYPTIONPASSWORD="*string*"
specifies a password for encrypting or decrypting stored data.

PATH="*/directory-path*"
specifies the fully qualified path to the directory to use as a data source. Notice that the PATH= argument is specified outside of the parenthesis for the DATASOURCE= argument.

SRCTYPE="HDFS"
specifies that the data source is an HDFS directory that is co-located with SAS Cloud Analytic Services.

Requirement The SRCTYPE= argument is required.

DNFS Data Source

Specifies a server-side directory for loading and saving files that the SAS Cloud Analytic Services controller can access. The directory must be mounted by every machine that is used by the server.

Applies to: Distributed servers only, CASLIB statement

Example: Add a caslib to access files from the `/data01` directory on the controller.

```
caslib dnfsds datasource=(srctype="dnfs") path="/net/fileserver/";

proc casutil incaslib="dnfsds" outcaslib="dnfsds";
   list files;
quit;
```

Syntax

Data Source Arguments

ENCRYPTIONPASSWORD="*string*"
specifies a password for encrypting or decrypting stored data.

PATH="*/directory-path*"
specifies the fully qualified path to the directory to use as a data source. Notice that the PATH= argument is specified outside of the parenthesis for the DATASOURCE= argument.

SRCTYPE="DNFS"
specifies that the data source is a directory that is mounted by every machine that is used for SAS Cloud Analytic Services.

Requirement The SRCTYPE= argument is required.

Details

DNFS is an abbreviation for distributed network file system. This data source type provides support for distributed data access to NFS directories. Several systems such as MapR-FS, EMC Isilon, and others provide high-availability, replicated, high-performance, stand-alone storage clusters with an NFS interface. These systems offer popular alternatives to Hadoop. DNFS provides a good alternative for deployments where the server is not co-located with Hadoop and yet must provide similar capabilities.

DNFS can also be used to access NFS-mounted directories from standard UNIX or Linux file systems.

The design principle is that NFS-mounted directories are accessed concurrently by each controller node and worker node in a distributed server. This is why the directory path for a DNFS caslib must be mounted on every machine. DNFS performs parallel read and write for SASHDAT and CSV files that are stored in the directory path specified for the caslib.

SAS LASR Analytic Server

Specifies the connection options for loading data from the SAS LASR Analytic Server into SAS Cloud Analytic Services.

Applies to: CASUTIL procedure, CASLIB statement

Examples: Add a caslib to access data from a SAS LASR Analytic Server.

```
caslib publiclasr datasource=(
  srctype="lasr"
  server="gridhost.example.com"
  port=10050
  signer="https://webserver.example.com/SASLASRAuthorization"
  username="sasdemo"
  password="secret"
);
```

Load a table from a SAS LASR Analytic Server into SAS Cloud Analytic Services.

```
proc casutil incaslib="publiclasr";
   load casdata="epa_cars" casout="epa_cars"
        importoptions=(filetype="lasr" varchars="true");
run;
```

Syntax

Data Source Arguments

METALIB="*metadata-libref*"
specifies the libref name for the SAS LASR Analytic Server engine library.

PASSWORD="*string*"
specifies the password for the identity in the USERNAME= option.

PORT=*integer*
specifies the network port that the SAS LASR Analytic Server listens on.

SERVER="*host-name*"
specifies the host name or IP address of the SAS LASR Analytic Server.

SIGNER="*authorization-web-service-uri*"
specifies the URI for the SAS LASR Authorization web service. This is specified in the form SIGNER="https://server.example.com/SASLASRAuthorization".

SRCTYPE="LASR"
specifies that the data source is a SAS LASR Analytic Server.

Requirement The SRCTYPE= argument is required.

TAG=*server-tag*
specifies a tag that is used to qualify the names of SAS LASR Analytic Server tables that are accessed using this caslib.

USERNAME="*user-ID*"
specifies an identity that is authorized to access data in the SAS LASR Analytic Server.

File Type Arguments

COMPPGM="*string*"
specifies an expression for each variable that you included in the COMPVARS option. End the expression for each variable with a semicolon.

COMPVARS=("*computed-variable-1*" <, "*computed-variable-2*", ...>)
specifies the names of the computed variables to create. Specify an expression for each parameter in the COMPPGM option.

FILETYPE="LASR"
specifies the file type.

Requirement The FILETYPE= argument is required.

PARALLELMODE="FALLBACK" | "FORCE" | "NONE"
specifies how the table is transferred from SAS LASR Analytic Server to SAS Cloud Analytic Services when both servers are distributed servers. If either server is running as a single-machine server, then parallel data transfer is not possible and that is equivalent to NONE.

FALLBACK
specifies that the worker nodes try to establish communication with each other. If the worker nodes cannot connect, the operation falls back to a serial data transfer between the SAS LASR Analytic Server root node and the SAS Cloud Analytic Services controller node. Serial data transfer is slower than parallel data transfer.

FORCE
specifies that the worker nodes try to establish communication with each other. If the worker nodes cannot connect to perform a parallel data transfer, then the load request fails.

NONE
specifies to perform a serial data transfer between the SAS LASR Analytic Server root node and the SAS Cloud Analytic Services controller node.

Default FALLBACK

PRESERVEORDER=TRUE | FALSE
when set to True, the rows are inserted into the new table in the same order as they are received from the SAS LASR Analytic Server. Creating the table is less efficient when this parameter is used.

Default FALSE

VARCHARS=TRUE | FALSE
when set to True, variable-length strings are used for character variables.

Default FALSE

VARS=("*string-1*" <, "*string-2*", ...>)
specifies the variables to use in the action.

WHERE="*where-expression*"
specifies an expression for subsetting the input data.

Details

Arguments Summary

Table 7.2 *Summary of Arguments for SAS LASR Analytic Server*

Argument	Valid Data Source Options in the CASLIB Statement	Valid Import Options in the CASUTIL Procedure
COMPPGM=		•
COMPVARS=		•
FILETYPE=		Required
METALIB=	•	
PARALLELMODE=		•
PASSWORD=	•	•
PORT=	Required	•
PRESERVEORDER=		•
SRCTYPE=	Required	
SERVER=	Required	
SIGNER=	•	
USERNAME=	•	•
VARCHARS=		•
VARS=		•
WHERE=		•

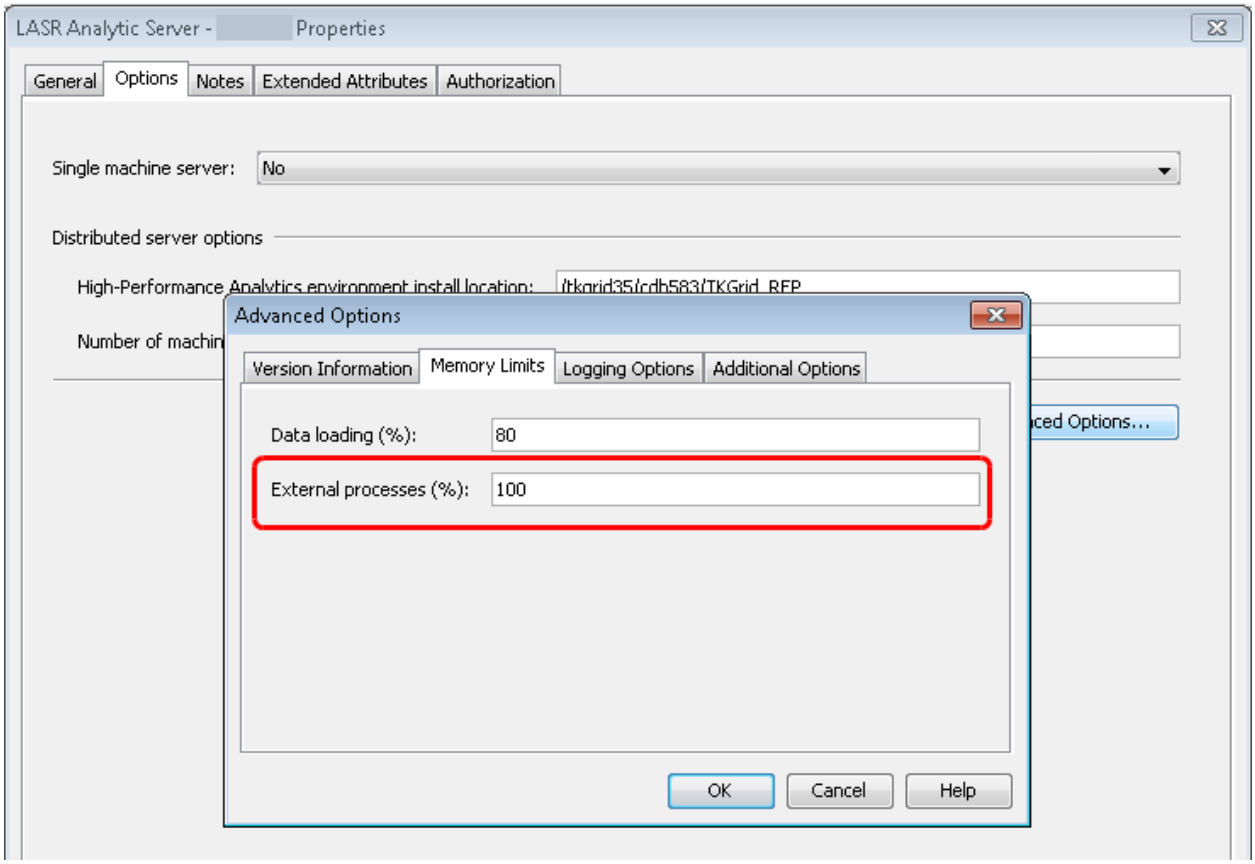
Considerations for Memory Use and Parallel Data Transfer

A distributed SAS LASR Analytic Server monitors the amount of memory in use on all of its nodes, and will not transfer data in parallel to SAS Cloud Analytic Services when the amount that is in use exceeds the EXTERNALMEM= setting. (The memory use monitoring applies to parallel data transfer with any external process.) By default, a SAS LASR Analytic Server stops the data transfer when the memory use on any host exceeds 75% of memory.

TIP Serial data transfer is slower, but is not subject to the memory monitoring during a data transfer.

If the SAS LASR Analytic Server and CAS are on separate clusters, you can set the threshold to 100% to transfer tables to CAS. The threshold can be set when the SAS LASR Analytic Server is started by specifying a value for the EXTERNALMEM= option. Or, you can modify the threshold for a running server by specifying a new value in the SERVERPARM statement of the IMSTAT procedure.

The value can also be set with SAS Management Console and then restarting the SAS LASR Analytic Server.



Example

This example shows how to load a table from a SAS LASR Analytic Server and use the VARS= options. The sample Epa_cars data is available from the SAS Visual Analytics 7.3 documentation page at http://support.sas.com/documentation/onlinedoc/va/index.html.

```
caslib valasr datasource=(
  srctype="lasr"
  /* ...data source options... */
);

cas casauto sessopts=(caslib="valasr");                    /* 1 */


proc casutil;
   list files;                                             /* 2 */
```

```
   load casdata="epa_cars" casout="epa_cars"
        importoptions=(
           filetype="lasr"
           varchars="true"
           vars=("model_year" "vehicle_manufacturer_name")      /* 3 */
        );

   contents casdata="epa_cars";
quit;
```

1 The CAS statement is used to set the active caslib for the Casauto session to Valasr explicitly.

2 The LIST FILES statement shows the tables that are in SAS LASR Analytic Server.

3 The VARS= option is used to subset the columns that are read from SAS LASR Analytic Server and loaded into SAS Cloud Analytic Services.

Path Data Source

Specifies a server-side directory for loading and saving files that the SAS Cloud Analytic Services controller can access.

Applies to: CASLIB statement

Example: Add a caslib to access files from the `/data01` directory on the controller.

```
caslib pathds datasource=(srctype="path") path="/data01";

proc casutil incaslib="pathds" outcaslib="pathds";
   list files;
quit;
```

Syntax

Data Source Arguments

ENCRYPTIONPASSWORD="*string*"
specifies a password for encrypting or decrypting stored data.

PATH="*/directory-path*"
specifies the fully qualified path to the directory to use as a data source. Notice that the PATH= argument is specified outside of the parenthesis for the DATASOURCE= argument.

SRCTYPE="PATH"
specifies that the data source is a directory that is accessible to the SAS Cloud Analytic Services controller.

Requirement The SRCTYPE= argument is required.

Chapter 8

Platform File Types

Dictionary

Delimited Files (CSV)

Specifies the file type options for loading data from delimited files.

Applies to: CASUTIL procedure

Example: Load a Latin1 encoded CSV file into SAS Cloud Analytic Services.

```
proc casutil;
   load casdata="iris.csv"
        importoptions=(filetype="csv" encoding="latin1");
run;
```

Syntax

File Type Arguments

DELIMITER="*string*"
specifies the character to use as the field delimiter.

Default ","

ENCODING="*string*"
specifies the text encoding of the file. If the file is not encoded in UTF-8 or 7-bit ASCII, then specify the encoding.

Default utf-8

FILETYPE="CSV"
specifies the file type.

Requirement The FILETYPE= argument is required.

GETNAMES=TRUE | FALSE
when set to True, the values in the first line of the file are used as variable names.

Default TRUE

GUESSROWS=*integer*
specifies the number of rows to scan in order to determine data types for variables.

Default 20

LOCALE="*string*"
specifies the locale for interpreting data in the file.

STRIPBLANKS=TRUE | FALSE
removes leading and trailing blanks from character variables.

Default FALSE

VARCHARS=TRUE | FALSE
when set to True, variable-length strings are used for character variables.

Default TRUE

VARS=((*casvardesc-1*) <, (*casvardesc-2*, …)>)
specifies the names, types, formats, and other metadata for variables.

FORMAT="*string*"
specifies the format to apply to the variable.

FORMATTEDLENGTH=*integer*
specifies the format field length plus the format precision length.

LABEL="*string*"
specifies the descriptive label for the variable.

LENGTH=*integer*
specifies the unformatted length of the variable. This parameter applies to fixed-length character variables (type="CHAR") only.

Default 8

NAME="*string*"
specifies the name for the variable.

NFD=*integer*
specifies the format precision length.

NFL=*integer*
specifies the format field length.

TYPE="CHAR" | "DOUBLE" | "VARCHAR"
specifies the data type for the variable.

Details

Delimited files can be read from caslibs with the following data source types:

- DNFS
- HDFS

- PATH

Example

By default, SAS Cloud Analytic Services expects to find column names in the first line of the file. If you have a file that does not include names, you must specify GETNAMES=FALSE. You might also prefer to specify column names when you load the data.

File 8.1 *Sample File Contents*

```
Masculin;André;14,00;69,00;112,50
Masculin;Benoît;14,00;63,50;102,50
Masculin;Kévin;12,00;57,30;83,00
```

Example Code 1 *Load a CSV File and Specify Column Names*

```
options validvarname=any;                                          /* 1 */

cas casauto sessopts=(caslib="casuser");                           /* 2 */

proc casutil;
  load casdata="class_fr.csv" casout="class_fr"
       importoptions=(                                             /* 3 */
          filetype="csv"
          encoding="utf8"
          delimiter=";"
          getnames=false
          locale="Fr_fr"
          vars=("sexe", "nom", "âge", "la taille", "poids")
       );

  contents casdata="class_fr";
quit;

libname mycas cas caslib="casuser";                                /* 4 */

proc print data=mycas.class_fr(obs=3);
run;
```

1 The VALIDVARNAME= system option is set to ANY so that column names can include national characters.

2 The CAS session, named Casauto, is set to use the Casuser caslib as the active caslib.

3 The IMPORTOPTIONS= option is used to describe how the CASUTIL procedure should read the Class_fr.csv file, including the encoding, locale, and column names to use for the table. The locale parameter is effective only when reading a CSV file from a caslib with a data source of PATH or DNFS. In this example, the locale enables CAS to use a comma as the decimal separator.

4 The Mycas libref is assigned to use the CAS engine. The CASLIB= option is used to bind the libref to the Casuser caslib.

Output 8.1 *Column Information from the CONTENTS Statement*

Column Information for CLASS_FR in Caslib CASUSER( )			
Column	**Type**	**Length**	**Formatted Length**
sexe	varchar	8	8
nom	varchar	10	10
âge	double	8	12
la taille	double	8	12
poids	double	8	12

Output 8.2 *Three Rows from the Class_Fr Table*

Obs	sexe	nom	âge	la taille	poids
1	Masculin	André	14	69.0	112.5
2	Masculin	Benoît	14	63.5	102.5
3	Masculin	Kévin	12	57.3	83.0

SASHDAT Files

Specifies the file type options for loading data from SASHDAT files.

Applies to: CASUTIL procedure

Example: Load an encrypted SASHDAT file into SAS Cloud Analytic Services.

```
proc casutil;
   load casdata="cars.sashdat"
        importoptions=(filetype="hdat" encryptionPassword="pasquotank");
run;
```

Syntax

File Type Arguments

FILETYPE="HDAT"
specifies the file type.

Requirement The FILETYPE= argument is required.

ENCRYPTIONPASSWORD="*string*"
specifies a password for encrypting or decrypting stored data. You can use this option to override an ENCRYPTIONPASSWORD= value that is set with the CASLIB statement.

Details

SASHDAT files can be read from caslibs with the following data source types:

- DNFS
- HDFS
- PATH

Instead of specifying the ENCRYPTIONPASSWORD= option on a file-by-file basis, you can use it as a data source option when you add a caslib. For an example, see Encrypt Tables in a Caslib on page 104.

Chapter 9

Data Connectors

Working with SAS Data Connectors

To access and load data, the administrator must create and configure data connectors for SAS Cloud Analytic Services (CAS). Data connectors contain connection information and data-source specifics to connect with data sources, such as Oracle or SAS data sets. For instructions on how to configure data connectors, see the SAS Viya for Linux: Deployment Guide.

There are two methods to load data into CAS. You can load data serially with a data connector, or you can load data in parallel with a data connect accelerator. Data connect accelerators work with the SAS Embedded Process and must be licensed separately.

You can associate data connectors that require logins with a domain on the CAS server. When users connect to the data source through a data source name (DSN), the domain name is used to retrieve user credentials that are associated with that data connector. The credentials are then passed to the third-party data source. User credentials are stored on your system and are accessible by the CAS server.

Data connectors can also contain optional information to control CAS data source behavior. Data connectors form the foundation for connectivity to a third-party database or data source. You can assign privileges that control user access to the data. However, relational databases provide authorization that limits the operations that can be performed on the data. CAS respects authorizations that are defined and enforced by a third-party database or data source. Authorizations that are defined on a third-party database overrule permissions and privileges that are set in CAS.

Be aware that, if you save the contents of a SAS data set to a table in a database management system (DBMS), the resulting database table might contain different data types than the original SAS data set. In addition, any formats on the SAS data set that cannot be mapped to an equivalent DBMS type are not preserved. Verify that your data is what you expect after copying from SAS to a DBMS. For more information about what SAS data types and functions can be preserved, refer to *SAS/ACCESS for Relational Databases: Reference* for the target database.

Quick Reference for Data Connector Syntax

This table shows the syntax, supported file types (if applicable), and an example for each data source. For path-based data source syntax, see the DATASOURCE= parameter for the CASLIB statement on page 91.

Table 9.1 *Data Source Types and Options*

Data Connector	srcType= Type	Option Syntax	Example

Amazon Redshift	redshift	"SAS Data Connector to Amazon Redshift" on page 173	`caslib RScaslib desc='Amazon Redshift Caslib'`
			`dataSource=(srctype='redshift',`
			`server='RSserver'`
			`username='user1',`
			`password='myPwd',`
			`database='rsdatabase');`

DB2	db2	"SAS Data Connector to DB2" on page 181	`caslib db2caslib desc='DB2 Caslib'`
			`datasource=(srctype='db2'`
			`username='myusr1'`
			`password='myPwd'`
			`database="sample");`

Data Connector	srcType= Type	Option Syntax	Example

Hadoop	hadoop	"SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop" on page 187	`caslib hvcaslib desc="Hadoop Caslib"`
			`datasource=(srctype="hadoop",`
			`dataTransferMode="parallel",`
			`hadoopjarpath="/data/cdh58/sdm/lib",`
			`hadoopconfigdir="/data/cdh58/sdm/conf",`
			`username="hadoopuser",`
			`server="hive01.example.com",`
			`schema="default");`

Impala	impala	"SAS Data Connector to Impala" on page 201	`caslib imcaslib desc="Impala Caslib"`
			`datasource=(srctype="impala",`
			`username="impalauser",`
			`server="impala01.example.com");`

Microsoft SQL Server	sqlserver	"SAS Data Connector to Microsoft SQL Server" on page 208	`caslib sqlservercaslib desc='Microsoft SQL Server Caslib'`
			`dataSource=(srctype='sqlserver'`
			`username='user1'`
			`password='myPwd'`
			`sqlserver_dsn="SQLserverDSN-name");`

ODBC	odbc	"SAS Data Connector to ODBC" on page 215	`caslib odbccaslib desc="ODBC caslib"`
			`datasource=(srctype="odbc"`
			`username="user1"`
			`password="password1"`
			`database="dbodbc");`

Oracle	oracle	"SAS Data Connector to Oracle" on page 222	`caslib oraclecaslib desc="Oracle Caslib"`
			`datasource=(srctype="oracle",`
			`username="user1",`
			`password="password1",`
			`path="//machine.lnx.com:1521/exadat");`

PC Files	path	"SAS Data Connector to PC Files" on page 228	`caslib pcfilescaslib desc='PCFiles Caslib'`
			`datasource=(srctype='path')`
			`path="/mytest/customer";`

PostgreSQL	postgres	"SAS Data Connector to PostgreSQL" on page 231	`caslib postgrescaslib desc='PostgreSQL Caslib'`
			`datasource=(srctype='postgres'`
			`server='PGserver'`
			`username='user1'`
			`password='myPwd'`
			`database="PGdatabase-name");`

SAP HANA	hana	"SAS Data Connector to SAP HANA" on page 238	`caslib halib sessref=mysess`
			`datasource=(srctype='hana',`
			`dataTransferMode='serial',`
			`server='hana server',`
			`username='user1',`
			`password='myPwd',`
			`tableType='column'`
			`schema='hana schema'`
			`instance='00');`

Data Connector	srcType= Type	Option Syntax	Example
SAS Data Sets	path	"SAS Data Connector to SAS Data Sets" on page 253	`caslib sasdatasetcaslib desc='SASDataSets Caslib' datasource=(srctype='path') path="/mytest/customer";`
SPD Engine Files	spde	"SAS Data Connector to SPD Engine Files and SAS Data Connect Accelerator for SPD Engine Files" on page 244	`caslib spdecaslib desc='SPD Engine Files Caslib' dataSource=(srctype='spde', mdfpath="/myMDFpath");`
Teradata	teradata	"SAS Data Connector to Teradata and SAS Data Connect Accelerator for Teradata" on page 259	`caslib TDcaslib desc='Teradata Caslib' datasource=(srctype='teradata' username='user1' password='password1' server="teradataServer");`

Where to Specify Data Connector Options

The options that you specify for your data connector are specific to your data source. In general, you specify options for data connectors in your SAS code as described in this section.

Note: You can specify data connector options for all data connectors except SAS Data Connector to PC Files or SAS Data Connector to SAS Data Sets. For these you would instead use IMPORTOPTIONS= to read data and EXPORTOPTIONS= to save data.

When you add a caslib, specify data connector options within the DATASOURCE= option. Use the DATASOURCE= option in the CASLIB statement. If the server is named Teraserver, the database is named Teradatabase, and a schema is named MySchema, here is how the caslib can be added.

```
/* CASLIB statememt */
caslib simple
  sessref=mysess
  dataSource=(srcType="teradata",
              dataTransferMode="parallel",
              server="teraserver",
              database="teradatabase",
              schema="mySchema");
```

For PROC CASUTIL, specify data connector options for DATASOURCEOPTIONS= in a CONTENTS, LIST, LOAD, or SAVE statement.

```
proc casutil;
   list files incaslib="tdlib" dataSourceOptions=(schema="tdSchema");
   contents casdata="cars" vars((name="make"), (name="model"))
      dataSourceOptions=(schema="tdSchema2");
   load casdata="cars" incaslib="tdlib" casout="cars_CAS"
      vars=((name="make"), (name="model"))
      dataSourceOptions=(statusinterval=5);
```

```
   save casdata="cars" incaslib="tdlib" casout="cars_CAS"
      dataSourceOptions=(schema="tdSchema");
quit;
```

For more information about using SAS code that is related to data connectors, see these sections.

- CASLIB Statement
- CASUTIL Procedure

For more information about data connectors, see these sections.

- "SAS Data Connector to Amazon Redshift"
- "SAS Data Connector to DB2"
- "SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop"
- "SAS Data Connector to Impala"
- "SAS Data Connector to Microsoft SQL Server"
- "SAS Data Connector to ODBC"
- "SAS Data Connector to Oracle"
- "SAS Data Connector to PC Files"
- "SAS Data Connector to PostgreSQL"
- "SAS Data Connector to SAP HANA"
- "SAS Data Connector to SAS Data Sets"
- "SAS Data Connector to SPD Engine Files and SAS Data Connect Accelerator for SPD Engine Files"
- "SAS Data Connector to Teradata and SAS Data Connect Accelerator for Teradata"

You can also specify data connector parameters using CAS actions. For more information, see "Tables Action Set" in *SAS Viya: System Programming Guide*.

Using Wildcards for Pattern Matching

Overview of Wildcards

There are instances when you might use wildcards to facilitate pattern matching. For example, you might use wildcards to see a listing of all tables that begin with the string "aBc". To do this, you might specify this pattern to match:

```
path='aBc%'
```

You can use wildcards in the path= parameter for the fileInfo action. Case sensitivity is based on the behavior of your data source. You can use multiple wildcard characters within a pattern.

Here are the available wildcard characters:

%	matches any number of characters.
_	matches a single character. You can include more than one '_' in a pattern.
\	escapes a wildcard character so that it is treated as a literal character instead of a wildcard in a pattern.

If you have filenames that contain underscores, escape the '_' character with the '\'. For example, this pattern searches for all files that begin with 'aBc_':

```
path='aBc\_%'
```

Using Wildcards with SAS Macros

The SAS Macro language also uses the '%' character to indicate variables to resolve during program compilation. When you use the '%' character for pattern matching, include the value in single quotations marks. If you enclose a value in double quotation marks, the SAS Macro parser tries to resolve a macro variable name. For example, the code **path="testmisc/prdsa%e.csv"** results in an attempt to resolve a macro variable, E, and typically results in an error. Instead, specify the code as **path='testmisc/prdsa%e.csv'**.

Using Data Transfer Modes with Data Connectors for DBMS Data Sources

Overview

In CAS, SAS data connectors and data connect accelerators provide access to data in a database management system (DBMS) or distributed system using one of these modes of data transfer:

- serial (data connectors), which includes multinode
- parallel (data connect accelerators)

In *serial* data transfer mode, the data connector makes a single connection to the data source—Hadoop or a database management system (DBMS)—using a single data stream to the DBMS. This allows Read and Write access to data.

Data connectors that use serial data transfer mode can take advantage of *multinode* mode. This mode uses multiple CAS nodes to connect to a data source, which helps speed performance. A CAS controller node controls data transfer to and from worker nodes through **n** concurrent connections with the data source. This controller node directs each CAS worker node on how to query the source data to obtain the needed data. Each worker node therefore connects directly to the data source independently. As a result, multiple data streams can move data simultaneously.

In *parallel* mode, each CAS node connects directly to a data source node for parallel Read and Write. To use this mode, you must have access to a data connect accelerator. Data access is fastest with this method, as SAS Embedded Process can connect to individual slices of data on a data source node.

How to Implement Data Transfer Modes

By default, all SAS data connectors use serial data transfer. To use multinode data transfer, specify a value other than 1 for the NUMREADNODES= and NUMWRITENODES= options.

To specify how many nodes to use for serial data transfer using multiple worker nodes, you specify a specific value.

- Set NUMREADNODES= and NUMWRITENODES= to a value other than 1 to enable multinode data transfer.
- Multinode can use only the number of nodes that are available. So if you specify five nodes but only four are available, then four nodes are used.
- If you specify NUMREADNODES=0 and NUMWRITENODES=0, then all available nodes are used for multinode data transfer.

Data connect accelerators use the DATATRANSFERMODE= option to specify serial or parallel data transfer. When you specify DATATRANSFERMODE="PARALLEL", data transfer uses parallel mode. When you specify DATATRANSFERMODE="AUTO", SAS attempts data transfer in parallel first and then in serial mode if parallel transfer fails. With serial data transfer, if the values for NUMREADNODES= and NUMWRITENODES= are different from 1, then data transfer uses multiple nodes to perform the transfer.

About Loading Data When Using Multinode Data Transfer

For multinode data loads, SAS checks the target data source table for a numeric variable. It takes the first numeric variable that it finds and uses those values to divide the table into slices. Division is accomplished by using the MOD function and the number of nodes that you specify. Therefore, the higher the cardinality of the numeric variable, the easier the data can be divided into slices.

Here is the order of data types that SAS uses to divide data into slices for multinode Read.

- INT (includes BIGINT, INTEGER, SMALLINT, TINYINT)
- DECIMAL
- NUMERIC
- DOUBLE

Dictionary

SAS Data Connector to Amazon Redshift

SAS Data Connector to Amazon Redshift enables you to load data from Amazon Redshift into SAS Cloud Analytic Services. Data connector options are used in the context of different statements that connect your data in Amazon Redshift with CAS.

Valid in: CASLIB statement
PROC CASUTIL statements (see options for details)

Examples: Establish a connection between your Amazon Redshift database and SAS Cloud Analytic Services.

```
caslib redshiftcaslib desc='Amazon Redshift Caslib'
     dataSource=(srctype='redshift'
                 server='RSserver'
                 username='user1'
                 password='myPwd'
                 database="rsdatabase-name");
```

Overriding the user and password values.

```
proc casutil;
   load casdata="mycas.pgexamp" casout="myRSdata" casuser dataSourceOptions=(
         username='user5'
         password='myPwd');
quit;
```

Syntax

Data Connector Options for Amazon Redshift

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

AUTHENTICATIONDOMAIN="*domain*"
specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
Alias	AUTHDOMAIN=
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.

CATALOG="*catalog-name*"
specifies a logical catalog name. The logical name can be any user-defined name. This name is displayed in the Catalog column for all tables in the results from the CONTENTS statement in PROC CASUTIL.

Valid in	CASLIB statement
	PROC CASUTIL: CONTENTS, LOAD, and LIST statements
Default	Active caslib

CHARMULTIPLIER=*value*
specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Default	1.0
Range	1.0–5.0

CLIENT_ENCODING="*encoding*"
specifies the encoding that the data source client uses to transmit CHAR and VARCHAR data. Typically, this corresponds to the LANG environment variable.

If you have configured the underlying ODBC driver to use a different encoding than the LANG environment variable, set CLIENT_ENCODING= to the corresponding encoding value.

The CLIENT_ENCODING= value is independent of the NLS_LANG environment variable. For encodings other than UTF-8, the data is transcoded into UTF-8 when the data is loaded into CAS.

Valid in	CASLIB statement
Default	UTF-8

CONOPTS="*connection-options*"
specifies optional connection options that you pass to the DBMS.

Connection options that you specify in statements other than the CASLIB statement override any connection options that were set when a caslib was added.

Valid in	CASLIB statement
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Default	none
Example	`conopts="QueryTimeout=0"`

DATABASE="*database-name*"
specifies the name of the database.

When you specify a value for DATABASE=, the table name is qualified with the database name. For example, if you set DATABASE="myDB" and you want to access table Studydata, the table myDB.Studydata is accessed.

Valid in	CASLIB statement [Required]
	PROC CASUTIL: LOAD statement
Default	none
Interaction	If you supply values for SCHEMA= and DATABASE=, the value for SCHEMA= is used to qualify a table name.

DRIVER_VENDOR="SAS" | "DATADIRECT" | "AMAZON"
specifies the name of the third-party vendor that supports the Amazon Redshift driver.

By default, the Amazon Redshift data connector uses the SAS version of the DataDirect Amazon Redshift ODBC driver that is included with SAS/ACCESS Interface to Amazon Redshift. This corresponds to the value SAS.

Valid in CASLIB statement

Default SAS

INSERTBUFF="*number-of-rows*"
specifies the number of rows per block of data to save to a table in the data source.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default Calculated automatically based on row size

NUMREADNODES=*number*
specifies the number of CAS worker nodes to use for a multinode load of data into CAS. Setting this option to a value other than 1 enables a data connector to use multiple nodes to load data into CAS. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: LOAD statement

Default 1

NUMWRITENODES=*number*
specifies the number of CAS worker nodes to use for a multinode save of data from CAS into a data source table. Setting this option to a value other than 1 enables a data connector to use multiple nodes to save data to a data source table. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: SAVE statement

Default 1

PASSWORD="*password*"
specifies the DBMS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases PASS=

PWD=

Default none

Requirement If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

READBUFF="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default Calculated automatically based on row size

REDSHIFT_DSN="*Amazon-Redshift-datasource-name*"
specifies the data source name.

With this option, you can use an Amazon Redshift data source that you have previously configured in an odbc.ini file.

Valid in CASLIB statement [Required]

Default none

Requirement Set the ODBCSYSINI or ODBCINI environment variable. If the ODBCSYSINI environment variable is set, it must point to the full path of the directory that contains the configured odbc.ini and odbcinst.ini files. If ODBCSYSINI is not set, set the ODBCINI environment variable to the full path of the directory that contains the configured odbc.ini file.

SCHEMA="*schema-name*"
specifies the schema name to use for the connection to the database or data source.

When you specify a value for SCHEMA=, the table name is qualified with the schema name. For example, if you set SCHEMA="mySchema" and you want to access table Studydata, the table mySchema.Studydata is accessed. Specify a non-blank value when you set a value for SCHEMA=.

If you specify a value for SCHEMA= in a statement other than the CASLIB statement, this value overrides any value of SCHEMA= that was set when the caslib was added.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Default none

Interaction If you specify values for SCHEMA= and DATABASE=, the value for SCHEMA= is used to qualify the table name.

SERVER="*server-identifier*"
specifies the server identifier for the DBMS server.

Valid in CASLIB statement [Required]

Default none

SRCTYPE="*redshift*"
specifies that the data source is an Amazon Redshift database.

Valid in	CASLIB statement [Required]
Default	none

USENARROWCHARACTERTYPES=TRUE | FALSE
when set to TRUE, specifies to use narrow character types, such as CHAR or VARCHAR, for all character variables when writing data to the data source. When set to FALSE, this option specifies to use wide character types, such as NCHAR or NVARCHAR, for all character variables when writing data to the data source.

Because the CAS server uses UTF-8 encoding by default, this option defaults to FALSE.

Valid in	CASLIB statement
	PROC CASUTIL: SAVE statement
Default	FALSE

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	UID=
	USER=
Default	none
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Details

Supported Amazon Redshift Data Types

The following table shows the data types that can be loaded from Amazon Redshift into CAS. This table also shows the resulting data type for the data after it has been loaded into CAS. The length of the data format in CAS is based on the length of the source data.

Table 9.2 *Supported Amazon Redshift Data Types*

Amazon Redshift Data Type	CAS Data Type
Character Data Types	
CHAR(*n*)	VARCHAR
VARCHAR(*n*)	VARCHAR
Numeric Data Types	
BIGINT	INT64
BOOLEAN	DOUBLE
DECIMAL(*p*, *s*)	DOUBLE
DOUBLE PRECISION	DOUBLE
INTEGER	INT32
NUMERIC	DOUBLE
REAL	DOUBLE
SERIAL	DOUBLE
SMALLINT	INT32
Date and Time Data Types	
DATE	DOUBLE (formatted as DATE*w.*)
TIMESTAMP(*p*) (without time zone)	DOUBLE (formatted as DATETIME*w.d*)

About CHAR and VARCHAR Data Types

With Amazon Redshift, CHAR and VARCHAR data is defined in terms of bytes instead of characters. A CHAR variable can contain only single-byte characters. This means that a CHAR(10) variable can contain a string with a maximum length of 10 bytes. A VARCHAR variable can contain multiple-byte characters, up to a maximum of four bytes per character. For example, a VARCHAR(12) variable can contain 12 single-byte

characters, six two-byte characters, four three-byte characters, or three four-byte characters.

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Examples

Example 1: Add an Amazon Redshift Database as a Data Source For SAS Cloud Analytic Services

Use the CASLIB statement to establish a connection between your Amazon Redshift source data and a caslib, RScaslib. All of the options supplied in this example are required in the CASLIB statement, except the PASSWORD= option.

In this example, the Amazon Redshift data is stored at the location designated by the SERVER= and DATABASE= options.

```
caslib RScaslib desc='Amazon Redshift Caslib'
     dataSource=(srctype='redshift',
                 server='RSserver'
                 username='user1',
                 password='myPwd',
                 database='rsdatabase');
```

Example 2: Load Amazon Redshift Data into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="RScaslib";                    1
   load casdata="myRSdata" incaslib="RScaslib" outcaslib="casuser"
        casout="class_from_RScaslib";                 2
   list files incaslib="casuser";                     3
   contents casdata="%upcase(class_from_RScaslib)" incaslib="casuser";
                                                      4
quit;
```

1 List the tables in RScaslib before loading your data.

2 Load the table myRSdata from RScaslib into caslib Casuser. Call the new table class_from_RScaslib.

   Note: You must specify Amazon Redshift table names using the capitalization that is used in the database.

3 List the tables in Casuser to see the newly created table, class_from_RScaslib, that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

SAS Data Connector to DB2

SAS Data Connector to DB2 specifies the options to use when loading data from DB2 into SAS Cloud Analytic Services. Data connector options are used in the context of different statements that connect your data in DB2 with CAS.

Valid in: CASLIB statement
PROC CASUTIL statements (see options for details)

Examples: Establish a connection between your DB2 database and SAS Cloud Analytic Services.

```
caslib db2caslib desc='DB2 Caslib'
   datasource=(srctype='db2'
               username='myusr1'
               password='myPwd'
               database="sample");
```

Override USER= and PASSWORD= values.

```
proc casutil;
   load casdata="%upcase(mycas.db2examp)"
   dataSourceOptions=(username='user5' password='pwd5');
```

Syntax

Data Connector Options for DB2

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

AUTHENTICATIONDOMAIN="*domain*"
specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
Alias	AUTHDOMAIN=
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.

CATALOG="*catalog-name*"
specifies a logical catalog name. The logical name can be any user-defined name. This name is displayed in the Catalog column for all tables in the results from the CONTENTS statement in PROC CASUTIL.

Valid in	CASLIB statement
	PROC CASUTIL: CONTENTS, LOAD, and LIST statements
Default	Active caslib

CHARMULTIPLIER=*value*
specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Default	1.0
Range	1.0–5.0

CLIENT_ENCODING="*encoding*"
specifies the encoding that the data source client uses to transmit CHAR and VARCHAR data. Typically, this corresponds to the LANG environment variable.

If you have configured the DB2 client to use a different encoding than the LANG environment variable, set CLIENT_ENCODING= to the corresponding encoding value.

The CLIENT_ENCODING= value is independent of the NLS_LANG environment variable. For encodings other than UTF-8, the data is transcoded into UTF-8 when the data is loaded into CAS.

Valid in	CASLIB statement
Default	none

CONOPTS="*connection-options*"
specifies optional connection options that you pass to the DBMS.

Connection options that you specify in statements other than the CASLIB statement override any connection options that were set when a caslib was added.

Valid in	CASLIB statement
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Default	none
Example	`conopts="QueryTimeout=0"`

DB="*database-name*"
specifies the database name.

Valid in	CASLIB statement [Required]
Default	"" (empty string)
Range	1-4

DBCREATETABLEOPTS="*text*"
specifies any table options that should be appended to a CREATE TABLE statement that is passed to the data source.

Valid in	PROC CASUTIL: SAVE statement
Default	none
Restriction	The DBCREATETABLEOPTS= option applies to serial data transfer only.

INSERTBUFF="*number-of-rows*"
specifies the number of rows per block of data to save to a table in the data source.

Valid in	CASLIB statement
	PROC CASUTIL: SAVE statement
Default	Calculated automatically based on row size

NUMREADNODES=*number*
specifies the number of CAS worker nodes to use for a multinode load of data into CAS. Setting this option to a value other than 1 enables a data connector to use multiple nodes to load data into CAS. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in	PROC CASUTIL: LOAD statement
Default	1

NUMWRITENODES=*number*
specifies the number of CAS worker nodes to use for a multinode save of data from CAS into a data source table. Setting this option to a value other than 1 enables a data connector to use multiple nodes to save data to a data source table. You can specify any value up to the total number of available worker nodes. If the available

number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: SAVE statement

Default 1

PASSWORD="*password*"
specifies the DBMS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases PASS=

PWD=

Default none

Requirement If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

READBUFF="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default Calculated automatically based on row size

SCHEMA="*schema-name*"
specifies the schema name to use for the connection to the database or data source.

When you specify a value for SCHEMA=, the table name is qualified with the schema name. For example, if you set SCHEMA="mySchema" and you want to access table Studydata, the table mySchema.Studydata is accessed. Specify a non-blank value when you set a value for SCHEMA=.

If you specify a value for SCHEMA= in a statement other than the CASLIB statement, this value overrides any value of SCHEMA= that was set when the caslib was added.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Default none

SRCTYPE="*db2*"
specifies that the data source is a DB2 database.

Valid in	CASLIB statement [Required]
Default	none

USENARROWCHARACTERTYPES=TRUE | FALSE
when set to TRUE, specifies to use narrow character types, such as CHAR or VARCHAR, for all character variables when writing data to the data source. When set to FALSE, this option specifies to use wide character types, such as NCHAR or NVARCHAR, for all character variables when writing data to the data source.

Because the CAS server uses UTF-8 encoding by default, this option defaults to FALSE.

Valid in	CASLIB statement
	PROC CASUTIL: SAVE statement
Default	FALSE

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	UID=
	USER=
Default	none
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Details

Supported DB2 Data Types

The DB2 data connector supports the following data types.

Table 9.3 *Supported DB2 Data Types*

DB2 Data Type	CAS Data Type
Character	
CHAR(*n*)	CHAR

DB2 Data Type	CAS Data Type
CLOB	VARCHAR
GRAPHIC(*n*)	VARCHAR
LONG VARCHAR	VARCHAR
LONG VARGRAPHIC	VARCHAR
VARCHAR(*n*)	VARCHAR
Numeric	
BIGINT	INT64
DECIMAL	DOUBLE
DOUBLE	DOUBLE
INTEGER	INT32
SMALLINT	INT32
Date and Time	
DATE	DOUBLE (formatted as DATE*w.*)
TIME	DOUBLE (formatted as TIME*w.d*)
TIMESTAMP	DOUBLE (formatted as DATETIME*w.d*)

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is

converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Examples

Example 1: Add a DB2 Database as a Data Source For SAS Cloud Analytic Services

Use the CASLIB statement to establish a connection between your DB2 source data and a caslib, db2lib. All of the options supplied in this example are required in the CASLIB statement, except the PASSWORD= option.

In this example, DB2 data is stored in the location that the DB= option designates.

```
caslib db2lib
   datasource=(srctype='db2',
               username='myuser1',
               password='myPwd',
               db="myDB2data");
```

Example 2: Load DB2 Data into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list tables incaslib="casuser";                    1
   load casdata="MYDB2DATA" incaslib="db2caslib" outcaslib="casuser"
        casout="DB2data_from_db2caslib";              2
   list tables incaslib="casuser";                    3
   contents casdata="%upcase(class_from_db2caslib)" incaslib="casuser";
                                                      4
quit;
```

1 List the tables in Casuser before loading your data.

2 Load the table myDB2data from DB2 into caslib Casuser. Call the new table DB2data_from_db2caslib.

Note: You must list DB2 table names with capitalization that matches that in the DB2 database.

3 List the tables in Casuser again to see the newly created table, DB2data_from_db2caslib, that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop

SAS Data Connector to Hadoop lets you load data serially from Hive into SAS Cloud Analytic Services. All users can use SAS Data Connector to Hadoop. SAS Data Connect Accelerator for Hadoop is a separately licensed product that lets you load data in parallel using the SAS Embedded Process. In addition to loading data from Hive, SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop can also

load data that is described by a SASHDMD file from HDFS. Data connector options are used in the context of different statements that connect your data in Hadoop with CAS.

Valid in: CASLIB statement

PROC CASUTIL statements (see options for details)

Restriction: SAS Data Connect Accelerator for Hadoop does not support the ARRAY, MAP, STRUCT, UNION, BINARY, or BOOLEAN data types.

Requirements: A massively parallel processing (MPP) system is required for parallel loading of data using SAS Data Connect Accelerator for Hadoop.

JAVA_HOME must point to the location of the installed Java 8 JRE, and LIBJVM.SO must be specified in LD_LIBRARY_PATH. Modify this example to match the layout of your SAS Cloud Analytic Services cluster.

```
export JAVA_HOME=/usr/java/latest/jre
export LD_LIBRARY_PATH=$JAVA_HOME/lib/amd64/server:$LD_LIBRARY_PATH
```

Supports: Hadoop, HDMD, SPD Engine Files

See: Chapter 10, “Data Types,”

Examples: Load Hadoop source data and add a caslib to it.

```
caslib hvcaslib sessref=mysess
  dataSource=(srctype='hadoop',
              dataTransferMode='parallel',
              server='HiveServer',
              username='myuser1',
              hadoopJarPath='<Hadoop JAR path directory>',
              hadoopConfigDir='<Hadoop configuration directory>',
              schema='<Hive schema name>');
```

Load Hadoop source data using PROC CASUTIL.

```
proc casutil;
   incaslib='hvcaslib'
   sessref=mysess;
   load incaslib='hvcaslib' casdata='cars'
      casout='cars_CAS'
      where='cylinders=8'
      options=(dataTransferMode='parallel');
run;
quit;
```

Syntax

Data Connector Options for Hadoop

For each described option, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements and action calls, see “Where to Specify Data Connector Options” on page 170.

AUTHENTICATIONDOMAIN="*domain*"
specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement

other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
Alias	AUTHDOMAIN=
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.
Supports	HDMD, Hive, parallel data transfer, serial data transfer

BUFFERSIZE=*bytes*
specifies the buffer size length in bytes of the buffer that is used to send data to or receive data from SAS embedded processes. This value overrides a value of BUFFERSIZE= that is set in the CASLIB statement. Increasing the size might result in better performance with a trade-off of increased memory usage.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD and SAVE statements
Default	1048576
Restriction	This option applies only with SAS Data Connect Accelerator for Hadoop (DATATRANSFERMODE="parallel").
Supports	HDMD, Hive, parallel data transfer

CATALOG="*catalog-name*"
specifies a logical catalog name. The logical name can be any user-defined name. This name is displayed in the Catalog column for all tables in the results from the CONTENTS statement in PROC CASUTIL.

Valid in	CASLIB statement
Default	Active caslib

DATATRANSFERMODE="AUTO" | "PARALLEL" | "SERIAL"
specifies the mode of data transfer. If you specify this option outside of a CASLIB statement, this value overrides the value in a CASLIB statement.

Here are the valid values.

AUTO	specifies to first try to load the data in parallel using embedded processing. If this fails, a note-level message is issued and serial processing is then attempted.
PARALLEL	specifies to load the data in parallel by using the SAS Data Connect Accelerator to your database or data source.
SERIAL	specifies to load the data serially by using the SAS Data Connector to your database or data source.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD and SAVE statements
Aliases	DATATRANSFER=
	DTM=
Default	SERIAL
Requirement	To use the PARALLEL option, you must have a licensed copy of the SAS Data Connect Accelerator for your database or data source.
Interaction	If NUMREADNODES= or NUMWRITENODES= is a value other than 1 and DATATRANSFERMODE="AUTO", SAS attempts first to transfer the data in parallel using the data connect accelerator. If parallel transfer fails, SAS attempts to transfer data with the data connector using the specified number of CAS worker nodes. If multiple nodes are not available, SAS transfers the data serially.
Supports	HDMD, Hive, parallel data transfer, serial data transfer

DBCREATETABLEOPTS="*text*"
specifies any table options that should be appended to a CREATE TABLE statement that is passed to the data source.

specifies additional table options to use when creating a Hive table.

Valid in	PROC CASUTIL: SAVE statement
Default	none
Supports	Hive, parallel data transfer, serial data transfer

DBMAXTEXT=*maximum-string-column-length*
specifies the maximum length of a string by which all Hive string data types should be interpreted. For example, if `dbMaxText=100`, all Hive string data types are loaded as `VARCHAR(100)`.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Default	32767
Range	1 (minimum)
Supports	Hive, parallel data transfer, serial data transfer

HADOOPCONFIGDIR="*configuration-files-directory*"
specifies the Hadoop configuration files directory. This is the one that is obtained by running Ansible or the Hadoop extract script on the target Hadoop cluster to set the configuration path.

Valid in	CASLIB statement [required]
Default	none

Supports	HDMD, Hive, parallel data transfer, serial data transfer

HADOOPJARPATH="*Hadoop-jar-files-path*"
specifies one or more paths to the Hadoop JAR files. These are the JAR files that are obtained by running Ansible or the Hadoop extract script on the target Hadoop cluster to set the JAR path. These files are delimited by colons for Linux.

Valid in	CASLIB statement [required]
Default	none
Supports	HDMD, Hive, parallel data transfer, serial data transfer

HDFSDATADIR="*Hadoop-HDFS-data-directory*"
specifies the Hadoop HDFS directory to use to store table data when saving a CAS table to HDMD.

Valid in	CASLIB statement
	PROC CASUTIL: SAVE statement
Default	none
Supports	HDMD, parallel data transfer, serial data transfer

HDFSMETADIR="*Hadoop-HDFS-metadata-directory*"
specifies the Hadoop HDFS directory to use that contains one or more SASHDMD files.

Valid in	CASLIB statement
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Default	none
Supports	HDMD, parallel data transfer, serial data transfer

HDFSTEMPDIR="*Hadoop-HDFS-temporary-directory*"
specifies the Hadoop HDFS directory to use to store temporary data.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Default	"/TMP"
Supports	HDMD, Hive, parallel data transfer, serial data transfer

INSERTBUFF="*number-of-rows*"
specifies the number of rows per block of data to save to a table in the data source.

Valid in	CASLIB statement
	PROC CASUTIL: SAVE statement
Default	Calculated automatically based on row size
Supports	HDMD, Hive, serial data transfer

NAME="*hadoop*"
specifies Hive as the data source.

Default none

NUMREADNODES=*number*
specifies the number of CAS worker nodes to use for a multinode load of data into CAS. Setting this option to a value other than 1 enables a data connector to use multiple nodes to load data into CAS. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: LOAD statement

Default 1

Supports serial data transfer

NUMWRITENODES=*number*
specifies the number of CAS worker nodes to use for a multinode save of data from CAS into a data source table. Setting this option to a value other than 1 enables a data connector to use multiple nodes to save data to a data source table. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: SAVE statement

Default 1

Supports serial data transfer

PASSWORD="*password*"
specifies the DBMS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases PASS=

PWD=

Default none

Restriction When a Kerberos connection is used, no values should be specified for USERNAME= and PASSWORD=.

Requirement If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Supports	HDMD, Hive, parallel data transfer, serial data transfer

PATH="*DBMS-data-path*"
specifies the path and name of a DBMS source table. You can specify wildcards to match a specific set of table names.

Valid in	PROC CASUTIL: LOAD statement [Required]
Default	none

PORT=*port value*
specifies the Hive JDBC port number.

Valid in	CASLIB statement
Default	10000
Range	1–65535
Supports	Hive, parallel data transfer, serial data transfer

PROPERTIES="*Hive-JDBC-properties-value*"
specifies a free-form value for Hive JDBC properties. The value is appended to the JDBC connection URI. You can use it to override default Hive behaviors.

Valid in	CASLIB statement
Default	none
Supports	Hive, parallel data transfer, serial data transfer
Example	`properties=` `"hive.exec.dynamic.partition.mode=nonstrict;hive.exec.dynamic.partition=tru`

READBUFF="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Aliases	RAS=
	ROW_ARRAY_SIZE=
Default	Calculated automatically based on row size
Restriction	This option is not valid with the SAS Data Connect Accelerator (DATATRANSFERMODE="parallel"). Specify BUFFERSIZE= when loading data in parallel.
Supports	HDMD, Hive, serial data transfer

SCHEMA="*schema-name*"
specifies the schema name to use for the connection to the database or data source.

When you specify a value for SCHEMA=, the table name is qualified with the schema name. For example, if you set SCHEMA="mySchema" and you want to

access table Studydata, the table mySchema.Studydata is accessed. Specify a non-blank value when you set a value for SCHEMA=.

If you specify a value for SCHEMA= in a statement other than the CASLIB statement, this value overrides any value of SCHEMA= that was set when the caslib was added.

Valid in	CASLIB statement
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Default	'default'
Supports	Hive, parallel data transfer, serial data transfer

SERVER="*server-identifier*"
specifies the name of the node in the Hadoop cluster where the Hive service is running.

Valid in	CASLIB statement
	CASLIB statement
	CAS action: addCaslib
Default	none
Restriction	If your Hive server does not support high availability, you must specify SERVER= for a Hive caslib.
Supports	Hive, parallel data transfer, serial data transfer

SCRATCH_DB="*Hive-schema-name*
specifies the Hive database to use when creating temporary tables and views.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Default	none
Supports	Hive, parallel data transfer, serial data transfer

SRCTYPE="*hadoop*"
specifies that the data source is Hadoop (Hive or HDMD data source).

Valid in	CASLIB statement [Required]
Default	none
Supports	HDMD, Hive, parallel data transfer, serial data transfer

STATUSINTERVAL=*number*
specifies whether to print a message to the client when a node adds *n* buffers to the table by a node, where *n* is the value of this option. If you specify a value for this option outside of a CASLIB statement, it overrides the value in a CASLIB statement.

Valid in	CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default 0 (no message)

Restriction This option applies only to a SAS Data Connect Accelerator (DATATRANSFERMODE="PARALLEL").

Supports HDMD, Hive, parallel data transfer

URI="*Hive-JDBC-URI*"
specifies a free-form JDBC URI to use as the Hive JDBC connection URI. You can use this to override the default URI.

Valid in CASLIB statement

Default none

Interaction If you use this option, options that alter the JDBC URI (such as PROPERTIES=) are ignored.

Supports Hive, parallel data transfer, serial data transfer

USENARROWCHARACTERTYPES=TRUE | FALSE
when set to TRUE, specifies to use narrow character types, such as CHAR or VARCHAR, for all character variables when writing data to the data source. When set to FALSE, this option specifies to use wide character types, such as NCHAR or NVARCHAR, for all character variables when writing data to the data source.

Because the CAS server uses UTF-8 encoding by default, this option defaults to FALSE.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default FALSE

Supports Hive, serial data transfer

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases UID=

USER=

Default none

Restriction	When a Kerberos connection is used, no values should be specified for USERNAME= and PASSWORD=.
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.
Supports	HDMD, Hive, parallel data transfer, serial data transfer
Note	If you do not specify this option, the current user context is used. One scenario for leaving this option unspecified is to use the current Kerberos context.

Details

Use Options

Data connector options are used in the context of different statements that connect your data in Hadoop with SAS Cloud Analytic Services. For each described option, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

Hadoop Naming

The data connector and data connect accelerator can load Hive tables with names up to 128 characters or with column names that are up to 128 characters.

Authentication to a Hive Data Source

Depending on the user setup, credentials might be required to access the data in a Hive data source. Your system administrator defines an authentication domain that is associated with a pair of user ID and password values. The authentication domain and associated credentials are then available to you and other users who need to access data.

Typically, you supply credentials when you add a caslib. To do this, use the CASLIB statement and specify a value for either AUTHENTICATIONDOMAIN= or both USERNAME= and PASSWORD=. These credentials are then used for any statement that accesses the data using that caslib.

You can specify different credentials when you issue a statement that accesses the data. If you originally specified an authentication domain when you added a caslib, then you must specify values for both USERNAME= and PASSWORD= to override the original credentials. You can also override authentication domain credentials with a different AUTHENTICATIONDOMAIN= value.

While not typical, it is possible to supply USERNAME= and PASSWORD= values in separate statements. For example, you might supply a USERNAME= value in the CASLIB statement and then supply a PASSWORD= value when you specify a LOAD statement that accesses the data. In this situation, you must supply the PASSWORD= value for each statement that accesses the data.

Load Data in Parallel

If SAS Data Connect Accelerator for Hadoop is installed on your system and SAS Embedded Process is installed on your Hadoop cluster, you can use DATATRANSFERMODE="PARALLEL" to load data in parallel.

Load a Connection to a Hadoop Data Source

When you load a connection to a data source, you specify the type of data source to which SAS Cloud Analytic Services is connecting. However, a connection is not made until you load data from the data source. To load the connection to the data source and to add a caslib that accesses that data source, use the CASLIB statement.

Work with Metadata Files

A SASHDMD file describes how to convert lines in a Hadoop file into a table of rows and columns. The SASHDMD file must contain the path to the Hadoop file that it describes. There are several ways to create a SASHDMD file.

- The Hadoop LIBNAME engine creates a SASHDMD file when a new table is created in Hadoop.
- PROC HDMD creates a SASHDMD file for an existing Hadoop file.
- The Save action creates a SASHDMD file when a CAS table is saved to an HDMD caslib.

A SASHDMD file is required when you use an HDMD caslib. The SASHDMD file should be located in the HDFSMETADIR= and be named <*table*>.sashdmd.

For more information about creating metadata files, see *SAS In-Database Products: User's Guide*.

Define a Hive or HDMD Caslib

When you define a Hadoop caslib, you must specify whether to use it to access Hive tables or HDMD tables. Using the HDFSMETADIR= option determines whether the caslib is a Hive caslib or an HDMD caslib.

Once you define the caslib as either Hive or HDMD, you cannot change the caslib type at the action level. Data source options that are specific to Hive that are on an HDMD caslib are ignored, as are data source parameters that are specific to HDMD that are on a Hive caslib.

Table 9.4 *Options for Specifying a Hive or HDMD Caslib*

HDFSMETADIR= Option	SERVER= Option	URI= Option	Caslib Type
specified	specified	specified	HDMD
		not specified	
	not specified	specified	
		not specified	
not specified	specified	specified	Hive, using URI=
		not specified	Hive, using the configuration that is found in HadoopConfigPath
	not specified	specified	
		not specified	Hive server must be high availability

Supported Hive and HDMD Data Types

The Hadoop data connector supports these data types for loading Hive and HDMD data into SAS Cloud Analytic Services.

Table 9.5 *Supported Hive and HDMD Data Types*

Hive Data Type	HDMD Data Type	CAS Data Type
Character Data Types		
CHAR	CHAR	CHAR
VARCHAR	VARCHAR	VARCHAR
STRING		VARCHAR
Numeric Data Types		
ARRAY		ERROR (not supported)
BIGINT	BIGINT	INT64
BINARY	BINARY	ERROR (not supported)
BOOLEAN		ERROR (not supported)
DECIMAL	DECIMAL	DOUBLE
DOUBLE	DOUBLE	DOUBLE
FLOAT		DOUBLE
	INT	INT32
INTEGER		INT32
INTERVAL		DOUBLE
MAP		ERROR (not supported)
	REAL	DOUBLE
SMALLINT	SMALLINT	INT32
STRUCT		ERROR (not supported)
TINYINT	TINYINT	INT32
UNIONTYPE		ERROR (not supported)
Date and Time Data Types		

Hive Data Type	HDMD Data Type	CAS Data Type
DATE	DATE	DOUBLE
TIME	TIME	DOUBLE
TIMESTAMP		DOUBLE

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Examples

Example 1: Specify Hive as a Data Source for a Caslib in SAS Cloud Analytic Services

Use the CASLIB statement to initialize the data source and add the caslib for Hadoop. No connection is made to the data source until a statement that accesses the data is called. The data is read in parallel into the caslib hvcaslib.

```
caslib hvcaslib desc='hadoop caslib'
   dataSource=(srctype='hadoop',
               dataTransferMode='parallel',
               server='hiveServer',
               username='myuser1',
               password='myPwd',
               hadoopJarPath='<Hadoop_jar_path_directory>',
               hadoopConfigDir='<Hadoop_configuration_directory>',
               schema='<Hive_schema_name>');
```

Example 2: Specify HDMD as a Data Source for a Caslib in SAS Cloud Analytic Services

Use the CASLIB statement to initialize the data source and add the caslib for Hadoop. No connection is made to the data source until a statement that accesses the data is called. The data is read in serially into the caslib hvcaslib.

```
caslib hdmdlib desc='hdmd caslib'
   dataSource=(srctype='hadoop',
               dataTransferMode='serial',
               username='myuser1',
               hadoopJarPath='<Hadoop JAR path directory>',
               hadoopConfigDir='<Hadoop configuration directory>',
               hadoopDataDir='<HDFS data directory>',
               hadoopMetaDir='<HDFS SASHDMD directory>');
```

Example 3: Load a Hive Table into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="Hadoopcaslib";           1
   load casdata="myHDdata" incaslib="Hadoopcaslib" outcaslib="casuser"
        casout="HDdata_from_Hadoopcaslib";       2
   list tables incaslib="casuser";               3
   contents casdata="HDdata_from_Hadoopcaslib" incaslib="casuser";
                                                 4
quit;
```

1 List the tables in Hadoopcaslib before loading your data.

2 Load the table myHDdata from Hive into the caslib, casuser. Call the new table HDdata_from_Hadoopcaslib.

3 List the tables in caslib casuser to see the newly created HDdata_from_Hadoopcaslib table that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

Example 4: Load an HDMD Table into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="HDMDcaslib";           1
   load casdata="myHDdata" incaslib="HDMDcaslib" outcaslib="casuser2"
        casout="HDdata_from_HDMDcaslib";       2
   list tables incaslib="casuser2";            3
   contents casdata="HDdata_from_HDMDcaslib" incaslib="casuser2";
                                               4
quit;
```

1 List the tables in HDMDcaslib before loading your data.

2 Load the table myHDdata from Hive into the caslib, casuser2 Call the new table HDdata_from_HDMDcaslib.

3 List the tables in caslib casuser2to see the newly created HDdata_from_HDMDcaslib table that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

SAS Data Connector to Impala

SAS Data Connector to Impala lets you load data serially from Impala into SAS Cloud Analytic Services. All users can use SAS Data Connector to Impala.

Valid in: CASLIB statement
PROC CASUTIL statements (see parameters for details)

See: Chapter 10, "Data Types,"

Examples: Load Impala source data and add a caslib to it.

```
caslib imcaslib sessref=mysess
  dataSource=(srctype='impala',
              server='impala01.example.com',
              username='myuser1'
              password='myPwd'
              database='mydb';
```

Load Impala source data using PROC CASUTIL.

```
proc casutil;
   incaslib='imcaslib'
   sessref=mysess;
   load incaslib='imcaslib' casdata='cars'
      casout='cars_CAS'
      where='cylinders=8';
run;
quit;
```

Syntax

Data Connector Options for Impala

For each described option, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

AUTHENTICATIONDOMAIN="*domain*"
specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
Alias	AUTHDOMAIN=

Requirement If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.

CATALOG="*catalog-name*"
specifies a logical catalog name. The logical name can be any user-defined name. This name is displayed in the Catalog column for all tables in the results from the CONTENTS statement in PROC CASUTIL.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LOAD, and LIST statements

Default Active caslib

CHARMULTIPLIER=*value*
specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default 1.0

Range 1.0–5.0

CLIENT_ENCODING="*encoding*"
specifies the encoding that the data source client uses to transmit CHAR and VARCHAR data. Typically, this corresponds to the LANG environment variable.

If you have configured the underlying ODBC driver to use a different encoding than the LANG environment variable, set CLIENT_ENCODING= to the corresponding encoding value.

The CLIENT_ENCODING= value is independent of the NLS_LANG environment variable. For encodings other than UTF-8, the data is transcoded into UTF-8 when the data is loaded into CAS.

Valid in CASLIB statement

Default UTF-8

DBCREATETABLEOPTS="*text*"
specifies any table options that should be appended to a CREATE TABLE statement that is passed to the data source.

Valid in PROC CASUTIL: SAVE statement

Default none

Restriction The DBCREATETABLEOPTS= option applies to serial data transfer only.

DM_UNICODE="*Unicode-setting*"
specifies the Unicode encoding for the driver manager. Possible values include UTF-8, UCS-2, and so on. This setting applies to Linux platforms when using third-party ODBC driver managers, such as unixODBC.

Valid in CASLIB statement

PROC CASUTIL: LOAD statements

Default UCS-2/UTF-16

DRIVER_VENDOR="DATADIRECT" | "MAPR"
specifies the name of the specific third-party vendor that supports the Impala driver.

Valid in CASLIB statement

Default none (the ODBC driver from Cloudera is used)

IMPALA_DSN="*Impala-datasource-name*"
specifies the data source name.

With this option, you can use an Impala data source that you have previously configured in an odbc.ini file.

Valid in CASLIB statement [Required]

Default none

Requirement Set the ODBCSYSINI or ODBCINI environment variable. If the ODBCSYSINI environment variable is set, it must point to the full path of the directory that contains the configured odbc.ini and odbcinst.ini files. If ODBCSYSINI is not set, set the ODBCINI environment variable to the full path of the directory that contains the configured odbc.ini file.

INSERTBUFF="*number-of-rows*"
specifies the number of rows per block of data to save to a table in the data source.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default Calculated automatically based on row size

NAME="*impala*"
specifies Impala as the data source.

Default none

NUMREADNODES=*number*
specifies the number of CAS worker nodes to use for a multinode load of data into CAS. Setting this option to a value other than 1 enables a data connector to use multiple nodes to load data into CAS. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: LOAD statement

Default 1

NUMWRITENODES=*number*
specifies the number of CAS worker nodes to use for a multinode save of data from CAS into a data source table. Setting this option to a value other than 1 enables a data connector to use multiple nodes to save data to a data source table. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: SAVE statement

Default 1

PASSWORD="*password*"
specifies the DBMS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases PASS=

PWD=

Default none

Requirement If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

PORT="*port-value*"
specifies the port to use to connect to the Impala database.

Valid in CASLIB statement

Default 21050

Range 1–65535

READBUFF="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Aliases RAS=

ROW_ARRAY_SIZE=

Default Calculated automatically based on row size

SCHEMA="*schema-name*"
specifies the schema name to use for the connection to the database or data source.

When you specify a value for SCHEMA=, the table name is qualified with the schema name. For example, if you set SCHEMA="mySchema" and you want to access table Studydata, the table mySchema.Studydata is accessed. Specify a non-blank value when you set a value for SCHEMA=.

If you specify a value for SCHEMA= in a statement other than the CASLIB statement, this value overrides any value of SCHEMA= that was set when the caslib was added.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Default none

SERVER="*server-identifier*"
specifies the server identifier for the DBMS server.

Valid in CASLIB statement [Required]

Default none

SRCTYPE="*impala*"
specifies that the data source is an Impala database.

Valid in CASLIB statement [Required]

Default none

USENARROWCHARACTERTYPES=TRUE | FALSE
when set to TRUE, specifies to use narrow character types, such as CHAR or VARCHAR, for all character variables when writing data to the data source. When set to FALSE, this option specifies to use wide character types, such as NCHAR or NVARCHAR, for all character variables when writing data to the data source.

Because the CAS server uses UTF-8 encoding by default, this option defaults to FALSE.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default FALSE

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	UID=
	USER=
Default	none
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Details

Use Options

Data connector options are used in the context of different statements that connect your data in Impala with SAS Cloud Analytic Services. For each described option, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

Impala Naming

The data connector can load Impala tables with names up to 128 characters or with column names that are up to 128 characters.

Authentication to an Impala Database

Credentials are required to access the data in an Impala database. Your system administrator defines an authentication domain that is associated with a pair of user ID and password values. The authentication domain and associated credentials are then available to you and other users who need to access data.

Typically, you supply credentials when you add a caslib. To do this, use the CASLIB statement and specify a value for either AUTHENTICATIONDOMAIN= or both USERNAME= and PASSWORD=. These credentials are then used for any statement that accesses the data using that caslib.

You can specify different credentials when you issue a statement that accesses the data. If you originally specified an authentication domain when you defined a caslib, then you must specify values for both USERNAME= and PASSWORD= to override the original credentials. You can also override authentication domain credentials with a different AUTHENTICATIONDOMAIN= value.

Although it is not typical, it is possible to supply USERNAME= and PASSWORD= in separate statements. For example, you might supply a USERNAME= value in the CASLIB statement and then supply a PASSWORD= value when you specify a LOAD statement that accesses the data. In this situation, you must supply the PASSWORD= value for each statement that accesses the data.

Load a Connection to an Impala Data Source

When you load a connection to a data source, you specify the type of data source to which SAS Cloud Analytic Services is connecting. However, a connection is not made until you load data from the data source. To load the connection to the data source and to define a caslib that accesses that data source, use the CASLIB statement.

Supported Impala Data Types

This table shows the data types that can be loaded from Impala into CAS. This table also shows the resulting data type for the data after it has been loaded into CAS. The length of the data format in CAS is based on the length of the source data.

Table 9.6 *Supported Impala Data Types*

Impala Data Type	CAS Data Type
Character Data Types	
CHAR	CHAR
VARCHAR	VARCHAR
STRING	VARCHAR
Numeric Data Types	
BIGINT	INT64
BOOLEAN	DOUBLE
DECIMAL(*p*, *s*)	DOUBLE
DOUBLE	DOUBLE
FLOAT	DOUBLE
INTEGER	INT32
SMALLINT	INT32
TINYINT	INT32
Date and Time Data Types	
TIMESTAMP(*p*) (without time zone)	DOUBLE (formatted as DATETIME*w.d*)

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action

supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Examples

Example 1: Specify an Impala Database as a Data Source for a Caslib in SAS Cloud Analytic Services

```
caslib imcaslib desc='Impala Caslib'
   dataSource=(srctype='Impala',
               server='impalaServer',
               username='myuser1',
               password='myPwd',)
               database='mydb';
```

Example 2: Load an Impala Table into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="Impalacaslib";                    1
   load casdata="myIMdata" incaslib="Impalacaslib" outcaslib="casuser"
        casout="IMdata_from_Impalacaslib";                2
   list tables incaslib="casuser";                        3
   contents casdata="IMdata_from_Impalacaslib" incaslib="casuser";
                                                          4
quit;
```

1 List the tables in Impalacaslib before loading your data.

2 Load the table myIMdata from Impala into caslib Casuser. Call the new table IMdata_from_Impalacaslib.

3 List the tables in caslib to see the newly created IMdata_from_Impalacaslib table that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

SAS Data Connector to Microsoft SQL Server

SAS Data Connector to Microsoft SQL Server enables you to load data from a Microsoft SQL Server data source into SAS Cloud Analytic Services. Data connector options are used in the context of different statements that connect your data in Microsoft SQL Server with CAS.

Valid in: CASLIB statement
PROC CASUTIL statements (see options for details)

Examples: Establish a connection between your Microsoft SQL Server database and SAS Cloud Analytic Services.

```
caslib sqlservercaslib desc='Microsoft SQL Server Caslib'
     dataSource=(srctype='sqlserver'
                 username='user1'
                 password='myPwd'
                 sqlserver_dsn="configured-DSN-name"
                 catalog='*');
```

Overriding the user and password values.

```
proc casutil;
   load casdata="mycas.sqlsvrexamp" casout="mySQLsvrdata" casuser dataSourceOptions=(
         username='user5'
         password='myPwd');
quit;
```

Syntax

Data Connector Options for Microsoft SQL Server

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

AUTHENTICATIONDOMAIN="*domain*"

specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
Alias	AUTHDOMAIN=
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.

CATALOG="*"

CATALOG="*" specifies that Microsoft SQL Server supports multiple native databases. This option is required for SQL Server connections.

The native Microsoft SQL Server database name is displayed in the Catalog column for all tables in the results from the CONTENTS statement in PROC CASUTIL.

Valid in	CASLIB statement [Required]
	PROC CASUTIL: CONTENTS, LOAD, and LIST statements
Default	Active caslib

CHARMULTIPLIER=*value*
specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default 1.0

Range 1.0–5.0

CLIENT_ENCODING="*encoding*"
specifies the encoding that the data source client uses to transmit CHAR and VARCHAR data. Typically, this corresponds to the LANG environment variable.

If you have configured the underlying ODBC driver to use a different encoding than the LANG environment variable, set CLIENT_ENCODING= to the corresponding encoding value.

The CLIENT_ENCODING= value is independent of the NLS_LANG environment variable. For encodings other than UTF-8, the data is transcoded into UTF-8 when the data is loaded into CAS.

Valid in CASLIB statement

CONOPTS="*connection-options*"
specifies optional connection options that you pass to the DBMS.

Connection options that you specify in statements other than the CASLIB statement override any connection options that were set when a caslib was added.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Default none

Example `conopts="QueryTimeout=0"`

DBCREATETABLEOPTS="*text*"
specifies any table options that should be appended to a CREATE TABLE statement that is passed to the data source.

Valid in PROC CASUTIL: SAVE statement

Default none

Restriction The DBCREATETABLEOPTS= option applies to serial data transfer only.

INSERTBUFF="*number-of-rows*"
specifies the number of rows per block of data to save to a table in the data source.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default Calculated automatically based on row size

NUMREADNODES=*number*
specifies the number of CAS worker nodes to use for a multinode load of data into CAS. Setting this option to a value other than 1 enables a data connector to use multiple nodes to load data into CAS. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: LOAD statement

Default 1

NUMWRITENODES=*number*
specifies the number of CAS worker nodes to use for a multinode save of data from CAS into a data source table. Setting this option to a value other than 1 enables a data connector to use multiple nodes to save data to a data source table. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: SAVE statement

Default 1

PASSWORD="*password*"
specifies the DBMS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases PASS=

PWD=

Default none

Requirement If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

READBUFF="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default Calculated automatically based on row size

SCHEMA="*schema-name*"
specifies the schema name to use for the connection to the database or data source.

When you specify a value for SCHEMA=, the table name is qualified with the schema name. For example, if you set SCHEMA="mySchema" and you want to access table Studydata, the table mySchema.Studydata is accessed. Specify a non-blank value when you set a value for SCHEMA=.

If you specify a value for SCHEMA= in a statement other than the CASLIB statement, this value overrides any value of SCHEMA= that was set when the caslib was added.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Default none

SRCTYPE="*sqlserver*"
specifies that the data source is a Microsoft SQL Server database.

Valid in CASLIB statement [Required]

Default none

USENARROWCHARACTERTYPES=TRUE | FALSE
when set to TRUE, specifies to use narrow character types, such as CHAR or VARCHAR, for all character variables when writing data to the data source. When set to FALSE, this option specifies to use wide character types, such as NCHAR or NVARCHAR, for all character variables when writing data to the data source.

Because the CAS server uses UTF-8 encoding by default, this option defaults to FALSE.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default FALSE

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases UID=

USER=

Default	none
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Details

Supported Microsoft SQL Server Data Types

The following table shows the data types that can be loaded from Microsoft SQL Server into CAS. This table also shows the resulting data type for the data after it has been loaded into CAS. The length of the data format in CAS is based on the length of the source data.

Table 9.7 *Supported Microsoft SQL Server Data Types*

Microsoft SQL Server Data Type	CAS Data Type
BIT(*n*)	VARCHAR
CHARACTER(*n*)	VARCHAR
NCHAR	CHAR
NVARCHAR	VARCHAR
TEXT	VARCHAR
BIGINT	INT64
INTEGER	INT32
SMALLINT	INT32
TINYINT	INT32
DOUBLE PRECISION	DOUBLE
DECIMAL(*p*, *s*)	DOUBLE
FLOAT	DOUBLE
NUMERIC	DOUBLE
REAL	DOUBLE
MONEY	DOUBLE
SMALLMONEY	DOUBLE

Microsoft SQL Server Data Type	CAS Data Type
DATE	DOUBLE (formatted at DATE*w.*)
TIME(*p*) (without time zone)	DOUBLE (formatted as TIME*w.*)
DATETIME(*p*) (without time zone)	DOUBLE (formatted as DATETIME*w.*)
DATETIME2(*p*)	DOUBLE (formatted as DATETIME*w.*)

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Examples

Example 1: Add a Microsoft SQL Server Database as a Data Source For SAS Cloud Analytic Services

Use the CASLIB statement to establish a connection between your Microsoft SQL Server source data and a caslib, SQLservercaslib. All of the options supplied in this example are required in the CASLIB statement, except the PASSWORD= option.

In this example, the Microsoft SQL Server data is stored at the location designated by the SQLSERVER_DSN= option.

```
caslib SQLservercaslib desc='Microsoft SQL Server Caslib'
     dataSource=(srctype='sqlserver',
                 username='user1',
                 password='myPwd',
                 sqlserver_dsn='SQLserverDSN');
```

Example 2: Load Microsoft SQL Server Data into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="SQLservercaslib";                    1
   load casdata="mySQLsvrdata" incaslib="SQLservercaslib" outcaslib="casuser"
        casout="class_from_SQLservercaslib";                 2
   list files incaslib="casuser";                            3
   contents casdata="class_from_SQLservercaslib" incaslib="casuser";
                                                             4
quit;
```

1 List the tables in SQLservercaslib before loading your data.

2 Load the table mySQLsvrdata from SQLservercaslib into caslib Casuser. Call the new table class_from_SQLservercaslib.

3 List the tables in Casuser to see the newly created table, class_from_SQLservercaslib, that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

SAS Data Connector to ODBC

SAS Data Connector to ODBC specifies the options to use when loading data from an ODBC data source into SAS Cloud Analytic Services. Data connector options are used in the context of different statements that connect your data in ODBC with CAS.

Valid in: CASLIB statement
PROC CASUTIL: LOAD statement

Example: Establish a connection to ODBC data. (Note: The GLOBAL option is restricted to administrators.)

```
caslib odbccaslib desc="SQLviaODBCtoCaslib"
     datasource=(srctype="odbc"
                 username="user1"
                 password="password1"
                 odbc_dsn="dbodbc") global;
```

Syntax

Data Connector Options for ODBC

For each option that is described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see “Where to Specify Data Connector Options” on page 170.

AUTHENTICATIONDOMAIN="*domain*"
specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement

other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
Alias	AUTHDOMAIN=
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.

CATALOG="*catalog-name*"
specifies a logical catalog name. The logical name can be any user-defined name. This name is displayed in the Catalog column for all tables in the results from the CONTENTS statement in PROC CASUTIL.

For data sources that natively support multiple catalogs, such as Microsoft SQL Server or Netezza, specify CATALOG="*". Specifying CATALOG="*" enables all catalogs to be referenced.

Valid in	CASLIB statement
	PROC CASUTIL: CONTENTS, LOAD, and LIST statements
Default	Active caslib

CHARMULTIPLIER=*value*
specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Default	1.0
Range	1.0–5.0

CLIENT_ENCODING="*encoding*"
specifies the encoding that the data source client uses to transmit CHAR and VARCHAR data. Typically, this corresponds to the LANG environment variable.

If you have configured the ODBC driver to use a different encoding than the LANG environment variable, set CLIENT_ENCODING= to the corresponding encoding value.

The CLIENT_ENCODING= value is independent of the NLS_LANG environment variable. For encodings other than UTF-8, the data is transcoded into UTF-8 when the data is loaded into CAS.

Valid in	CASLIB statement

Default none

CONOPTS="*connection-options*"
specifies optional connection options that you pass to the underlying DBMS.

Connection options that you specify in statements other than the CASLIB statement override any connection options that were set when a caslib was added.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Default none

Example `conopts="QueryTimeout=0"`

DBCREATETABLEOPTS="*text*"
specifies any table options that should be appended to a CREATE TABLE statement that is passed to the data source.

Valid in PROC CASUTIL: SAVE statement

Default none

Restriction The DBCREATETABLEOPTS= option applies to serial data transfer only.

DM_UNICODE="*unicode-setting*"
specifies the Unicode encoding that the driver manager uses. Possible values include UTF-8, UCS-2, and so on.

This setting applies to Linux platforms when using third-party ODBC driver managers, such as unixODBC.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default UTF-8

INSERTBUFF="*number-of-rows*"
specifies the number of rows per block of data to save to a table in the data source.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default Calculated automatically based on row size

ODBC_DSN="*DSN-name*"
specifies the data source name for your DBMS.

Valid in CASLIB statement [Required]

Aliases DATABASE=

DB=

PASSWORD="*password*"
specifies the DBMS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	PASS=
	PWD=
Default	none
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

READBUFF="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Default	Calculated automatically based on row size

SCHEMA="*schema-name*"
specifies the schema name to use for the connection to the database or data source.

When you specify a value for SCHEMA=, the table name is qualified with the schema name. For example, if you set SCHEMA="mySchema" and you want to access table Studydata, the table mySchema.Studydata is accessed. Specify a non-blank value when you set a value for SCHEMA=.

If you specify a value for SCHEMA= in a statement other than the CASLIB statement, this value overrides any value of SCHEMA= that was set when the caslib was added.

Valid in	CASLIB statement
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Default	none

SRCTYPE="*odbc*"
specifies that the data source is accessed through ODBC.

Valid in	CASLIB statement [Required]
Default	none

USENARROWCHARACTERTYPES=TRUE | FALSE
when set to TRUE, specifies to use narrow character types, such as CHAR or VARCHAR, for all character variables when writing data to the data source. When set to FALSE, this option specifies to use wide character types, such as NCHAR or NVARCHAR, for all character variables when writing data to the data source.

Because the CAS server uses UTF-8 encoding by default, this option defaults to FALSE.

Valid in	CASLIB statement
	PROC CASUTIL: SAVE statement
Default	FALSE

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	UID=
	USER=
Default	none
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Details

Supported ODBC Data Types

The following table lists the supported data types that the ODBC data connector can load from ODBC into SAS Cloud Analytic Services. This table also shows the resulting data type in CAS.

Note: Be aware that when performing calculations on numeric values and when storing numeric values, SAS maintains up to 15 digits of precision. When you read values that contain more than 15 decimal digits of precision from a database into SAS, the values that are read are rounded to meet this condition. For noncomputational purposes, such as storing ID values or credit card numbers, you can read the data in as character data.

Table 9.8 *Data Type Conversions When Loading Data from ODBC into CAS*

ODBC Data Type	CAS Data Type
Character	
SQL_BINARY	CHAR
SQL_CHAR	CHAR
SQL_LONGVARBINARY	VARCHAR
SQL_LONGVARCHAR	VARCHAR
SQL_VARBINARY	VARCHAR
SQL_VARCHAR	VARCHAR
SQL_WCHAR	CHAR
SQL_WLONGVARCHAR	VARCHAR
SQL_WVARCHAR	VARCHAR
Numeric	
SQL_BIGINT	BIGINT (INT64)
SQL_BIT	DOUBLE
SQL_DECIMAL	DOUBLE
SQL_DOUBLE	DOUBLE
SQL_FLOAT	DOUBLE
SQL_INTEGER	INTEGER (INT32)
SQL_NUMERIC	DOUBLE
SQL_REAL	DOUBLE
SQL_SMALLINT	INTEGER (INT32)
SQL_TINYINT	INTEGER (INT32)
Date and Time	
SQL_TYPE_DATE	DOUBLE (formatted as DATE)
SQL_TYPE_TIME	DOUBLE (formatted as TIME)

ODBC Data Type	CAS Data Type
SQL_TYPE_TIMESTAMP	DOUBLE (formatted as DATETIME)

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Examples

Example 1: Establish a Connection between a Microsoft SQL Database and a Caslib through ODBC

Use the CASLIB statement to add a Microsoft SQL database as a data source for SAS Cloud Analytic Services using an ODBC connection. For connections to Microsoft SQL, specify CATALOG="*".

In this example, GLOBAL specifies that the data source is potentially available to all sessions. You might not have access to the GLOBAL option if you are not an administrator. For more information, see "CASLIB Statement" on page 91.

```
caslib odbccaslib desc="SQLviaODBCtoCaslib"
     datasource=(srctype="odbc"
                      username="user1"
                      password="password1"
                      odbc_dsn="dbodbc"
                      catalog="*") global;
```

Example 2: PROC CASUTIL: Load Data from an External Database Into SAS Cloud Analytic Services

```
proc casutil;
  list files incaslib="odbccaslib";       /* 1 */
  load casdata="myDBdata" incaslib="odbccaslib" outcaslib="casuser"
       casout="class_from_odbccaslib";    /* 2 */
  list tables incaslib="casuser";         /* 3 */
  contents casdata="class_from_odbccaslib" incaslib="casuser";
```

```
                                                    /* 4 */
   quit;
```

1 List the files in odbccaslib before loading your data.

2 Load the table myDBdata from an external database into memory in caslib Casuser. Call the new table class_from_oraclecaslib.

3 List the tables in casuser to see the newly created table, class_from_odbccaslib, that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

SAS Data Connector to Oracle

SAS Data Connector to Oracle specifies the options to use when loading data from Oracle into SAS Cloud Analytic Services. Data connector options are used in the context of different statements that connect your data in Oracle with CAS.

Valid in: CASLIB statement

PROC CASUTIL statements (see options for details)

Examples: Establish a connection between your Oracle database and SAS Cloud Analytic Services.

```
caslib orcaslib desc='Oracle Caslib'
   datasource=(srctype='oracle'
               username='myusr1'
               password='myPwd'
               path="//machine.lnx.com:1521/exadat");
```

Override USER= and PASSWORD= values.

```
proc casutil;
   load casdata="%upcase(mycas.orexamp)"
   dataSourceOptions=(username='user5' password='pwd5');
```

Syntax

Data Connector Options for Oracle

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

AUTHENTICATIONDOMAIN="*domain*"
specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements

Alias AUTHDOMAIN=

Requirement If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.

CHARMULTIPLIER=*value*
specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default 1.0

Range 1.0–5.0

DBCREATETABLEOPTS="*text*"
specifies any table options that should be appended to a CREATE TABLE statement that is passed to the data source.

Valid in PROC CASUTIL: SAVE statement

Default none

Restriction The DBCREATETABLEOPTS= option applies to serial data transfer only.

INSERTBUFF="*number-of-rows*"
specifies the number of rows per block of data to save to a table in the data source.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default Calculated automatically based on row size

NUMREADNODES=*number*
specifies the number of CAS worker nodes to use for a multinode load of data into CAS. Setting this option to a value other than 1 enables a data connector to use multiple nodes to load data into CAS. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: LOAD statement

Default 1

NUMWRITENODES=*number*
specifies the number of CAS worker nodes to use for a multinode save of data from CAS into a data source table. Setting this option to a value other than 1 enables a data connector to use multiple nodes to save data to a data source table. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: SAVE statement

Default 1

ORA_ENCODING=" <*encoding-name*> "
specifies the encoding of the data in the Oracle database. This value is independent of the NLS_LANG environment variable setting. The data is transcoded from the database encoding into UTF-8 when data is loaded into CAS.

Valid values include LATIN1, WLATIN1, and UNICODE. Set this value to UNICODE when you are loading non-Latin1 data.

TIP If the source data is encoded in UTF-8, then data does not need to be transcoded into UTF-8 when loading it into CAS. This reduces the time it takes to load the data.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default UNICODE

PASSWORD="*password*"
specifies the DBMS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases PASS=

PWD=

Default none

Requirement If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

PATH="*DBMS-data-path*"
specifies the Oracle driver, node, and database. Aliases are required if you are using SQL*Net Version 2.0 or higher.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, and LOAD statements [Required for LOAD]

Default none

Example
```
caslib oraclecaslib desc='Oracle Caslib'
   datasource=(srcType='oracle'
               username='myuser1'
               password='myPwd'
               path="//machine.lnx.com:1521/exadat");
```

READBUFF="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default Calculated automatically based on row size

SCHEMA="*schema-name*"
specifies the schema name to use for the connection to the database or data source.

When you specify a value for SCHEMA=, the table name is qualified with the schema name. For example, if you set SCHEMA="mySchema" and you want to access table Studydata, the table mySchema.Studydata is accessed. Specify a non-blank value when you set a value for SCHEMA=.

If you specify a value for SCHEMA= in a statement other than the CASLIB statement, this value overrides any value of SCHEMA= that was set when the caslib was added.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Default none

SRCTYPE="*oracle*"
specifies that the data source is an Oracle database.

Valid in CASLIB statement [Required]

Default none

USENARROWCHARACTERTYPES=TRUE | FALSE
when set to TRUE, specifies to use narrow character types, such as CHAR or VARCHAR, for all character variables when writing data to the data source. When set to FALSE, this option specifies to use wide character types, such as NCHAR or NVARCHAR, for all character variables when writing data to the data source.

Because the CAS server uses UTF-8 encoding by default, this option defaults to FALSE.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default FALSE

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	UID=
	USER=
Default	none
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Details

Supported Oracle Data Types

This table shows the data types that can be loaded from Oracle into CAS. This table also shows the resulting data type for the data after it has been loaded into CAS. The length of the data format in CAS is based on the length of the source data.

Table 9.9 *Supported Oracle Data Types*

Oracle Data Type	CAS Data Type
CHAR	CHAR
VARCHAR	VARCHAR
Numeric Data Types	
ARRAY	ERROR (not supported)
BOOLEAN	ERROR (not supported)
DOUBLE	DOUBLE
large object types (BLOB, CLOB)	ERROR (not supported)
MAP	ERROR (not supported)
STRUCT	ERROR (not supported)

Oracle Data Type	CAS Data Type
UNION	ERROR (not supported)

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Case Sensitivity with Oracle

All quoted values are passed to the Oracle database exactly as you enter them. This means that you must specify all values, such as table names or ID values and using the same capitalization as the Oracle database uses.

Examples

Example 1: Add an Oracle Database as a Data Source For SAS Cloud Analytic Services

Use the CASLIB statement to establish a connection between your Oracle source data and a caslib, Oralib. All of the options supplied in this example are required in the CASLIB statement, except the PASSWORD= option.

In this example, Oracle data is stored in the location that the PATH= option designates.

```
caslib oralib
   datasource=(srctype='oracle',
               username='myuser1',
               password='myPwd',
               path="myORAdata");
```

Example 2: Load Oracle Data into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list tables incaslib="casuser";          1
   load casdata="MYORADATA" incaslib="orcaslib" outcaslib="casuser"
```

```
      casout="ORAdata_from_orcaslib";        2
   list tables incaslib="casuser";           3
   contents casdata="%upcase(class_from_orcaslib)" incaslib="casuser";
                                             4
quit;
```

1 List the tables in casuser before loading your data.

2 Load the table myORAdata from Oracle into caslib Casuser. Call the new table ORAdata_from_orcaslib.

Note: You must list Oracle table names with capitalization that matches that in the Oracle database.

3 List the tables in casuser again to see the newly created table, ORAdata_from_orcaslib, that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

SAS Data Connector to PC Files

Specifies the options to use when loading data from files of various file types into SAS Cloud Analytic Services.

Valid in: CASLIB statement

Applies to: CAS procedure

See: Chapter 10, "Data Types,"

Example: Load the Titanic table into SAS Cloud Analytic Services. The data was obtained from http://biostat.mc.vanderbilt.edu/DataSets.

```
options validvarname=any;
proc casutil;
   load file="/path/to/titanic3.xls" casout="titanic3"
      importOptions=(filetype="xls" getnames=true);
quit;
```

Syntax

Data Connector Options for PC Files

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

DBCREATETABLEOPTS="*text*"
specifies any table options that should be appended to a CREATE TABLE statement that is passed to the data source.

Valid in	PROC CASUTIL: SAVE statement
Default	none

Restriction The DBCREATETABLEOPTS= option applies to serial data transfer only.

FILETYPE="DTA" | "EXCEL" | "JMP" | "SPSS" | "XLS"
specifies the file type. Specify FILETYPE="EXCEL" to work with XLSX files (.xlsx filename suffix).

Valid in CASLIB statement

Default XLS

Restriction Use XLSX for national language characters. These characters do not display correctly for XLS files.

Requirement You must specify this option for any non-XLS file type.

GETNAMES="TRUE" | "FALSE"
uses the values in the first line of the file as variable names when GETNAMES=TRUE.

Valid in CASLIB statement

Default TRUE

PATH="*table-path-and-name*"
specifies the location and the name of a PC Files source table.

Valid in CASLIB statement [required]

Default none

Requirement Although no data source options are required when you work with Microsoft Excel files, you must use a caslib with a data source type of PATH.

RANGE="*cell-range*"
specifies the range of data (a subset of cells) within the worksheet to import.

Valid in CASLIB statement

Default none (imports the entire worksheet)

Restriction For XLSX files (FILETYPE="EXCEL"), do not include the worksheet name in the range.

Requirement To use this option, you must specify a range that represents a rectangle in the worksheet.

Examples A17–D51

For XLS files, the range "Sheet1$A1:B5" is the range address for a rectangular block of 10 cells, where the top left cell is A1 and the bottom right cell is B5.

SHEET="*worksheet-name*"
specifies the name of the worksheet within the Excel file to import. If you plan to import a range of cells from an XLS file, do not specify the SHEET= option. Instead, specify the worksheet as part of the RANGE= option.

Valid in	CASLIB statement
Default	none (imports the entire worksheet)

Details

SAS Studio Tip

If you can navigate to the file in the **Server Files and Folders** section, you can use the CASUTIL procedure with the LOAD FILE= syntax to load data into SAS Cloud Analytic Services.

For large files, if SAS Cloud Analytic Services can access the file, you can use the CASUTIL procedure with the LOAD CASDATA= syntax.

Specifying a Range of Cells

The specification for the RANGE= option is "*sheet name*$*top left cell*:*bottom right cell*".

- If you omit the worksheet name, the first worksheet in the workbook is used. However, you must include the dollar sign ($) character if you want to specify a range of cells on the first worksheet. For example, RANGE="$A1:E1" specifies to read five cells from the first row of the first worksheet.
- If you omit the top left cell value, A1 is used as the first cell.
- If you omit the bottom right cell value, the last row and column in the worksheet is used.

Supported PC Files Data Types

The PC Files data connector supports these data types.

Table 9.10 *Supported PC Files Data Types*

PC Files Data Type	CAS Data Type
DTA	VARCHAR
JMP	VARCHAR
SPSS	VARCHAR
XLS	VARCHAR
XLSX	VARCHAR

Example: Use RANGE= with the CASUTIL Procedure to Specify Data to Import

This example shows how to use the RANGE= option to specify a block of cells to import. The file is available from http://catalog.data.gov/dataset/2010-federal-stem-education-inventory-data-set. One characteristic of the file is that the first two rows are used to provide column descriptions and the columns exceed SAS naming rules. The RANGE= option skips the first two rows. GETNAMES="false" specifies to read data in Row 3 as data values rather than column names.

```
proc casutil;
   load file="2010 Federal STEM Education Inventory Data Set.xls"
      casout="stem2010"
      importoptions=(filetype="xls" getnames="false" range="Sheet1$A3:IV254");

  /* Be sure that the data type is listed as DOUBLE
   * for Columns G, H, and I.                        */
   contents casdata="stem2010";
run;

libname mycas cas;
proc print data=mycas.stem2010(caslib="casuser" obs=2);
   var a -- j;
run;
```

Disclaimer: SAS might reference other websites, content, or resources for use at the Customer's sole discretion. SAS has no control over any websites or resources that companies or persons other than SAS provide. The Customer acknowledges and agrees that SAS is not responsible for the availability or use of any such external sites or resources, and does not endorse any advertising, products, or other materials on or available from such websites or resources. The Customer acknowledges and agrees that SAS is not liable for any loss or damage that the Customer or its end users might incur as a result of the availability or use of those external sites or resources, or as a result of any reliance that the Customer or its end users places on the completeness, accuracy, or existence of any advertising, products, or other materials on or available from such websites or resources.

SAS Data Connector to PostgreSQL

SAS Data Connector to PostgreSQL enables you to load data from PostgreSQL into SAS Cloud Analytic Services. Data connector options are used in the context of different statements that connect your data in PostgreSQL with CAS.

Valid in: CASLIB statement
PROC CASUTIL statements (see options for details)

Examples: Establish a connection between your PostgreSQL database and SAS Cloud Analytic Services.

```
caslib postgrescaslib desc='PostgreSQL Caslib'
     dataSource=(srctype='postgres'
                 server='PGserver'
                 username='user1'
                 password='myPwd'
                 database="PGdatabase-name"
                 schema='myschema');
```

Overriding the user and password values.

```
proc casutil;
   load casdata="mycas.pgexamp" casout="myPGdata" casuser dataSourceOptions=(
         username='user5'
         password='myPwd');
quit;
```

Syntax

Data Connector Options for PostgreSQL

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

AUTHENTICATIONDOMAIN="*domain*"

specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
Alias	AUTHDOMAIN=
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.

CATALOG="*catalog-name*"

specifies a logical catalog name. The logical name can be any user-defined name. This name is displayed in the Catalog column for all tables in the results from the CONTENTS statement in PROC CASUTIL.

Valid in	CASLIB statement
	PROC CASUTIL: CONTENTS, LOAD, and LIST statements
Default	Active caslib

CHARMULTIPLIER=*value*

specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Default	1.0
Range	1.0–5.0

CLIENT_ENCODING="*encoding*"
specifies the encoding that the data source client uses to transmit CHAR and VARCHAR data. Typically, this corresponds to the LANG environment variable.

If you have configured the underlying ODBC driver to use a different encoding than the LANG environment variable, set CLIENT_ENCODING= to the corresponding encoding value.

The CLIENT_ENCODING= value is independent of the NLS_LANG environment variable. For encodings other than UTF-8, the data is transcoded into UTF-8 when the data is loaded into CAS.

Valid in CASLIB statement

CONOPTS="*connection-options*"
specifies optional connection options that you pass to the DBMS.

Connection options that you specify in statements other than the CASLIB statement override any connection options that were set when a caslib was added.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Default none

Example
```
conopts="QueryTimeout=0"
```

DATABASE="*database-name*"
specifies the name of the database.

When you specify a value for DATABASE=, the table name is qualified with the database name. For example, if you set DATABASE="myDB" and you want to access table Studydata, the table myDB.Studydata is accessed.

Valid in CASLIB statement [Required]

PROC CASUTIL: LOAD statement

Default none

Interaction If you supply values for SCHEMA= and DATABASE=, the value for SCHEMA= is used to qualify a table name.

DBCREATETABLEOPTS="*text*"
specifies any table options that should be appended to a CREATE TABLE statement that is passed to the data source.

Valid in PROC CASUTIL: SAVE statement

Default none

Restriction The DBCREATETABLEOPTS= option applies to serial data transfer only.

INSERTBUFF="*number-of-rows*"
specifies the number of rows per block of data to save to a table in the data source.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default Calculated automatically based on row size

NUMREADNODES=*number*
specifies the number of CAS worker nodes to use for a multinode load of data into CAS. Setting this option to a value other than 1 enables a data connector to use multiple nodes to load data into CAS. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: LOAD statement

Default 1

NUMWRITENODES=*number*
specifies the number of CAS worker nodes to use for a multinode save of data from CAS into a data source table. Setting this option to a value other than 1 enables a data connector to use multiple nodes to save data to a data source table. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: SAVE statement

Default 1

PASSWORD="*password*"
specifies the DBMS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases PASS=

PWD=

Default none

Requirement If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

PTG_DSN="*PostgreSQL-datasource-name*"
specifies the data source name.

With this option, you can use a PostgreSQL data source that you have previously configured in an odbc.ini file.

Valid in CASLIB statement [Required]

Default none

Requirement Set the ODBCSYSINI or ODBCINI environment variable. If the ODBCSYSINI environment variable is set, it must point to the full path of the directory that contains the configured odbc.ini and odbcinst.ini files. If ODBCSYSINI is not set, set the ODBCINI environment variable to the full path of the directory that contains the configured odbc.ini file.

READBUFF="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default Calculated automatically based on row size

SCHEMA="*schema-name*"
specifies the schema name to use for the connection to the database or data source.

When you specify a value for SCHEMA=, the table name is qualified with the schema name. For example, if you set SCHEMA="mySchema" and you want to access table Studydata, the table mySchema.Studydata is accessed. Specify a non-blank value when you set a value for SCHEMA=.

If you specify a value for SCHEMA= in a statement other than the CASLIB statement, this value overrides any value of SCHEMA= that was set when the caslib was added.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Default none

Interaction If you specify values for SCHEMA= and DATABASE=, the value for SCHEMA= is used to qualify the table name.

SERVER="*server-identifier*"
specifies the server identifier for the DBMS server.

Valid in CASLIB statement [Required]

Default none

SRCTYPE="*postgres*"
specifies that the data source is a PostgreSQL database.

Valid in CASLIB statement [Required]

Default none

USENARROWCHARACTERTYPES=TRUE | FALSE
when set to TRUE, specifies to use narrow character types, such as CHAR or VARCHAR, for all character variables when writing data to the data source. When set to FALSE, this option specifies to use wide character types, such as NCHAR or NVARCHAR, for all character variables when writing data to the data source.

Because the CAS server uses UTF-8 encoding by default, this option defaults to FALSE.

Valid in	CASLIB statement
	PROC CASUTIL: SAVE statement
Default	FALSE

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	UID=
	USER=
Default	none
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Details

Case Sensitivity with PostgreSQL

All quoted values are passed to the PostgreSQL database exactly as you type them. This means that you must specify all values, such as table names or ID values, using the same capitalization that is used in the PostgreSQL database.

Supported PostgreSQL Data Types

The following table shows the data types that can be loaded from PostgreSQL into CAS. This table also shows the resulting data type for the data after it has been loaded into CAS. The length of the data format in CAS is based on the length of the source data.

Table 9.11 *Supported PostgreSQL Data Types*

PostgreSQL Data Type	CAS Data Type
BIT(*n*)	VARCHAR
CHARACTER(*n*)	VARCHAR
TEXT	VARCHAR

PostgreSQL Data Type	CAS Data Type
INTEGER (INT32)	INTEGER (INT32)
BIGINT (INT64)	BIGINT (INT64)
SMALLINT	INTEGER (INT32)
DOUBLE PRECISION	DOUBLE
DECIMAL(*p*, *s*)	DOUBLE
NUMERIC	DOUBLE
REAL	DOUBLE
SERIAL	DOUBLE
BIGSERIAL	DOUBLE
DATE	DOUBLE (formatted as DATE*w.*)
TIME(*p*) (without time zone)	DOUBLE (formatted as TIME*w.d*)
TIMESTAMP(*p*) (without time zone)	DOUBLE (formatted as DATETIME*w.d*)
JSON, JSONB	VARCHAR

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is

recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Examples

Example 1: Add a PostgreSQL Database as a Data Source For SAS Cloud Analytic Services

Use the CASLIB statement to establish a connection between your PostgreSQL source data and a caslib, PostgreSQLcaslib. All of the options supplied in this example are required in the CASLIB statement, except the PASSWORD= option.

In this example, the PostgreSQL data is stored at the location designated by the SERVER= and DATABASE= options.

```
caslib PostgreSQLcaslib desc='PostgreSQL Caslib'
     dataSource=(srctype='postgres',
                 server='PGserver'
                 username='user1',
                 password='myPwd',
                 database='PGdatabase',
                 schema='myschema');
```

Example 2: Load PostgreSQL Data into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="PostgreSQLcaslib";                    1
   load casdata="myPGdata" incaslib="PostgreSQLcaslib" outcaslib="casuser"
        casout="class_from_PostgreSQLcaslib";                 2
   list files incaslib="casuser";                             3
   contents casdata="%upcase(class_from_PostgreSQLcaslib)" incaslib="casuser";
                                                              4
quit;
```

1 List the tables in PostgreSQLcaslib before loading your data.

2 Load the table myPGdata from PostgreSQLcaslib into caslib Casuser. Call the new table class_from_PostgreSQLcaslib.

Note: You must specify PostgreSQL table names using the capitalization that is used in the database.

3 List the tables in Casuser to see the newly created table, class_from_PostgreSQLcaslib, that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

SAS Data Connector to SAP HANA

SAS Data Connector to SAP HANA lets you load data serially from SAP HANA into SAS Cloud Analytic Services. All users can use SAS Data Connector to SAP HANA.

Valid in: CASLIB statement
PROC CASUTIL statements (see options for details)

Examples: Load an SAP HANA data source and add a caslib to it.

```
caslib hacaslib sessref=mysess
   datasource=(srctype='hana',
               server='hana server',
               username='user1',
               password='myPwd',
               tableType='column'
               schema='hana schema'
               instance='00');
```

Load SAP HANA source data using PROC CASUTIL.

```
proc casutil;
   incaslib='hacaslib'
   sessref=mysess;
   list files;
   run;
quit;
```

Syntax

Data Connector Parameters for SAP HANA

For each parameter described, the applicable statements where you can use that parameter are indicated. For information about where to specify these parameters within statements and action calls, see "Where to Specify Data Connector Options" on page 170.

AUTHENTICATIONDOMAIN="*domain*"
specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
Alias	AUTHDOMAIN=
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.

CHARMULTIPLIER=*value*
specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Default	1.0
Range	1.0–5.0

DBCREATETABLEOPTS="*text*"

Valid in	PROC CASUTIL: SAVE statement
Default	none

INSERTBUFF="*number-of-rows*"
specifies the number of rows per block of data to save to a table in the data source.

Valid in	CASLIB statement
	PROC CASUTIL: SAVE statement
Default	Calculated automatically based on row size

INSTANCE="*number*"
specifies the SAP HANA port instance.

Valid in	CASLIB statement
Default	00
Restriction	Specify a value for only INSTANCE= or PORT=, not both. (As an alternative to PORT=, you can instead specify SERVER="*server-identifier*:*port*.) If you specify a value for both, a connection error results.

NAME="*hana*"
specifies the SAP HANA data source.

Valid in	PROC CASUTIL: LOAD statement
Default	hana

NUMREADNODES=*number*
specifies the number of CAS worker nodes to use for a multinode load of data into CAS. Setting this option to a value other than 1 enables a data connector to use multiple nodes to load data into CAS. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in	PROC CASUTIL: LOAD statement
Default	1

NUMWRITENODES=*number*
specifies the number of CAS worker nodes to use for a multinode save of data from CAS into a data source table. Setting this option to a value other than 1 enables a data connector to use multiple nodes to save data to a data source table. You can

specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in PROC CASUTIL: SAVE statement

Default 1

PASSWORD="*password*"
specifies the DBMS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases PASS=

PWD=

Default none

Requirement If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

PORT=*port value*
specifies the SAP HANA port number.

Valid in CASLIB statement

Default 30015

Restriction Specify a value for only INSTANCE= or PORT=, not both. (As an alternative to PORT=, you can instead specify SERVER="*server-identifier:port*.) If you specify a value for both, a connection error results.

READBUFF="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default Calculated automatically based on row size

SCHEMA="*schema-name*"
specifies the schema name to use for the connection to the database or data source.

When you specify a value for SCHEMA=, the table name is qualified with the schema name. For example, if you set SCHEMA="mySchema" and you want to access table Studydata, the table mySchema.Studydata is accessed. Specify a non-blank value when you set a value for SCHEMA=.

If you specify a value for SCHEMA= in a statement other than the CASLIB statement, this value overrides any value of SCHEMA= that was set when the caslib was added.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Default none

SERVER="*server-identifier*"
SERVER="*server-identifier:port*"
specifies the server identifier for the DBMS server.

Valid in CASLIB statement [Required]

Default none

SRCTYPE="*hana*"
specifies that the data source is an SAP HANA database.

Valid in CASLIB statement [Required]

Default none

STATUSINTERVAL=*number*
specifies whether to print a message to the client when a node adds *n* buffers to the table by a node, where *n* is the value of this option. If you specify a value for this option outside of a CASLIB statement, it overrides the value in a CASLIB statement.

Valid in CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default 0 (no message)

Restriction This option applies only to a SAS Data Connect Accelerator (DATATRANSFERMODE="PARALLEL").

TABLETYPE="ROW | COLUMN"
specifies the format type of the SAP HANA table to return: ROW for a row-type table or COLUMN for a column-type table.

Default 1

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Aliases	UID=
	USER=
Default	none
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Details

Supported SAP HANA Data Types

This table shows the data types that can be loaded from SAP HANA into CAS. This table also shows the resulting data type for the data after it has been loaded into CAS. The length of the data format in CAS is based on the length of the source data.

Table 9.12 *Supported SAP HANA Data Types*

SAP HANA Data Type	CAS Data Type
CHAR	CHAR
VARCHAR	VARCHAR
DOUBLE	DOUBLE

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Example: Specify SAP HANA as a Data Source for a Caslib in SAS Cloud Analytic Services

Use the CASLIB statement to initialize the data source and add the caslib for SAP HANA. No connection is made to the data source until a statement that accesses the data is called. The data is read in parallel into the caslib hacaslib.

```
caslib hacaslib sessref=mysess desc="hana caslib"
   dataSource=(srctype="hana",
               server="hanaServer",
               username="myuser1",
               password="myPwd",
               tableType="column",
               schema="<Hana_schema_name>",
               instance="00");
```

SAS Data Connector to SPD Engine Files and SAS Data Connect Accelerator for SPD Engine Files

SAS Data Connector to SPD Engine Files lets you load data serially into SAS Cloud Analytic Services from SPD Engine files that reside on a file system, including HDFS. All users can use SAS Data Connector to SPD Engine Files. SAS Data Connect Accelerator for SPD Engine Files requires a license to SAS In-Database Technologies for Hadoop on SAS Viya. Using SAS Data Connect Accelerator for SPD Engine Files, you can load SPD Engine file data in parallel using the SAS Embedded Process. Data connector options are used in the context of different statements that connect your data in SPD Engine files with CAS.

Valid in: CASLIB statement
PROC CASUTIL statements (see parameters for details)

Restriction: When you use parallel data transfer mode, files cannot be encrypted or compressed.

Examples: Establish a connection between your SPD Engine Files data source and SAS Cloud Analytic Services using serial mode.

```
caslib spdecaslib desc='SPDE Caslib'
   dataSource=(srctype='spde',
               dataTransferMode="serial",
               mdfpath="/user/MyMDFPath");
```

Establish a connection between your SPD Engine Files data source and SAS Cloud Analytic Services using serial mode.

```
caslib spdecaslib desc='SPDE Caslib'
   dataSource=(srctype='spde',
               dataTransferMode="serial",
               hadoopjarpath="/hadoop/jars",
               hadoopconfigdir="/hadoop/conf",
               mdfpath="/user/spdemdfs");
```

Load an SPD Engine Files data source using PROC CASUTIL.

```
proc casutil;
   incaslib="spdecaslib" sessref=mysess;
   load incaslib="spdecaslib" casdata="cars"
      casout="cars_CAS"
      options=(dataTransferMode="serial");
run;
```

```
quit;
```

Syntax

Data Connector Parameters for SPD Engine Files

For each parameter described, the applicable statements where you can use that parameter are indicated. For information about where to specify these parameters within statements and action calls, see "Where to Specify Data Connector Options" on page 170.

ALTERPW="*password*"
specifies the password to access and modify or delete a password-protected SPD Engine table.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD and SAVE statements
Default	none
Supports	parallel data transfer (SAVE statement), serial data transfer (LOAD and SAVE statements)

AUTHENTICATIONDOMAIN="*domain*"
specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
Alias	AUTHDOMAIN=
Restriction	The AUTHENTICATIONDOMAIN= option is only valid when SPD Engine files are stored in Hadoop.
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.

BUFFERSIZE=*bytes*
specifies the buffer size length in bytes of the buffer that is used to send data to or receive data from SAS embedded processes. This value overrides a value of BUFFERSIZE= that is set in the CASLIB statement. Increasing the size might result in better performance with a trade-off of increased memory usage.

Valid in	CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default 1048576

Supports parallel data transfer

COMPRESS="YES | NO | BINARY | CHAR"
specifies the compression method to use for the SPD Engine table.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default NO

Supports serial data transfer

DATAPATH="*DPF-file-path*"
specifies the path where to place the DPF files for the SPD Engine table.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default MDFPATH=

DATATRANSFERMODE="AUTO" | "PARALLEL" | "SERIAL"
specifies the mode of data transfer.

Here are the valid values.

AUTO — specifies to first try to load the data in parallel using embedded processing. If this fails, a note-level message is issued and serial processing is then attempted.

PARALLEL — specifies to load the data in parallel by using the SAS Data Connect Accelerator to your database or data source.

SERIAL — specifies to load the data serially by using the SAS Data Connector to your database or data source.

Valid in CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Aliases DATATRANSFER=

DTM=

Default SERIAL

Requirement To use the PARALLEL option, you must have a licensed copy of the SAS Data Connect Accelerator for Hadoop.

Interaction If NUMREADNODES= or NUMWRITENODES= is a value other than 1 and DATATRANSFERMODE="AUTO", SAS attempts first to transfer the data in parallel using the data connect accelerator. If parallel transfer fails, SAS attempts to transfer data with the data

connector using the specified number of CAS worker nodes. If multiple nodes are not available, SAS transfers the data serially.

Supports parallel data transfer, serial data transfer

ENCRYPT="YES" | "NO" | "AES"
specifies the form of encryption to use to access an SPD Engine table.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LOAD, and SAVE statements

Default NO

Restriction This option applies only with SAS Data Connector to SPD Engine Files (DATATRANSFERMODE="SERIAL").

Supports serial data transfer

ENCRYPTIONPASSWORD="*passphrase*"
specifies a passphrase to use for an SPD Engine table that is encrypted using the Advanced Encryption Standard (AES).

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LOAD, and SAVE statements

Default none

Restriction You can specify this option only with the DATASOURCE= option.

Requirement This option is required only if the SPD Engine table uses AES encryption. When you create a table, the encryption key must be in uppercase letters to ensure that the table can be read.

Supports serial data transfer

HADOOPCONFIGDIR="*configuration-files-directory*"
specifies the Hadoop configuration files directory. This directory can be obtained by running the Hadoop tracer tool on the target Hadoop cluster.

Valid in CASLIB statement

Default none

Supports parallel data transfer, serial data transfer

HADOOPJARPATH="*jar-files-path*"
specifies one or more paths to the Hadoop JAR files. These files can be obtained by running the Hadoop tracer tool on the target Hadoop cluster. The files are delimited by colons for Linux.

Valid in CASLIB statement

Default none

Requirement delimited by a colon

Supports parallel data transfer, serial data transfer

HDFS=TRUE | FALSE
specifies where SPD Engine file data is stored.

TRUE
specifies that SPD Engine file data is stored in HDFS.

FALSE
specifies that SPD Engine file data is stored on a traditional disk system.

Restriction This option applies only when DATATRANSFERMODE="SERIAL", which defaults to HDFS=FALSE. HDFS=TRUE is the default when DATATRANSFERMODE="PARALLEL".

Supports parallel data transfer, seriial data transfer

HDFSTEMPDIR="*temporary-directory*"
specifies the Hadoop HDFS directory to use to store temporary data.

Valid in CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default "/tmp"

Supports parallel data transfer

LABEL="*label-name*"
specifies a label name for the created SPD Engine table.

Default none

Supports serial data transfer

MASTERPW="*password*"
specifies the password to access and delete, modify, read, or save a password-protected SPD Engine table.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, LOAD, and SAVE statements

Default none

Supports parallel data transfer (SAVE statement), serial data transfer (LOAD and SAVE statements)

MDFPATH="*SPDE-MDF-file-directory*"
specifies the HDFS or local disk directory that contains the SPD Engine MDF file.

Valid in CASLIB statement [required]

PROC CASUTIL: CONTENTS, LIST, and SAVE statements

Default none

Supports parallel data transfer, serial data transfer

NUMREADNODES=*number*
specifies the number of CAS worker nodes to use for a multinode load of data into CAS. Setting this option to a value other than 1 enables a data connector to use multiple nodes to load data into CAS. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in	PROC CASUTIL: LOAD statement
Default	0
Supports	serial data transfer

PADCOMPRESS=*number*
specifies the amount of padding to add when you use the COMPRESS= option.

Valid in	CASLIB statement
	PROC CASUTIL: SAVE statement
Default	0
Supports	serial data transfer

PARTSIZE=*number*
specifies the size of partitions to use for the SPD Engine table when it is created using the SAVE table action.

Valid in	CASLIB statement
	PROC CASUTIL: SAVE statement
Default	SPD Engine file default
Supports	serial data transfer

PASSWORD="*password*"
specifies the HDFS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	PASS=
	PWD=
Default	none
Restriction	This option is valid only when SPD Engine files are stored in Hadoop (HDFS=TRUE).

Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.
Supports	parallel data transfer, serial data transfer

READPW="*password*"
specifies the password to access and read a password-protected SPD Engine table.

Valid in	CASLIB statement
Default	none
Supports	parallel data transfer (SAVE statement), serial data transfer (LOAD and SAVE statements)

SRCTYPE="*spde*"
specifies that the data source is an SPD Engine file.

Valid in	CASLIB statement [Required]
Default	none
Supports	parallel data transfer, serial data transfer

STATUSINTERVAL=*number*
specifies whether to print a message to the client when a node adds *n* buffers to the table by a node, where *n* is the value of this option. If you specify a value for this option outside of a CASLIB statement, it overrides the value in a CASLIB statement.

Valid in	CASLIB statement PROC CASUTIL: LOAD and SAVE statements
Default	0 (no message)
Restriction	This option applies only to a SAS Data Connect Accelerator (DATATRANSFERMODE="PARALLEL").
Supports	parallel data transfer

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in	CASLIB statement [see Requirement] PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	UID=

	USER=
Default	none
Restriction	This option is valid only when SPD Engine files are stored in Hadoop (HDFS=TRUE).
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

WRITEPW="*password*"
specifies the password to access and save a password-protected SPD Engine table.

Valid in	CASLIB statement
Default	none
Supports	serial data transfer

Details

Load an SPD Engine Table

You can load an SPD Engine table into SAS Cloud Analytic Services from any Hadoop cluster to which you have access—SAS Cloud Analytic Services does not have to be co-located with Hadoop. You can also run SAS Cloud Analytic Services on a subset of the Hadoop machines and still perform the load. Network performance can limit load times.

Load a Connection to an SPD Engine Files Data Source

When you load a connection to a data source, you specify the type of data source to which SAS Cloud Analytic Services is connecting. However, a connection is not made until you load data from the data source.

You can load a connection to a data source in two ways. To load only the connection, call the loadDataSource action. To load the connection to the data source and to define a caslib that accesses that data source, use the CASLIB statement. In addition, if you use the CASLIB statement, you do not need to call the loadDataSource action.

Supported SPD Engine Files Data Types

This table shows the data types that can be loaded from SPD Engine files into CAS. This table also shows the resulting data type for the data after it has been loaded into CAS. The length of the data format in CAS is based on the length of the source data. SAS formats are supported when loading and saving SPD Engine files data sets.

Table 9.13 *Supported SPD Engine Files Data Types*

SPD Engine Files Data Type	CAS Data Type
CHAR	CHAR
DOUBLE	DOUBLE

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Examples

Example 1: Specify SPD Engine Files as a Data Source for a Caslib in SAS Cloud Analytic Services

Use the CASLIB statement to initialize the data source and add the caslib for SPD Engine file data that is stored in Hadoop. No connection is made to the data source until a statement that accesses the data is called. Data is read serially into the caslib SPDEcaslib. SPD Engine file credentials are required to access the data. You can specify these in the CASLIB statement or when you use statements that access the data.

```
caslib spdecaslib sessref=mysess desc="SPDE caslib"
   datasource=(srctype="spde",
               username="user1",
               password="myPwd",
               mdfpath="hdfs_path_to_your_mdf_files",
               hadoopJarPath="<Hadoop_jar_path_directory>",
               hadoopConfigDir="<Hadoop_configuration_directory>",
               dtm="serial",
               hdfs="true"
);
```

Example 2: Load SPD Engine Files Data into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="SPDEcaslib";          1
   load casdata="mySPDEdata" incaslib="SPDEcaslib" outcaslib="casuser"
        casout="SPDEdata_from_SPDEcaslib";      2
   list tables incaslib="casuser";              3
   contents casdata="SPDEdata_from_SPDEcaslib" incaslib="casuser";
                                                4
quit;
```

1. List the tables in SPD Engine files before loading your data.
2. Load the table mySPDEdata from SPD Engine files into caslib Casuser. Call the new table SPDEdata_from_SPDEcaslib.
3. List the tables in caslib to see the newly created SPDEdata_from_SPDEcaslib table that you loaded.
4. List information about the newly loaded table, including column names, data types, and so on.

SAS Data Connector to SAS Data Sets

Specifies the settings to use when loading data from SAS data sets into SAS Cloud Analytic Services.

Applies to: CAS procedure, CASUTIL procedure

Restriction: Although this data connector supports cross-environment data access (CEDA), it does not support DELETE operations. Therefore, if CEDA processing is invoked, you must use the DATASETS procedure to delete a SAS data set.

Requirement: Table names must be specified only in lowercase letters.

Tip: Except for DELETE operations (see Restriction), this data connector supports cross-environment data access (CEDA). CEDA enables a data set that is created on one computer architecture to be read by CAS on the same or different computer architecture. For more information, see "Definition of Cross-Environment Data Access (CEDA)" in *SAS Language Reference: Concepts*.

See: Chapter 10, "Data Types,"

Example: Load a password-protected SAS data set into SAS Cloud Analytic Services.

```
proc casutil;
   load casdata="salary.sas7bdat" casout="salary"
      importoptions=(filetype="basesas" password="myPwd");
run;
```

Syntax

Data Connector Options for SAS Data Sets

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

ALTER="*password*"
specifies the password to access and delete a password-protected data set.

Default none

Requirement This option is required only if the data set is password-protected.

CHARMULTIPLIER=*value*
specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates. For double-byte character sets, set CHARMULTIPLIER=2.0.

Valid in	PROC CASUTIL: LOAD statement
Default	1.0
Range	1.0–5.0

COMPRESS="YES" | "NO" | "BINARY" | "CHAR"
specifies the compression method to use for the data set when you save an in-memory table as a SAS data set.

Valid in	PROC CASUTIL: SAVE statement
Default	NO
Interaction	Use this option with the REUSE= option.

DATATRANSFERMODE="AUTO" | "PARALLEL" | "SERIAL"
specifies the mode of data transfer.

Here are the valid values.

AUTO	specifies to first try to load the data in parallel. If this fails, a note-level message is issued and serial processing is then attempted.
PARALLEL	specifies to load the data in parallel. The controller directs each worker to read a portion of the file. This method requires that all workers have access to the data set.
SERIAL	specifies to load the data serially by using the SAS Data Connector to SAS Data Sets.

Valid in	importOptions option on the loadTable action
	PROC CASUTIL: LOAD and SAVE statements
Aliases	DATATRANSFER=
	DTM=
Default	AUTO
Restriction	Data is loaded serially if only the controller has access to the SAS data set.
Requirement	To load data serially, the controller must have access to the SAS data set. To load data in parallel, the controller and all worker nodes must have access to the SAS data set.
Supports	parallel data transfer, serial data transfer

ENCODING="*encoding*"
specifies the encoding for the data set.

Valid in	PROC CASUTIL: SAVE statement
Default	UTF-8

ENCRYPT="YES" | "NO" | "AES" | "AES2"
specifies the form of encryption to use to access the table.

Valid in	PROC CASUTIL: SAVE statement
Default	NO

ENCRYPTIONPASSWORD="*passphrase*"
specifies a passphrase to use for a data set that is encrypted using the Advanced Encryption Standard (AES).

Valid in	PROC CASUTIL: DELETESOURCE, LOAD, SAVE statement
Default	none
Requirement	This option is required only if the SAS data set uses AES encryption. When you create a table, the encryption key must be in uppercase letters to ensure that the table can be read.

FILETYPE="BASESAS"
specifies the file type.

Valid in	PROC CASUTIL: DELETESOURCE, LOAD, SAVE statement
Default	This is based on the file extension that is specified in the loadTable path option.
Requirement	This option is required for SAS7BDAT files only if you do not include the SAS7BDAT extension in the file name.

PASSWORD="*password*"
specifies the password for a password-protected SAS data set.

Valid in	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Default	none
Restriction	Use this option only if the SAS data set is password-protected or uses SAS proprietary encryption.

PATH="*data-set-path-and-name*"
specifies an optional subdirectory and the SAS data set name. When you use this with the fileInfo action, you can use wildcards to match a specific set of file names. For more information, see "Using Wildcard Characters for Filename Matching" on page 123.

Default	none
Requirement	To use the SAS Data Connector to SAS Data Sets, you must specify SRCTYPE="PATH" in the DATASOURCE= argument when you add a caslib.
Note	When you use the loadTable action with a caslib of type PATH, the value that is specified in the path option on the loadTable action is appended to the path option value that is specified in the addCaslib action. The loadTable path option value can be a file name or a subdirectory with a file name. In this case it is assumed the caslib was defined to allow subdirectories. You can specify the SAS7BDAT

extension. However, if the BASESAS filetype option is provided, you can omit the extension from the loadTable path option value.

POINTOBS="YES" | "NO"
specifies whether to locate a data set by returning an observation (OBS) number from a subset of observations.

Valid in PROC CASUTIL: SAVE statement

Default none

READ="*password*"
specifies the password to access and read a password-protected data set.

Default none

Requirement This option is required only if the data set is password-protected.

REUSE="YES" | "NO"
specifies whether to write new rows to freed space in a compressed data set.

Valid in PROC CASUTIL: SAVE statement

Default NO

Interaction You can set this option when you use the COMPRESS= option.

VARCHARCONVERSION=*column-length*
specifies the column length at which to begin converting CHAR to VARCHAR. Specify a number that is either greater than or equal to 1 (>=1) or that is less than or equal to 32767 (<= 32767).

Valid in PROC CASUTIL: LOAD statement

Default none

Range >=1 or <=32767

WRITE="*password*"
specifies the password to access and save a password-protected data set.

Default none

Requirement This option is required only if the data set is password-protected.

Details

Access SAS Data Sets

To access SAS data sets requires a DATASOURCE= argument when you add a caslib with **`srctype="path"`** as the specified value. Except for SAS data sets that are protected by metadata-bound library controls, you can use the SAS Data Connector to SAS Data Sets to access any SAS data set.

For large data sets in particular, if SAS Cloud Analytic Services can access the SAS data set, you can use the CASUTIL procedure with the LOAD CASDATA= syntax.

If you can access the SAS data set with the LIBNAME statement, you can use the CASUTIL procedure with the LOAD DATA= syntax to load the data into SAS Cloud

Analytic Services. For an example, see sashelp.iris in load a client-side file on page 4. See the CASUTIL procedure for details about the LOAD CASDATA and LOAD DATA syntax.

Supported SAS Data Sets Data Types

This table shows the data types that can be loaded from SAS data sets into CAS. This table also shows the resulting data type for the data after it has been loaded into CAS. The length of the data format in CAS is based on the length of the source data.

Table 9.14 Supported SAS Data Sets Data Types

SAS Data Sets Data Type	CAS Data Type
CHAR	CHAR, VARCHAR
DOUBLE	DOUBLE

Examples

Example 1: Access a SAS Data Set Using the CASLIB Statement and the CASUTIL Procedure

```
/* Replace sasdata and the path with values for your site. */
libname sasdata '/data';

proc contents data=sasdata.customer;
run;

/* Start a CAS session. */
cas casauto;

/* Assign a Path caslib to the same location as the SAS library. */
caslib sasdata datasource=(srctype="path") path="/data";

/* Use the CASUTIL procedure with the LOAD CASDATA= syntax */
/* to have the server load the SAS data set from disk.     */
proc casutil;
  load casdata="customers.sas7bdat" casout="customers";
quit;

/* Use the CAS LIBNAME engine to print       */
/* the in-memory copy of the customers data. */
libname mycas cas sessref=casauto;

proc print data=mycas.customers;
run;
```

For information about the SAS language elements used in this example, see the following list:

- “CASLIB Statement” on page 91
- Chapter 5, “CASUTIL Procedure,” on page 110
- “CAS LIBNAME Statement” on page 66

Example 2: Access a SAS Data Set Using the CAS Procedure and Actions

```
/* Start a CAS session. */
cas casauto;

proc cas;
/* Add a PATH type CASLIB called sasdata and      */
/* point it to where the SAS data set is located. */
  table.addcaslib /
    caslib="sasdata"
    datasource={srctype="path"}
    path="/data";
run;

/* Use the loadTable action to load the SAS data set from disk. */
  table.loadTable /
    path="customers.sas7bdat"
    importOptions={fileType="basesas"}
    casout={name="customers", replace=True};
run;

/* Use the fetch action to print the first five records */
/* from the in-memory copy of the customers data.       */
  table.fetch / table="customers" to=5;
run;
```

For information about the CAS procedure and the actions used in this example, see the following list:

- "CAS Procedure" in *SAS Cloud Analytic Services: CASL Reference*
- "Add caslib" in *SAS Viya: System Programming Guide*
- "Load table" in *SAS Viya: System Programming Guide*
- "Fetch rows" in *SAS Viya: System Programming Guide*

Example 3: Save a SAS Data Set Using the CAS Procedure and Actions

```
table.save /
   caslib="casuser"
   table="xyz"
   name="test/xyz.sas7bdat"
      exportOptions={filetype="basesas",
                     encoding="utf-8",
                     encrypt="aes2",
                     compress="char",
                     password="myPwd",
                     encryptionpassword="myEncryptPwd"};
run;
```

Example 4: Delete a SAS Data Set Using the CAS Procedure and Actions

```
table.deleteSource /
   caslib="casuser"
   source="caspp7/test/delthis.sas7bdat"
      deleteOptions={filetype="basesas"
```

```
        encryptionpassword="myEncryptPwd"};
run;
```

SAS Data Connector to Teradata and SAS Data Connect Accelerator for Teradata

SAS Data Connector to Teradata enables you to load data serially from Teradata into SAS Cloud Analytic Services. All users can use SAS Data Connector to Teradata. SAS Data Connect Accelerator for Teradata is a separately licensed product that enables you to load data in parallel using the SAS Embedded Process. Data connector options are used in the context of different statements that connect your data in Teradata with CAS.

Valid in: CASLIB statement

PROC CASUTIL statements (see options for details)

Examples: Load a Teradata data source and add a caslib to it.

```
caslib tdlib
     datasource=(srctype='teradata'
                 dataTransferMode='parallel'
                 server='TDserver'
                 username='user1'
                 password='myPwd'
                 database='TDdatabase');
```

Load Teradata source data using PROC CASUTIL.

```
proc casutil;
   load incaslib='tdlib' casdata='cars'
        casout='cars_CAS'
        options=(dataTransferMode='parallel');
quit;
```

Syntax

Data Connector Options for Teradata

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see "Where to Specify Data Connector Options" on page 170.

AUTHENTICATIONDOMAIN="*domain*"
specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) that are used to access a data source.

Typically, you specify an AUTHENTICATIONDOMAIN= value when you add a caslib. The associated credentials are then used for any statement that accesses the data source. If you specify an AUTHENTICATIONDOMAIN= value in a statement other than the CASLIB statement, these credentials override any that were set when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements

Alias	AUTHDOMAIN=
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.

BUFFERSIZE=*bytes*
specifies the buffer size length in bytes of the buffer that is used to send data to or receive data from SAS embedded processes. This value overrides a value of BUFFERSIZE= that is set in the CASLIB statement. Increasing the size might result in better performance with a trade-off of increased memory usage.

Valid in	CASLIB statement PROC CASUTIL: LOAD and SAVE statements
Default	1048576
Restriction	This option applies only with SAS Data Connect Accelerator for Teradata (DATATRANSFERMODE="parallel").

CATALOG="*catalog-name*"
specifies a logical catalog name. The logical name can be any user-defined name. This name is displayed in the Catalog column for all tables in the results from the CONTENTS statement in PROC CASUTIL.

Valid in	CASLIB statement
Default	Active caslib

CHARMULTIPLIER=*value*
specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in	CASLIB statement PROC CASUTIL: LOAD statement
Default	1.0
Range	1.0–5.0

CLIENT_ENCODING="*encoding*"
specifies the encoding that the data source client uses to transmit CHAR and VARCHAR data. Typically, this corresponds to the LANG environment variable.

If you have configured the Teradata client to use a different encoding than the LANG environment variable, set CLIENT_ENCODING= to the corresponding encoding value.

The CLIENT_ENCODING= value is independent of the NLS_LANG environment variable. For encodings other than UTF-8, the data is transcoded into UTF-8 when the data is loaded into CAS.

Valid in	CASLIB statement
Default	UTF-8
Restriction	This option applies only with a SAS Data Connector (DATATRANSFERMODE="serial").

DATABASE="*database-name*"
specifies the name of the database.

When you specify a value for DATABASE=, the table name is qualified with the database name. For example, if you set DATABASE="myDB" and you want to access table Studydata, the table myDB.Studydata is accessed.

Valid in	CASLIB statement [Required]
	PROC CASUTIL: LOAD statement
Alias	DB=
Default	none
Interaction	If you supply values for SCHEMA= and DATABASE=, the value for SCHEMA= is used to qualify a table name.

DATATRANSFERMODE="AUTO" | "PARALLEL" | "SERIAL"
specifies the mode of data transfer. If you specify this option outside of a CASLIB statement, this value overrides the value in a CASLIB statement.

Here are the valid values.

AUTO	specifies to first try to load the data in parallel using embedded processing. If this fails, a note-level message is issued and serial processing is then attempted.
PARALLEL	specifies to load the data in parallel by using the SAS Data Connect Accelerator to your database or data source.
SERIAL	specifies to load the data serially by using the SAS Data Connector to your database or data source.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD and SAVE statements
Aliases	DATATRANSFER=
	DTM=
Default	SERIAL
Requirement	To use the PARALLEL option, you must have a licensed copy of the SAS Data Connect Accelerator for your database or data source.
Interaction	If NUMREADNODES= or NUMWRITENODES= is a value other than 1 and DATATRANSFERMODE="AUTO", SAS attempts first to transfer the data in parallel using the data connect accelerator. If parallel transfer fails, SAS attempts to transfer data with the data

connector using the specified number of CAS worker nodes. If multiple nodes are not available, SAS transfers the data serially.

DBCREATETABLEOPTS="*text*"
specifies any table options that should be appended to a CREATE TABLE statement that is passed to the data source.

Valid in	PROC CASUTIL: SAVE statement
Default	none
Restriction	The DBCREATETABLEOPTS= option applies to serial data transfer only.

INSERTBUFF="*number-of-rows*"
specifies the number of rows per block of data to save to a table in the data source.

Valid in	CASLIB statement PROC CASUTIL: SAVE statement
Default	Calculated automatically based on row size
Supports	serial data transfer

NUMREADNODES=*number*
specifies the number of CAS worker nodes to use for a multinode load of data into CAS. Setting this option to a value other than 1 enables a data connector to use multiple nodes to load data into CAS. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in	PROC CASUTIL: LOAD statement
Default	1
Restriction	The NUMREADNODES= option applies to serial data transfer only.

NUMWRITENODES=*number*
specifies the number of CAS worker nodes to use for a multinode save of data from CAS into a data source table. Setting this option to a value other than 1 enables a data connector to use multiple nodes to save data to a data source table. You can specify any value up to the total number of available worker nodes. If the available number of nodes is smaller than the value that you specify, all available nodes are used. Specify a value of 0 to use all available worker nodes.

Valid in	PROC CASUTIL: SAVE statement
Default	1
Restriction	The NUMWRITENODES= option applies to serial data transfer only.

PASSWORD="*password*"
specifies the DBMS password for a user. You typically specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify a USERNAME= and PASSWORD= value in a

statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	PASS=
	PWD=
Default	none
Requirement	If your database or data source requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

READBUFF="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in	CASLIB statement
	PROC CASUTIL: LOAD statement
Aliases	RAS=
	ROW_ARRAY_SIZE=
Default	Calculated automatically based on row size
Restriction	This option is not valid with the SAS Data Connect Accelerator (DATATRANSFERMODE="parallel"). Specify BUFFERSIZE= when loading data in parallel.

ROLE="*name*"
specifies the Teradata role name.

Valid in	CASLIB statement
Default	"" (empty string)
Restriction	This option is not valid with the SAS Data Connect Accelerator (DATATRANSFERMODE="parallel").

SCHEMA="*schema-name*"
specifies the schema name to use for the connection to the database or data source.

When you specify a value for SCHEMA=, the table name is qualified with the schema name. For example, if you set SCHEMA="mySchema" and you want to access table Studydata, the table mySchema.Studydata is accessed. Specify a non-blank value when you set a value for SCHEMA=.

If you specify a value for SCHEMA= in a statement other than the CASLIB statement, this value overrides any value of SCHEMA= that was set when the caslib was added.

Valid in	CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements

Default none

Interaction If you specify values for SCHEMA= and DATABASE=, the value for SCHEMA= is used to qualify the table name.

SERVER="*server-identifier*"
specifies the server identifier for the DBMS server.

Valid in CASLIB statement [Required]

Default none

SRCTYPE="*teradata*"
specifies that the data source is a Teradata database.

Valid in CASLIB statement [Required]

Default none

STATUSINTERVAL=*number*
specifies whether to print a message to the client when a node adds *n* buffers to the table by a node, where *n* is the value of this option. If you specify a value for this option outside of a CASLIB statement, it overrides the value in a CASLIB statement.

Valid in CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default 0 (no message)

Restriction This option applies only to a SAS Data Connect Accelerator (DATATRANSFERMODE="PARALLEL").

USENARROWCHARACTERTYPES=TRUE | FALSE
when set to TRUE, specifies to use narrow character types, such as CHAR or VARCHAR, for all character variables when writing data to the data source. When set to FALSE, this option specifies to use wide character types, such as NCHAR or NVARCHAR, for all character variables when writing data to the data source.

Because the CAS server uses UTF-8 encoding by default, this option defaults to FALSE.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default FALSE

USERNAME="*user-name*"
specifies the database or data source user name.

Typically, you specify USERNAME= and PASSWORD= values (or an AUTHENTICATIONDOMAIN= value) when you add a caslib. These credentials are then used for any statement that accesses the data using that caslib. If you specify USERNAME= and PASSWORD= values in a statement other than the CASLIB statement, these credentials override any credentials that were specified when the caslib was added.

Valid in	CASLIB statement [see Requirement]
	PROC CASUTIL: CONTENTS, LIST, LOAD, and SAVE statements
Aliases	UID=
	USER=
Default	none
Requirement	If it is required by your database or data source, you must specify valid credentials to access data. You can provide these credentials by specifying either USERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Details

Teradata Naming

The data connector and data connect accelerator can load Teradata tables with names up to 128 characters or with column names that are up to 128 characters.

Loading Data in Parallel

If the SAS Data Connect Accelerator for Teradata is installed on your system, you can use it to load data in parallel, by specifying DATATRANSFERMODE="parallel". When you load data in parallel, the data connect accelerator uses the Teradata database's hash distribution of the data to spread the data across multiple connections for parallel loading into CAS. The data distribution in the database is determined by the Teradata Primary Index (PI) for tables or by a hash that is calculated by the database for Teradata views. Talk to your Teradata administrator or review the Teradata user documentation to get more information about how Teradata distributes data across its units of parallelism (AMPs). The more evenly the data is divided, the more efficiently the data can be loaded into CAS.

Security: Authentication with Teradata

Authentication to a Teradata Database

Credentials are required to access the data in a Teradata database. Typically, you supply credentials when you add a caslib. To do this, use the CASLIB statement and specify USERNAME= and PASSWORD= values. These credentials are then used for any statement that accesses the data using that caslib.

Alternatively, your system administrator might have defined an authentication domain that is associated with a pair of user ID and password values. The authentication domain and associated credentials are then available to you and other users who need to access data. In this case, you can specify an authentication domain when you add a caslib by using the AUTHENTICATIONDOMAIN= option. Use the AUTHENTICATIONDOMAIN= option instead of USERNAME= and PASSWORD= values.

You can specify different credentials when you issue a statement that accesses the data. If you originally specified an authentication domain when you added a caslib, then you must specify both a USERNAME= value and a PASSWORD= value to override the original credentials. You can also override authentication domain credentials with a different AUTHENTICATIONDOMAIN= value.

Although it is not typical, it is possible to supply USERNAME= and PASSWORD= values in separate statements. For example, you might supply a USERNAME= value in the CASLIB statement, and then you supply a PASSWORD= value when you specify a LOAD statement that accesses the data. In this situation, you must supply the PASSWORD= value for each statement that accesses the data.

Authentication Using TD2

Teradata 2 (TD2) is the default authentication method that is provided by Teradata. When TD2 is the default authentication method, use your standard user ID and password values. To use TD2 authentication when TD2 is not the default authentication method, append the "@TD2" token to the Teradata user name and supply the corresponding password. Appending the "@TD2" token to a user name overrides any default authentication that is configured.

Note: Support for TD2 authentication with the Teradata data connector was added in SAS Viya 3.3.

This example shows how to connect using a TD2 user name:

```
caslib tdlib
   datasource=(srctype='teradata'
               dataTransferMode='parallel'
               server='TDserver'
               username='janedoe@TD2'
               password='janedoepwd'
               database='TDdatabase');
```

For more information about TD2 authentication, see the Teradata security documentation.

Authentication Using Single Sign-On with Kerberos

Kerberos authentication allows nodes to communicate over a network after providing credentials to the system in a secure manner. Typically, symmetric key encryption is used to mask credentials during the authentication process. Kerberos authentication provides mutual authentication between the nodes on a network. Once authenticated, nodes can communicate freely across the network.

Note: Support for Kerberos authentication was added to the Teradata data connector in SAS Viya 3.3.

To use single sign-on (SSO) to Teradata with Kerberos authentication, your system must have a valid Kerberos ticket (Ticket Granting Ticket or TGT) before you can initiate a connection to Teradata using Kerberos authentication. On Linux systems, one way to generate a TGT is to use the **`kinit`** command. Other methods are available to the security administrator at your site. However, methods to authenticate are specific to your site. The security administrator must properly set the TGT expiration time so that any long running jobs have time to complete.

Here are the ways to initiate Kerberos authentication to Teradata with your data connector:

- Remove USERNAME= and PASSWORD= options or the AUTHENTICATIONDOMAIN= option from any statement that uses these options in existing code. Omit these options when you enter new code. This is the recommended way to specify Kerberos single sign-on as your authentication method.
- Indicate SSO by appending your user ID with "@KRB5", such as **`USERNAME="myname@KRB5"`**. Alternatively, you can specify simply **`USERNAME="@KRB5"`**. This syntax ignores any values that you supply for

USERNAME= and PASSWORD= and sets the authentication method to Kerberos (KRB5).

Note: When using Kerberos authentication, do not also specify the AUTHENTICATIONDOMAIN= option.

Authentication Using LDAP

LDAP is an authentication method that uses an LDAP server to store user names and passwords. Peripheral applications pass user names and passwords to the LDAP server for user authentication. The LDAP authentication system at your organization might also use security realms. These realms define groups, such as administrators or managers, that have access to defined roles and subsets of information.

For LDAP authentication with either a NULL or single security realm, append only the "@LDAP" token to the Teradata user name. In this case, no realm name is needed. If you append a realm name, the LDAP authentication server ignores it and authentication proceeds.

Note: Support for LDAP authentication with the Teradata data connector was added in SAS Viya 3.3.

This example shows how to connect to a single or NULL realm:

```
caslib tdlib
   datasource=(srctype='teradata'
               dataTransferMode='parallel'
               server='TDserver'
               username='johndoe@LDAP'
               password='johndoepwd'
               database='TDdatabase');
```

If your system is configured with multiple security realms, you must append the realm name to the "@LDAP" token. In this case, an LDAP server must already be configured to accept authentication requests from the Teradata server.

Here is an example of how to make the connection to a specific realm, jsrealm, where multiple realms are configured:

```
caslib tdlib
   datasource=(srctype='teradata'
               dataTransferMode='parallel'
               server='TDserver'
               username='johndoe@LDAPjsrealm'
               password='johndoepwd'
               database='TDdatabase');
```

Single sign-on is not supported with LDAP authentication.

Supported Teradata Data Types

The following table shows the data types that can be read into CAS from a Teradata database. This table also shows the resulting data type and format when data is read into CAS. The length of the data format in CAS is based on the length of the source data.

Table 9.15 *Supported Teradata Data Types*

Teradata Data Type	CAS Data Type	Default Format in CAS
	Character	

Teradata Data Type	CAS Data Type	Default Format in CAS
CHAR	CHAR	$CHAR*w.*
VARCHAR	VARCHAR	$CHAR*w.*
	Numeric	
DECIMAL	DOUBLE	*w.d*
DOUBLE	DOUBLE	*w.d*
INTEGER	INTEGER	*w.*
BYTEINT (I1)	INTEGER	*w.*
SMALLINT (I2)	INTEGER	*w.*
BIGINT (I8)	BIGINT	*w.*
	Date and Time	
DATE (DA)	DOUBLE	DATE*w.*
TIME (AT)	DOUBLE	TIME*w.d*
TIMESTAMP (TS)	DOUBLE	DATETIME*w.d*

VARCHAR Data and the CAS Server

The CAS server supports loading, storing, and writing VARCHAR data. All tasks that can be completed using the CAS server use VARCHAR data. Any tasks that are not completed by the CAS server convert VARCHAR data to fixed-length CHAR data before the data is printed or saved. For more information, see "VARCHAR Support for Implicit and Explicit Data Type Conversion" on page 62.

Integer Data Types and Numeric Precision

The CAS server supports loading, storing, and writing integer data types (INT32 and INT64). Some computations that can be completed by using the CAS server maintain the original data type. Consult the documentation to determine whether a CAS action supports an integer type. Any tasks that are not completed by the CAS server convert these data types into a SAS DOUBLE before the data is printed or saved.

A SAS DOUBLE value maintains approximately 15 digits of precision. When you read values that contain more than 15 decimal digits of precision into a DATA step, the data is converted to a DOUBLE value. When you read these values into a procedure that is not included with SAS Viya, the values are rounded and converted to a DOUBLE value with 15 digits of precision. Most procedures are supported on the CAS server. However, it is recommended that you verify that a procedure is supported before you use it for large numeric values. For more information, see *Base SAS Procedures Guide*.

Examples

Example 1: Specify a Teradata Database as a Data Source for a Caslib In SAS Cloud Analytic Services

Use the CASLIB statement to initialize the data source and add the caslib for Teradata. No connection is made to the database until a statement that accesses the data is called.

The data is read serially into the caslib Teradatacaslib.

Teradata credentials are required to access the data. You can specify these in the CASLIB statement or when you use statements that access the data.

```
caslib TDlib desc='Teradata Caslib'
     dataSource=(srctype='Teradata',
                 dataTransferMode='serial',
                 server='teradataServer',
                 username='user1',
                 password='******',
                 );
```

Example 2: Load Teradata Data into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="Teradatacaslib";                    1
   load casdata="myTDdata" incaslib="Teradatacaslib" outcaslib="casuser"
        casout="TDdata_from_Teradatacaslib";                2
   list tables incaslib="casuser";                          3
   contents casdata="TDdata_from_Teradatacaslib" incaslib="casuser";
                                                            4
quit;
```

1 List the tables in Teradatacaslib before loading your data.

2 Load the table myTDdata from Teradata into caslib Casuser. Call the new table TDdata_from_Teradatacaslib.

3 List the tables in caslib to see the newly created table, TDdata_from_Teradatacaslib, that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

Chapter 10

Data Types

SAS Cloud Analytic Services Data Types

Overview

A *data type* is an attribute of every column in a table that specifies the type of data that the column stores. For example, the data type is the characteristic of a piece of data that indicates whether it is a character string, an integer, a floating-point number, a date, or a time. The data type also determines how much memory to allocate for the column value.

SAS Cloud Analytic Services currently supports the data types that are covered in this section, which support missing values.

Character Data

CHAR(*n*)
stores a fixed-length character string, where *n* is the maximum number of characters to store. This maximum is required to store each value regardless of the actual size of the value. If CHAR(10) is specified and the character string is only five characters long, the value is right-padded with spaces.

Note: This data type cannot contain ANSI SQL null values.

VARCHAR(*n*)
Stores a varying-length character string, where *n* is the actual number of characters to store. If VARCHAR(10) is specified and the character string is only 5 characters long, the value is 5. It is not padded with spaces.

Note: This data type cannot contain ANSI SQL null values.

VARCHAR Benefits and Considerations

In most cases you can take advantage of the benefits of using VARCHAR instead of CHAR. Here are some examples.

- the lengths of the character data vary significantly.
- the longest strings are infrequent and would require a fixed length of 64 bytes.

In other cases, however, it is better to use a fixed-width column when data is consistently short—namely, less than 16 bytes, such as an ID column of airport codes—because it uses less memory and runs faster.

In addition, keep these considerations in mind for variables with an undefined maximum length:

- VARCHAR(*) indicates that no maximum length on the column is being defined.
- Using VARCHAR(*) can be helpful if the maximum length of data for a column is not known when the column is being defined.

There is another consideration to keep in mind when you use VARCHAR(*). If you copy a table that is defined with a VARCHAR(*) to an engine library that does not support VARCHAR, a CHAR data type is created instead and is defined with the maximum length of 32767 bytes. If you instead provide an explicit length, such as VARCHAR(10), a CHAR column is created in the new table with a byte length of 40. A maximum length of 40 bytes is required to hold 10 characters in a UTF-8 session.

Numeric Data

DOUBLE
: Stores a signed, approximate, double-precision, floating-point number. Allows numbers of large magnitude and permits computations that require many digits of precision to the right of the decimal point. For SAS Cloud Analytic Services, this is a 64-bit double precision, floating-point number.

Data Types for SAS Cloud Analytic Services Table Columns

Table 10.1 *Data Types for SAS Cloud Analytic Services Table Columns*

Data Type Definition Keyword	SAS Cloud Analytic Services Table Column Data Type	Data Type Returned	Missing Values
CHAR(*n*)	CHAR(*n*)	CHAR(*n*)	all blanks (the same as in SAS)
VARCHAR(*n*)	VARCHAR(*n*)	all blanks or a zero length	
DOUBLE	DOUBLE	DOUBLE	

Chapter 11

Functions

Dictionary

CLIBEXIST Function

Returns 0 when the specified caslib name is not found and 1 when the caslib is found.

Category: CAS

Alias: CLIBFOUND

Syntax

CLIBEXIST(*session name*,*caslib name*)

Arguments

session name
a valid session name.

caslib name
a valid caslib name.

Details

The CLIBEXIST function has access to sessions that are connected to the SAS client interfaces. To return a full set of caslibs that are known to the CAS server use the following:

```
caslib _ALL_ list;
```

Example

Statements	Results

`cas existingSession cashost=myhost casport=12345;`
`%put doIExist= %sysfunc(clibexist(mysess,notfound));`
`%put doIExist= %sysfunc(clibexist(existingSession,casuser));`
	
	`doIExist= 0`
	`doIExist=1`

GETCASURL Function

Returns the value for a URL for connecting to the CAS Server Monitor.

Category: CAS

Requirements: The server name identified by the SAS CASHOST= option is used when constructing the URL.

The value provided is a valid session name. If the value is not provided, the SAS SESSREF= option is valid.

Syntax

GETCASURL(<*session*>)

Optional Argument

session

If zero parameters are specified, then the SAS SESSREF= option value is used.

Example: GETCASURL Example

Statement	Result
`%put httpaddr= %sysfunc(getcasurl());`	`httpaddr=http://host and port value`

GETLCASLIB Function

Returns the caslib that was bound to a CAS LIBNAME engine libref using the CASLIB= option when it was assigned.

Category: CAS

Note: If a caslib is not bound, then the active caslib is returned.

Syntax

GETLCASLIB(*libref*)

Argument

libref
specifies the libref name.

Examples

Example 1

This example does not specify the CASLIB= option. The libref is not bound to a caslib. The result of the function is the active caslib for the session.

Statements	Results
`libname mycas cas ;` `%put "caslib name="  %sysfunc(getlcaslib(mycas));`	`caslib name=casuser(userid);`

Example 2

This example specifies a caslib in the CASLIB= option. This syntax binds the caslib to the libref and the result of the function shows the specified caslib.

Statements	Results

`caslib mycaslib datasource=(srctype=path) path='/tmp';`
 	
`libname mycas cas caslib=mycaslib;`
`%put "caslib name="  %sysfunc(getlcaslib(mycas));`
	`caslib name=mycaslib;`

GETLSESSREF Function

Returns the session reference that is associated with a CAS LIBNAME engine libref.

Category: CAS

Syntax

GETLSESSREF (*libref*)

Argument

libref
specifies the library reference name.

Example

Statements	Results

`cas mysess UUIDMAC=sessuuid;`
 	
`libname mycas cas;`
`%put "session name="  %sysfunc(getlsessref(mycas));`
	`session name=mysess;`

GETLTAG Function

Returns the tag that was associated with a CAS LIBNAME engine libref in the TAG= option when it is assigned.

Category: CAS

Notes: If a tag was not associated, a zero-length string ("") is returned.

The TAG= option is rarely used. It is useful after loading a server-side file into memory from a caslib that enables access to subdirectories with CASL.

Syntax

GETLTAG(*libref*)

Argument

libref
specifies the library reference.

Example

Statements	Results

`cas mysess UUIDMAC=sessuuid;`
 	
`libname mycas cas tag=test.models;`
`%put "tag="  %sysfunc(getltag(mycas));`
	`tag=test.models;`

GETSESSOPT Function

Returns the value for a SAS Cloud Analytic Services session option.

Syntax

GETSESSOPT (*session-name session-option-name*)

Required Arguments

session-name
CAS session name.

session-option-name
CAS session option name.

For more information, see Chapter 13, "Session Options," on page 281 .

TIP You can list the session option names with this code:

```
CAS mysess LISTSESSOPTS;
```

Example: Listing the Active CASLIB

This example returns the CASLIB option value.

Statements	Results

`%put caslib = %sysfunc(GETSESSOPT(mysess, caslib));`
 	
`data work.one;`
`  x = GETSESSOPT("mysess", "caslib");`
`  put x=;`
`run;`
	`x=CASUSERHDFS(userid)`

SESSFOUND Function

Returns a 0 when a CAS session is not connected to a server and a 1 when the session is connected to a server.

Syntax

SESSFOUND (*session-name*)

Required Argument

session-name
returns the name of the session that you are trying to find.

Details

This function has access to sessions that you started in your SAS session only. You can use the CAS statement with the LISTSESSIONS option to identify all your CAS sessions on a server.

When SAS writes data to the CAS server using the CAS engine, the engine estimates the number of bytes that are needed to transcode the data to UTF-8 based on the character set of the SAS session. SBCS environments estimate one byte in UTF-8 for every one byte in the local encoding. DBCS environments estimate 1.5 bytes in UTF-8 for every one byte in the local encoding. You can use the "CASNCHARMULTIPLIER= System Option" on page 302 system option to replace the estimate with an explicit value of the

byte multiplier when you know the number of bytes that are needed to represent the data in UTF-8.

The ENCODING= option identifies the character set.

Example: Determining CAS Sessions

This example shows whether the CAS session is found.

Statements	Results
`%put doIExist= %sysfunc(sessfound(mysess));`	`doIExist= 0`
`%put doIExist= %sysfunc(sessfound(existingSession));` `run;`	`doIExist= 1`

Chapter 12

Macro Variables

Dictionary

CASHOST Macro Variable

Specifies the name of the CAS server.

Default: Not defined

Range: 256 characters

Interactions: When set, this macro variable overrides SAS system option CASHOST.
The CAS statement HOST= option overrides this macro variable.

Tip: If you want to delete macro variable _CASHOST_ and allow SAS system option CASHOST to prevail, use the following statement:

```
%symdel _CASHOST_;
```

Syntax

%let _CASHOST_=cloud.example.com

CASPORT Macro Variable

Specifies the CAS server port.

Default: Not defined

Range: 0–65535

Interactions: When set, this macro variable overrides SAS system option CASPORT.
The CAS statement PORT= option overrides this macro variable.

Note: When set to 0, CAS selects a port number.

Tip: If you want to delete macro variable _CASPORT_ and allow SAS system option CASPORT to prevail, use the following statement:

```
%symdel _CASPORT_;
```

Syntax

%let _CASPORT_=5570

SESSREF Macro Variable

Stores the name of the active SAS Cloud Analytic Services session.

Default: Not defined

Interactions: When you use the CAS statement to create a new session, this macro variable is automatically set to the name of the new session, which is CASAUTO by default.

When you set SAS system option SESSREF, this macro variable is automatically set to the same value.

Chapter 13

Session Options

Setting Session Options

About the Session Options

The session options control various properties of your SAS Cloud Analytic Services session. To list the properties and their current setting for a session, use the LISTSESSOPTS option in a CAS statement. See “LISTSESSOPTS ” on page 29. To see the setting for a specific property, use the GETSESSOPT function. See “GETSESSOPT Function” on page 276.

Setting Session Options for a New Session

How the Session Option Values Are Determined

When you create a new session, the value for each of the session properties is provided by the following sources in descending order of precedence:

- options specified in the SESSOPTS= option in the CAS statement.
- when you set SAS system options CASTIMEOUT and CASNWORKERS in SAS, CASTIMEOUT for session option TIMEOUT and CASNWORKERS for session option NWORKERS.

  Note: The CASTIMEOUT and CASNWORKERS system options have effect only after you set them in SAS. Otherwise, they are ignored.
- configuration parameters specified in the SAS Cloud Analytic Services configuration file
- command-line options that are used in the server start-up command
- SAS Cloud Analytic Services system defaults

You can browse the configured and default option values on the Configuration page of the CAS Server Monitor. For each option, this page shows the current value and the source of the value. The values for the session options are used as defaults for the session options, unless they are overridden. For information about the Server Monitor, see SAS Viya Administration: Using CAS Server Monitor.

Overriding the Default Session Option Values

To override the default session option values for a new session, use the option shown in the following table to complete the desired task.

Task	Option to use:
Override one or more session options for a new session only.	CAS statement SESSOPTS= option
Override the TIMEOUT option value for all subsequently created sessions.	SAS system option CASTIMEOUT=****
Override the NWORKERS option value for all subsequently created sessions.	SAS system option CASNWORKERS=*****

* TIMEOUT in the CAS statement SESSOPTS= option overrides this option.

** NWORKERS in the CAS statement SESSOPTS= option overrides this option.

*** Setting this option does not affect existing sessions.

Setting Options for Existing Sessions

For an existing session, use the option shown in the following table to complete the desired task.

Task	Option to use:
Change one or more session options for a specific session.	CAS statement SESSOPTS option
Change one or more session options for the active session.*	SAS system option CASSESSOPTS= (alias SESSOPTS=)
Specify a caslib for the active session.*	SAS system option CASLIB=

* SAS system option SESSREF stores the name of the currently active session.

Session Options by Category

Category	Language Elements	Description
Action	APPTAG= Session Option (p. 284)	specifies the string to prefix to log messages.
Caslib	CASLIB= Session Option (p. 285)	specifies the caslib name to set as the active caslib.
	MAXTABLEMEM= Session Option (p. 292)	specifies the maximum amount of memory, in bytes, to allocate for a table.
Data Processing	EVENTDS Session Option (p. 290)	specifies one or more data sets that define events.
	INTERVALDS Session Option (p. 291)	specifies one or more interval-name=value pairs, where the value is the name of a data set that contains user-defined intervals.
Data Quality	DQLOCALE Session Option (p. 289)	specifies the default 5-letter SAS Quality Knowledge Base (QKB) ISO locale code to use as the default locale for data quality (DQ) operations.
	DQSETUPLOC Session Option (p. 290)	specifies the name of the default SAS Quality Knowledge Base (QKB) to use for data quality operations.
DATA step	DATASTEPFMTERR= Session Option (p. 288)	specifies how SAS reacts when a specified variable format cannot be found.
	DATASTEPMSGSUMLEVEL= Session Option (p. 288)	specifies the DATA step message summary level.

Category	Language Elements	Description
	DATASTEPREPLACETABLE = Session Option (p. 289)	specifies whether a DATA step can replace an existing table.
Formats	FMTCASLIB Session Option (p. 290)	specifies the caslib where persisted format libraries are retained.
Input Control	EVENTDS Session Option (p. 290)	specifies one or more data sets that define events.
	INTERVALDS Session Option (p. 291)	specifies one or more interval-name=value pairs, where the value is the name of a data set that contains user-defined intervals.
Localization	LOCALE= Session Option (p. 291)	specifies the locale to use for sorting and formatting.
Log	LOGFLUSHTIME= Session Option (p. 292)	specifies the log flush time, in milliseconds.
	MESSAGELEVEL= Session Option (p. 293)	specifies the log message level.
	METRICS= Session Option (p. 293)	specifies whether to include default detailed performance metrics reports in the SAS log.
Session	NWORKERS= Session Option (p. 295)	specifies the number of worker nodes for a new session.
	TIMEOUT= Session Option (p. 295)	specifies the SAS Cloud Analytic Services session time-out in seconds for a new or existing session.
	TIMEZONE= Session Option (p. 296)	specifies the user local time zone.
Sort	COLLATE= Session Option (p. 287)	specifies the collating sequence for sorting.
System Administration: Performance	CMPOPT= Session Option (p. 285)	specifies the type of code generation optimizations to use in the SAS language compiler.

Dictionary

APPTAG= Session Option

specifies the string to prefix to log messages.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option

GETSESSOPT function

Category: Action

Default: No prefix

Syntax

APPTAG="*tag-string*"

CASLIB= Session Option

specifies the caslib name to set as the active caslib.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Caslib

See: "CASLIB Statement" on page 91

Syntax

CASLIB="*caslib-name*"

CMPOPT= Session Option

specifies the type of code generation optimizations to use in the SAS language compiler.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: System Administration: Performance

Syntax

CMPOPT="*optimization-value <optimization-value <…>>*" | "ALL" | "NONE"

Parameter Values

optimization-value <optimization-value <…> >
specifies the type of optimization that the SAS compiler is to use. Specify one or more of the following as a space-delimited list enclosed in quotation marks:

EXTRAMATH | NOEXTRAMATH
specifies whether the compiler is to retain or remove the extra mathematical operations that do not affect the outcome of a statement. Here are some of the mathematical operations that are either included or excluded in the machine language code.

$x * 1$ $x * -1$

$x \div 1$	$x \div -1$
$x + 0$	x
$x - x$	$x \div x$
$- -x$	any operation on two literal constants

Specify EXTRAMATH to retain the extra mathematical operations.

Default NOEXTRAMATH

FUNCDIFFERENCING | NOFUNCDIFFERENCING
specifies whether numeric-differencing derivatives or analytic derivatives are calculated for user-defined functions. Specify FUNCDIFFERENCING to calculate numeric-differencing derivatives for user-defined functions. Specify NOFUNCDIFFERENCING to calculate analytic derivatives for user-defined functions.

Default NOFUNCDIFFERENCING

GUARDCHECK | NOGUARDCHECK
specifies whether the compiler checks for array boundary problems. Specify GUARDCHECK to check for array boundary problems.

Default NOGUARDCHECK

Interaction NOGUARDCHECK is set when CMPOPT is set to ALL or NONE.

MISSCHECK | NOMISSCHECK
specifies whether to check for missing values in the data. Specify MISSCHECK to check for missing data.

Default NOMISSCHECK

Tip If the data contains a significant amount of missing data, specify MISSCHECK to optimize the compilation. Otherwise, specify NOMISSCHECK.

PRECISE | NOPRECISE
specifies whether exceptions are handled at an operation boundary or at a statement boundary. Specify PRECISE to handle exceptions at the operation boundary. Specify NOPRECISE to handle exceptions at the statement boundary.

Default NOPRECISE

SHORTCIRCUIT | NOSHORTCIRCUIT
specifies whether to optimize the evaluation of expressions that use multiple logical operators.

Note NOSHORTCIRCUIT is set when an expression includes derivatives and more than one logical tree or ternary operations.

Note If you specify a value more than once, then the last value that you specified is used. For example, if you specify values PRECISE, NOEXTRAMATH, and NOPRECISE, in that order, NOPRECISE is used.

Tips EXTRAMATH, MISSCHECK, PRECISE, GUARDCHECK, FUNCDIFFERENCING, and SHORTCIRCUIT can be specified in any combination.

All leading, trailing, and embedded blanks are removed.

Example Specify EXTRAMATH, MISSCHECK, and PRECISE:

```
cas casauto sessopts=(cmpopt="extramath misscheck precise");
```

ALL
specifies that the compiler is to optimize the machine language code by using the NOEXTRAMATH, NOMISSCHECK, NOPRECISE, NOGUARDCHECK, NOFUNCDIFFERENCING, and SHORTCIRCUIT optimization values.

Restriction ALL cannot be specified with other values.

NONE
specifies that the compiler is not set to optimize the machine language code by using the EXTRAMATH, MISSCHECK, PRECISE, NOGUARDCHECK, FUNCDIFFERENCING, and NOSHORTCIRCUIT optimization values.

Restriction NONE cannot be specified with other values.

See Also

"CMPOPT= System Option" in *SAS System Options: Reference*

COLLATE= Session Option

specifies the collating sequence for sorting.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Sort

Default: UCA

Syntax

COLLATE="MVA" | "UCA"

Parameter Values

MVA
specifies SAS client collating.

UCA
specifies a locale-appropriate collating sequence.

DATASTEPFMTERR= Session Option

specifies how SAS reacts when a specified variable format cannot be found.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: DATA step

Alias: FMTERR

Default: TRUE

Restrictions: For DATA steps, this option applies only when the DATA step runs in SAS Cloud Analytic Services. See "Controlling Where the DATA Step Runs" in *SAS Cloud Analytic Services: DATA Step Programming*.

The FMTERR alias is valid in the CAS procedure and in the GETSESSOPT function only. In the CAS statement and in system option SESSOPT, specify DATASTEPFMTERR.

See: *SAS Cloud Analytic Services: User-Defined Formats*

Syntax

DATASTEPFMTERR=TRUE | FALSE

Parameter Values

TRUE
when SAS cannot find a specified variable format, it writes an error message to the SAS log, and then stops processing. This is the default.

FALSE
when SAS cannot find a specified variable format, it writes a note to the SAS log, substitutes the $w. or BEST12. format for the missing format, and then proceeds with processing.

DATASTEPMSGSUMLEVEL= Session Option

specifies the DATA step message summary level.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: DATA step

Default: ALL

Tip: When the DATA step runs on multiple threads, the same message can be generated on each thread. In that case, use this option to control the summary level of the duplicate messages to help reduce the client log output.

Syntax

DATASTEPMSGSUMLEVEL=ALL | PUT | NONE

Parameter Values

In a DATA step, messages are received from each thread, which can result in a large number of duplicate messages when multiple threads are used. The first occurrence of all messages, including PUT statement messages, are sent to the client when they occur. By default, duplicate messages are summarized, and then sent to the client to reduce client log output. Specify one of the following values to control the level of summarization for duplicate messages:

ALL
summarizes all duplicate messages, including PUT statement messages, and sends them to the client when the DATA step exits. This is the default.

PUT
summarizes all duplicate messages, except PUT statement messages, and sends them to the client when the DATA step exits. All PUT statement messages are not summarized and are sent to the client as they occur.

NONE
does not summarize duplicate messages. All messages, including PUT statement messages, are sent to the client as they occur.

DATASTEPREPLACETABLE= Session Option

specifies whether a DATA step can replace an existing table.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: DATA step

Default: TRUE

Syntax

DATASTEPREPLACETABLE=TRUE | FALSE

DQLOCALE Session Option

specifies the default 5-letter SAS Quality Knowledge Base (QKB) ISO locale code to use as the default locale for data quality (DQ) operations.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Data Quality

Syntax

DQLOCALE="*5-letter-locale-code*"

Details

For a list of the 5-letter QKB ISO locale codes, see QKB Locale ISO Codes.

DQSETUPLOC Session Option

specifies the name of the default SAS Quality Knowledge Base (QKB) to use for data quality operations.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Data Quality

Syntax

DQSETUPLOC="*QKB-name*"

EVENTDS Session Option

specifies one or more data sets that define events.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Categories: Input Control
Data Processing

Default: The SAS predefined holiday events

Syntax

EVENTDS="*event-data-set(s)*"

Parameter Value

Event-data-set specifies the name of a data set that contains event definitions. You can use a one-level name or a two-level name such as libref.dataset. When specifying multiple names, separate each name with a space.

FMTCASLIB Session Option

specifies the caslib where persisted format libraries are retained.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Formats

Default: FORMATS

Note: This option is set by the system administrator.

INTERVALDS Session Option

specifies one or more interval-name=value pairs, where the value is the name of a data set that contains user-defined intervals.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Categories: Input Control
Data Processing

Interaction: This option overrides system option INTERVALDS for your session.

Tip: To specify interval data sets for all of your sessions, use the INTERVALDS system option instead.

Syntax

INTERVALDS="*interval-1=libref.dataset-name-1 <interval-2=libref.dataset-name-2 ...>*"

Parameter Value

Interval specifies the name of an interval. The value of interval is used to represent the set of intervals that is specified in data set *libref.dataset-name*. *Libref.dataset-name* specifies the libref and the data set name of the file that contains the user-defined intervals.

See Also

"INTERVALDS= System Option" in *SAS System Options: Reference*

LOCALE= Session Option

specifies the locale to use for sorting and formatting.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Localization

Default: en_US

Syntax

LOCALE="*locale*"

See Also

SAS National Language Support (NLS): Reference Guide

LOGFLUSHTIME= Session Option

specifies the log flush time, in milliseconds.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Log

Default: 100

Range: –1–86400

Syntax

LOGFLUSHTIME=–1 | 0 | *number*

Parameter Values

–1
flushes logs after each action completes.

0
flush logs as they are produced.

number
flushes logs in *number* milliseconds.

MAXTABLEMEM= Session Option

specifies the maximum amount of memory, in bytes, to allocate for a table.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Caslib

Default: 16M

Note: After this threshold is reached, the server uses temporary files and operating system facilities for memory management.

Tip: You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

Syntax

MAXTABLEMEM=*number*

Details

When data is added to the server (loading a table, appending rows, and so on) the server organizes the data in-memory blocks and allocates memory for the blocks. The MAXTABLEMEM= value is used to specify the amount of memory that each thread should allocate before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable. See SAS Cloud Analytic Services: Reference in *SAS Viya Administration: SAS Cloud Analytic Services*.

See Also

"Memory" in *SAS Cloud Analytic Services: Fundamentals*

MESSAGELEVEL= Session Option

specifies the log message level.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Log

Default: ALL

Syntax

MESSAGELEVEL="ALL" | "DEFAULT" | "ERROR" | "NONE" | "NOTE" | "WARNING"

METRICS= Session Option

specifies whether to include default detailed performance metrics reports in the SAS log.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Log

Default: FALSE

Example: Enable metrics for session Casauto:

```
cas casauto sessopts=(metrics=true);
```

Syntax

METRICS=TRUE | FALSE

Details

Session option METRICS= enables you to display information about the resources that your session consumes as each action in your program is executed. You can use the

metrics information to track the resources that your session consumes and make adjustments, if necessary. By default, metrics are disabled for your session. Specify session option METRICS=TRUE to enable default metrics for your session. When enabled, after each action is executed, the metrics are written to the SAS log as notes. One note is written for each available metric. The following table lists the default metrics that are written to SAS the log when metrics are enabled.

Table 13.1 *Default Metrics*

Metric	Note Written to the SAS Log	
The number of bytes moved.	`NOTE: bytes moved`	*number<units>*
The CPU time in seconds and as a percentage of cluster utilization. Cluster utilization is the sum of the utilization for each core in the cluster and can exceed 100%. For example, cluster utilization for a 96-core cluster where each core is 100% used is 9600%.	`NOTE: cpu time`	*number* seconds (*number*%)
The data movement time in seconds.	`NOTE: data movement time`	*number* seconds
The amount of memory used in bytes and as a percentage of the total available memory.	`NOTE: memory`	*number<units>* (*number*%)
The amount of time it took to run the action start to finish (real time) in seconds.	`NOTE: real time`	*number* seconds
The total available memory.	`NOTE: total memory`	*number<units>*
The total number of nodes and cores in the cluster.	`NOTE: total nodes`	*number* (*number* cores)

Here is an example of the default metrics that are written to the SAS log when the MDSUMMARY procedure is executed.

```
NOTE: real time               0.126269 seconds
NOTE: cpu time                1.036837 seconds (821.13%)
NOTE: data movement time      0.013819 seconds
NOTE: total nodes             27 (1296 cores)
NOTE: total memory            6.65T
NOTE: memory                  57.25M (0.00%)
NOTE: bytes moved             4.80K
NOTE: The SAS Cloud Analytic Server processed the request in 0.126269 seconds.
NOTE: The data set MYCAS.MPGHW_SUM has 15 observations and 19 variables.
NOTE: PROCEDURE MDSUMMARY used (Total process time):
      real time           0.38 seconds
      cpu time            0.01 seconds
```

NWORKERS= Session Option

specifies the number of worker nodes for a new session.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Session

Default: In order of descending precedence:

1. SAS system option CASNWORKERS, if you explicitly set it in SAS
2. 0 (all)

Range: 0–5000

Restriction: The number of workers can be set for new sessions only.

See: "CASNWORKERS= System Option" on page 303

Syntax

NWORKERS=*number*

TIMEOUT= Session Option

specifies the SAS Cloud Analytic Services session time-out in seconds for a new or existing session.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Session

Default: In order of descending precedence:

1. SAS system option CASTIMEOUT=, if you explicitly set it in SAS to a value greater than 0
2. 60

Range: 0–31536000

Notes: The session time-out starts when the number of connections to the session becomes zero and no actions are executing.

If a connection is established before the time-out expires, the time-out is canceled. Otherwise, the session is automatically terminated when the time-out expires.

When set to 0, the session is terminated immediately when the connection count becomes zero and no actions are executing.

See: "CASTIMEOUT= System Option" on page 306

Syntax

TIMEOUT=*number*

TIMEZONE= Session Option

specifies the user local time zone.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Session

Syntax

TIMEZONE='*time-zone-name*' | '*time-zone-ID*'

Parameter Values

'time-zone-name'
specifies a three- or four-character time zone name.

Default BLANK, which indicates that the client time zone and the CAS session time zone are the same.

See "Time Zone IDs and Time Zone Names" in *SAS System Options: Reference* for a list of time-zone names.

'time-zone-ID'
specifies a region/area value that is defined by SAS. When you specify a time zone ID, the time zone that the CAS session uses is determined by the time zone name and daylight saving time rules.

See "Time Zone IDs and Time Zone Names" in *SAS System Options: Reference* for a list of time-zone IDs.

See Also

"TIMEZONE= System Option" in *SAS System Options: Reference*

Chapter 14

System Options

Dictionary

AUTHINFO= System Option

Specifies a file where user ID and passwords are kept for authentication.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Alias: CASAUTHINFO=

Interaction: SAS Studio user credentials are used to authenticate the connection to CAS.

Syntax

AUTHINFO='*authinfo_file_path*';

Syntax Description

authinfo_file_path
specifies the path where an authinfo file is located. End users can store an encoded password in an authinfo file. The file provides an alternative to including passwords in programs. For more information, see "Create an Authinfo File" in *Client Authentication Using an Authinfo File*.

Details

A common use of this option is to submit code to CAS from the command line, in batch mode.

AUTHINFO= is also an environment variable and an option in the CAS statement. The environment variable can hold the name of one or more files. This variable is formatted like a PATH environment variable where a colon separates the filenames.

Here is the order of precedence to using the AUTHINFO options:

1. The AUTHINFO environment variable overrides the authinfo file.
2. The AUTHINFO= system option overrides the AUTHINFO environment variable.
3. The CAS statement AUTHINFO option overrides the AUTHINFO system option and the environment variable.

The shipped default for authorization file is $HOME/authInfo-file.

Examples

Example 1

Set AUTHINFO= system option. This option overrides the authinfo file pointed to by the AUTHINFO environment variable.

```
Options AUTHINFO='$HOME/authInfo-file';
```

Example 2

AUTHINFO can also be set as an environment variable. This option overrides the authinfo file.

```
Options insert=(set=AUTHINFO='$HOME/authInfo-file');
```

See Also

- "Create an Authinfo File" in *Client Authentication Using an Authinfo File*
- "CAS Statement" on page 19

CASDATALIMIT= System Option

Specifies the maximum number of bytes of data from a single CAS table that can be transferred from the CAS server to SAS.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Not supported

Category: CAS

PROC OPTIONS GROUP= CAS

Default: The shipped default is 100M.

Restriction: This option is valid for SAS 9.4M5. In SAS Viya, you can limit the data transferred from the CAS server to SAS by using the DATALIMIT LIBNAME option or the DATALIMIT data set option.

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Tip: This option can prevent you from accidentally transferring a large amount of data from the server to the client.

Syntax

CASDATALIMIT=*integer* | *integer*K | *integer*M | *integer*G | ALL

Syntax Description

integer
specifies the total number of bytes to read.

***integer*K**
specifies the total number of kilobytes to read.

***integer*M**
specifies the total number of megabytes to read.

***integer*G**
specifies the total number of gigabytes to read.

ALL
specifies that the entire file can be read, no matter how large it is.

Details

When you use the **`caslib all assign;`** statement to create CAS LIBNAME assignments for all defined caslibs, the default value of 100M is used as the maximum number of bytes that can be transferred from the CAS server to SAS. To override this value, use the CASDATALIMIT= system option.

See Also

- "DATALIMIT= LIBNAME Option" on page 69
- "DATALIMIT= Data Set Option" on page 78

CASHOST= System Option

Specifies the CAS primary and backup host names that are associated with a CAS session.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Tip: Add the CASHOST= option to a SAS 9 configuration file or an autoexec file to eliminate the need to set this option in a SAS program.

Syntax

CASHOST= "*primary-host-name*"

CASHOST= ("*primary-host-name*" <<,> "*backup-host-name*">)

Syntax Description

primary-host-name
specifies the CAS server name.

Default None

Range 1–256 characters

backup-host-name
specifies the backup CAS server name.

Valid in SAS Viya 3.3 and later releases

Default None

Range 1–256 characters

Restriction When you specify the CASHOST option in the GETOPTION function, only *primary-host-name* is returned. *backup-host-name* cannot be returned at this time.

Requirement Parentheses are required when you specify both a primary host and a backup host. You can use either a space or a comma between the primary host and the backup host.

Details

Determining the Backup Host Name

To determine whether a backup host is available, connect to the primary server host and port and run the following code:

```
proc cas;
   builtins.listNodes result = r;
run;
   print r.nodelist.where(role eq 'controller');
run;
```

If a second host name is printed, you can set it as the backup controller.

When to Set the CASHOST Option in a SAS Program

In SAS 9, if you or a SAS administrator has not configured the CASHOST= option in a configuration file or an autoexec file, you must specify the option in all programming environments.

In SAS Viya, the CAS server is configured during deployment. SAS Studio connects to the CAS server when you log on. You do not need to specify the CASHOST= option. To determine whether you are using SAS Studio for SAS Viya, select  and then select **About SAS Studio**. If **SAS release:** begins with V, SAS Studio is a SAS Viya application.

You must also specify the CASHOST= option if you change to a different CAS server primary or backup host.

When a Failure Occurs

If *primary-host-name* fails, *backup-host-name* automatically becomes the primary CAS server controller. Existing sessions automatically connect to *backup-host-name*. New sessions connect to *backup-host-name*. Controller failure is transparent to users. It is more of a concern for administrators. You can use PROC OPTIONS to see the details for the primary and backup hosts.

Example

```
options cashost="cloud.example.com";
options cashost=("cloud.example.com" "cloudbackup.example.com");
```

See Also

- "CAS Statement" on page 19
- Accessing SAS Viya from SAS 9
- SAS Cloud Analytic Services: How To (CAS Server Monitor)

CASLIB= System Option

Specifies the caslib name for the session that is identified by the SESSREF= option.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Interaction: The CAS statement session option CASLIB= overrides this option. For more information, see "CASLIB= Session Option" on page 285.

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Tip: A best practice is to explicitly set options for a session using the CAS statement SESSOPTS= option. Here is an example:

```
cas mysess sessopts=(caslib=mycaslib collate=UCA);
```

See “SESSOPTS=(*session-option(s)*)” on page 32.

Syntax

CASLIB="*name*"

Optional Argument

name
specifies the caslib name.

Default None

Range 1–128 characters

Example: Set the Default Caslib

```
options caslib="casuser";
```

See Also

Statements:

- “CAS Statement” on page 19
- “CASLIB Statement” on page 91

CASNCHARMULTIPLIER= System Option

Specifies a multiplication factor that is used to increase the number of bytes for a fixed character variable when data is transcoded to the UTF-8 encoding in order to run in the CAS server.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Not supported

Category: CAS

PROC OPTIONS GROUP= CAS

Defaults: For SBCS environments, the shipped default is 1.

For DBCS environments, the shipped default is 1.5.

Restriction: This option is valid for SAS 9.4M5. In SAS Viya, use the NCHARMULTIPLIER= LIBNAME option or data set option.

Note: This option cannot be restricted by a site administrator. For more information, see “Restricted Options” in *SAS System Options: Reference*.

Syntax

CASNCHARMULTIPLIER=*n*

Syntax Description

n

specifies a number that is used as a multiplication factor to control the byte size for fixed character data that is transcoded to UTF-8 when the CAS engine writes data from SAS to the CAS server.

Range $0 < n \leq 4$

Details

When the SAS session encoding is not UTF-8, the CAS engine transcodes CHAR and VARCHAR values from the SAS session encoding to UTF-8. When DBCS characters and some SBCS characters are transcoded to UTF-8, they require additional bytes to represent a character. The CAS engine makes a best guess for the number of bytes that are needed to hold the transcoded character data. You can use the CASNCHARMULTIPLIER= option to replace the best guess with an explicit value of the byte multiplier when you know the number of bytes that are needed to represent the data in UTF-8.

Note: For example, when the SBCS characters for the accented characters in the Latin1 encoding are transcoded, they require additional bytes.

CAUTION:

Data truncation can occur. If the value of CASNCHARMULTIPLIER= option is too small and the data is truncated, an error occurs. A best practice is to test reading DBCS data by the CAS engine before the data is used in a production environment.

See Also

- “NCHARMULTIPLIER= LIBNAME Option” on page 70
- “NCHARMULTIPLIER= Data Set Option” on page 80

CASNWORKERS= System Option

Specifies the number of worker nodes to use for a CAS session.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Default: ALL

Restriction: The number of worker nodes can be set for new sessions only during session creation.

Note: This option can be restricted by a site administrator. For more information, see “Restricted Options” in *SAS System Options: Reference*.

Tip: The CAS statement session option NWORKERS= overrides this option. For more information, see “CAS Statement” on page 19 and “NWORKERS= Session Option” on page 295.

Syntax

CASNWORKERS= ALL | *number*

Syntax Description

ALL
specifies to use all of the worker nodes.

number
specifies the number of worker nodes to use.

In SMP mode, *number* is always 0, whether you set CASNWORKERS= to 0 or 1.

In MPP mode, specify *number*=0 to use the maximum number of worker nodes that are available. You can set CASNWORKERS= to a number that is less than or equal to the maximum number of worker nodes.

Range 0–5000

Example

```
options casnworkers=10;
```

CASPORT= System Option

Specifies the port to use when connecting to CAS.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Default: 0

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Syntax

CASPORT=*port-number*

Syntax Description

port-number
specifies the CAS server port number.

Range 0–65535

Note When *port-number* is set to 0, CAS selects a port number.

Example

```
options casport=12345;
```

CASSESSOPTS= System Option

Specifies one or more session options for the active CAS session.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Alias: SESSOPTS=

Default: None

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Tip: A best practice is to explicitly set options for a session using the CAS statement SESSOPTS= option. Here is an example:

```
cas mysess sessopts=(caslib=mycaslib collate=UCA);
```

See "SESSOPTS=(*session-option(s)*)" on page 32.

Syntax

CASSESSOPTS=(*session-option(s)*)

SESSOPTS=(*session-option(s)*)

Syntax Description

session-option(s)

specifies one or more session options as *option=value* pairs separated by a space and enclosed in parentheses.

Tip To reflect a session option value, use this statement:

```
%put caslib=%sysfunc(GETSESSOPT(session, option)):
```

See Chapter 13, "Session Options," on page 281 for a list of the options that you can specify for *session-option(s)*.

Example

For the default session, set the caslib to MYCASLIB and the session connection time-out to 60 minutes:

```
options sessopts=(caslib="mycaslib" timeout=3600)
```

See Also

Functions:

- "GETSESSOPT Function" on page 276

Macro Statement and Functions:

- "%PUT Statement" in *SAS Macro Language: Reference*
- "%SYSFUNC and %QSYSFUNC Functions" in *SAS Macro Language: Reference*

Statements:

- "CAS Statement" on page 19

CASTIMEOUT= System Option

Specifies the CAS session time-out in seconds for new sessions. The session time-out starts when the number of connections to the session becomes zero and all session activity is complete.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Default: 60

Interaction: The TIMEOUT= session option overrides this option. For more information, see "TIMEOUT= Session Option" on page 295.

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Tip: To change the time-out for an existing session, use the TIMEOUT= session option.

Syntax

CASTIMEOUT= *n* | *n*K | *n*M | *hex*X | MAX | MIN

Syntax Description

***n* | *n*K | *n*M**

specifies the session time out in seconds that are processed in multiples of 1, 1,024 (K) or 1,048,576 (M). For example, a value of 432008 specifies 43,200 seconds, and a value of 43k specifies 44,032 seconds.

Range 0–31536000

Note This option is ignored when the value is set to 0. In that case, the default for the TIMEOUT= session option applies.

***hex*X**
specifies the session time out seconds as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by an X. For example, the value 0a8c0x sets the number of seconds to 43200 seconds.

MAX
sets the time out value to 31536000.

MIN
sets the time out value to 0.

Details

If a connection is established before the time-out expires, the time-out is canceled. Otherwise, the session is automatically terminated when the time-out expires.

Example

```
options castimeout=28800;
```

CASUSER= System Option

Specifies the user ID to use when connecting to CAS.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Alias: CASUSERID=

Default: None

Note: This option cannot be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Syntax

CASUSER= *user-ID*

CASUSERID= *user-ID*

Syntax Description

user-ID
specifies your user ID.

Requirement The user ID that you specify must match a user ID in your personal .authinfo file. For more information about the .authinfo file, see "Authinfo File Requirement" in *SAS Viya: System Programming Guide*.

Interactions SAS Studio user credentials are used to authenticate your connection to CAS. After you are logged in to SAS Studio, you can submit code to CAS without using this option. The most frequent use of this option is to submit code to CAS from the command line, in batch mode.

The CAS statement USER= option overrides the user ID specified by this option. If the user ID is not specified in the CAS statement, SAS looks for a user ID that is set by the CASUSER= option. For more information, see "USER=*user-ID*" on page 33.

Example

```
options casuser=myid;
```

DSCAS System Option

Specifies whether the DATA step runs in the CAS server or in SAS by default.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

SAS Viya: Not supported

Categories: Environment Control: Language Control

CAS

PROC OPTIONS GROUP= LANGUAGECONTROL

Default: The shipped default is DSCAS.

Restriction: This option is valid for SAS 9.4M5.

Interaction: If the SESSREF option is specified in the DATA statement, SAS always attempts to run the DATA step in CAS, regardless of the value of the DSCAS system option.

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Syntax

DSCAS | NODSCAS

Syntax Description

DSCAS
specifies that by default, the DATA step runs in the CAS server.

NODSCAS
specifies that by default, the DATA step runs in SAS.

See Also

"Controlling Where the DATA Step Runs" in *SAS Cloud Analytic Services: DATA Step Programming*

SESSREF= System Option

Specifies the name of the default CAS session to use if a statement or procedure does not explicitly identify a session reference name.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Alias: CASNAME

Default: CASAUTO

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS System Options: Reference*.

Syntax

SESSREF= *session-name*

Syntax Description

session-name
specifies a valid SAS name that is less than 256 characters and is not enclosed in quotation marks.

Range 1–256 characters

Interactions When you create a session using the CAS statement, the value of the SESSREF= option and the _SESSREF_ macro variable are set to the session name. For more information, see "CAS Statement" on page 19.

When you name a session using the SESSREF= option, the value of the _SESSREF_ macro variable is set to the same name.

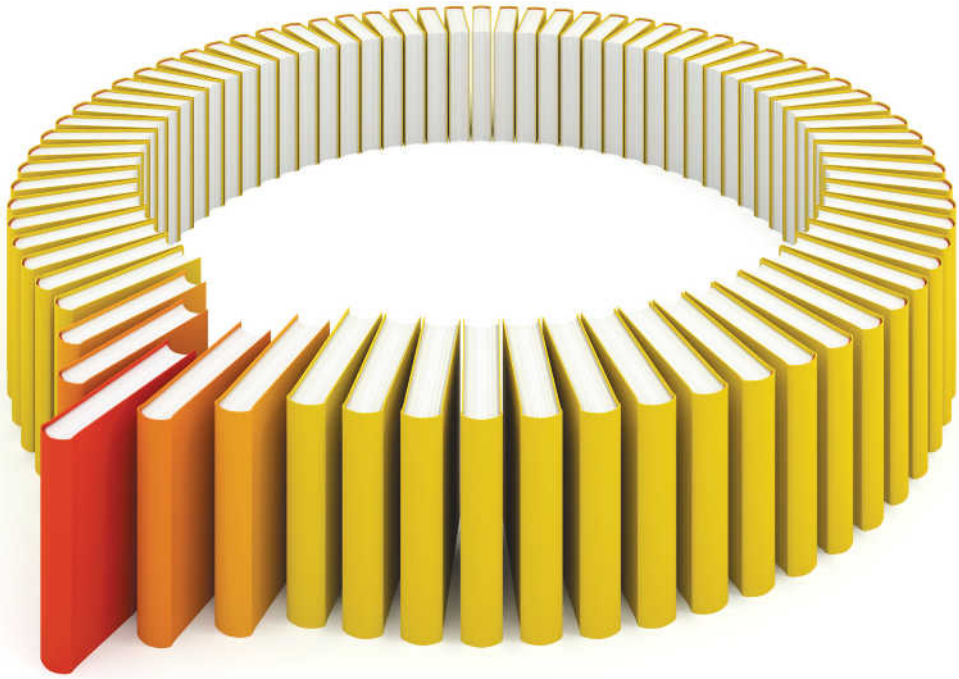
Example

```
options sessref=mysessref;
```

See Also

Statements:

- "CAS Statement" on page 19



Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore
for additional books and resources.