SAS® Cloud Analytic Services 3.5: User’s Guide
# Contents

*About This Book* .................................................. v
*What's New in SAS 9.4 and SAS Viya 3.5 for SAS Cloud Analytic Services* ........ vii

**Chapter 1 / Accessing Data** .................................. 1
  Introduction .................................................. 2
  Terms to Be Familiar With .................................. 2
  Common Tasks for Accessing Data ............................ 3
  Load a Client-Side File ...................................... 4
  Load a Server-Side File ..................................... 5
  Load a Database Table ...................................... 12
  Save an In-Memory Table .................................... 14
  Drop an In-Memory Table ................................... 15
  Delete a File from a Caslib’s Data Source ................. 16
  Data Compression ........................................... 16

**Chapter 2 / CAS LIBNAME Engine** ............................ 21
  CAS LIBNAME Engine Overview ................................ 22
  Getting Started ............................................. 23
  Data Types ................................................. 25
  Use of WHERE with the CAS LIBNAME Engine ............... 34
  Working with Temporary Columns ............................ 35
  Variable Names and Data Set Names in CAS Engine ....... 36
  Troubleshooting CAS LIBNAME Engine Errors ............... 37
  Dictionary ................................................. 37

**Chapter 3 / CASUTIL Procedure** ............................. 69
  Overview: CASUTIL Procedure .............................. 70
  Syntax: CASUTIL Procedure ................................ 71
  Usage: CASUTIL Procedure .................................. 93
  Results: CASUTIL Procedure .................................. 96
  Examples: CASUTIL Procedure .............................. 98

**Chapter 4 / MDSUMMARY Procedure** ....................... 107
  Overview: MDSUMMARY Procedure .......................... 107
  Syntax: MDSUMMARY Procedure ............................. 108
  Usage: MDSUMMARY Procedure .............................. 110
  Results: MDSUMMARY Procedure ............................ 111
  Examples: MDSUMMARY Procedure .......................... 112

**Chapter 5 / Platform Data Sources** ....................... 125
  Data Redundancy ........................................... 125
  Dictionary ................................................. 126

**Chapter 6 / Platform File Types** .......................... 147
  Dictionary ................................................. 147
<table>
<thead>
<tr>
<th>Chapter 7 / Data Connectors</th>
<th>155</th>
</tr>
</thead>
<tbody>
<tr>
<td>Working with SAS Data Connectors</td>
<td>156</td>
</tr>
<tr>
<td>Quick Reference for Data Connector Syntax</td>
<td>157</td>
</tr>
<tr>
<td>Where to Specify Data Connector Options</td>
<td>160</td>
</tr>
<tr>
<td>Using Wildcards for Pattern Matching</td>
<td>161</td>
</tr>
<tr>
<td>Using Data Transfer Modes with Data Connectors for DBMS Data Sources</td>
<td>162</td>
</tr>
<tr>
<td>Dictionary</td>
<td>164</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 8 / Data Types</th>
<th>371</th>
</tr>
</thead>
<tbody>
<tr>
<td>Overview</td>
<td>371</td>
</tr>
<tr>
<td>Character Data</td>
<td>371</td>
</tr>
<tr>
<td>Numeric Data</td>
<td>372</td>
</tr>
<tr>
<td>Data Types for SAS Cloud Analytic Services Table Columns</td>
<td>373</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 9 / Functions</th>
<th>375</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary</td>
<td>375</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 10 / Macro Variables</th>
<th>383</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary</td>
<td>383</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 11 / Session Options</th>
<th>387</th>
</tr>
</thead>
<tbody>
<tr>
<td>Setting Session Options</td>
<td>388</td>
</tr>
<tr>
<td>Session Options by Category</td>
<td>389</td>
</tr>
<tr>
<td>Dictionary</td>
<td>391</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 12 / Statements</th>
<th>409</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary</td>
<td>409</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Chapter 13 / System Options</th>
<th>467</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dictionary</td>
<td>467</td>
</tr>
</tbody>
</table>
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 SAS 9.4 and SAS Viya 3.5 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 CAS statement options connect to the SAS Metadata Server.
- The enhanced CONTENTS statement for the CASUTIL procedure provides information about column index size, compression ratio, and compressed data size.
- New functions provide information about your CAS session.
- New system options control data transfer between SAS and CAS.
- A new macro migrates user-defined formats from SAS catalogs to CAS format libraries.

In SAS Viya 3.5, support is added for Apache Parquet data files.

Enhanced CAS Statement

These 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.
Enhancements to the CASUTIL Procedure

Here are the new statements and options for SAS Viya 3.5.

- The `ALTERTABLE` statement on page 73 enables you to change your table, such as renaming the table.
- The `INDEX` statement on page 76 enables you to create indexes on one or more table variables.
- The `LOAD` statement on page 80 now supports reading data from an external file using a fileref.
- The `LOAD` statement now supports the `WRITETRANSFERSIZE=` on page 84, which specifies the maximum number of bytes that the CAS engine writes when transferring data to the CAS server.
- The `PARTITION` statement on page 86 enables you to partition a table.
- The `PROMOTE` statement on page 88 now supports the `QUIET` option, which does not return a message with status and severity.
- The `UPDATE` statement on page 91 enables you to update rows in a table.

Functions

These functions are new in SAS 9.4M5.

- `CLIBEXIST` on page 375 indicates whether a caslib name exists.
- `GETCASURL` on page 376 returns the CAS Server Monitor URL.
- `GETLCASLIB` on page 377 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.

This function is new in 9.4M6:

- `SESSBUSY` determines whether a CAS session is busy processing actions.
Platform Data Sources

- In **SAS Viya 3.5**, support is added for the USEIAMROLE= option for the S3 platform data source. This option enables you to use the credentials that are defined for your EC2 instance on S3.
- In **SAS Viya 3.5**, Azure Data Lake Storage is a new data source.
- Administrators can add HDFS caslibs that reference different Hadoop clusters. An administrative role must be assumed by the user, and the HADOOPHOME= and HADOOPNAMENODE= arguments are used to identify the Hadoop cluster to use for the caslib. For an example, see “Example 1: Add an HDFS Caslib and Specify HADOOPHOME= and HADOOPNAMENODE=” on page 133.

Platform File Types

In **SAS Viya 3.5**, Parquet is a new file type.

Session Options

These CAS session options are new in **SAS Viya 3.5**:

- The USEGPU session option specifies whether work is offloaded to the Graphics Processing Unit (GPU), if it is available.
- The DEBUGOPT session option specifies the host and port for the SAS Code Debugger in a session.
- The AZURETENANTID session option specifies the tenant ID for connecting to a Microsoft Azure storage system in a session.

System Options

These 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 data is written to the CAS server using the CAS engine, it estimates the number of bytes that are needed to transcode the data to UTF-8. This processing is 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 479 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**

These data connectors are new in the August 2019 release of SAS/ACCESS:
- SAS Data Connector to Google BigQuery
- SAS Data Connector to Snowflake

These data connectors are added with **SAS Viya 3.5**:
- SAS Data Connector to MongoDB
- SAS Data Connector to Salesforce
- SAS Data Connector to ORC

Also, in **SAS Viya 3.5**, support is added for these data connector enhancements:
- Amazon Redshift: bulk unloading (data retrieval). Bulk unloading support includes the new BULKUNLOAD= data connector option.
- Google BigQuery: connection via a proxy URL. Beginning in April 2020, you can use the new PROXY= option to specify a proxy URL.

Also in the April 2020 update, you can authenticate to Google BigQuery using OAuth authentication. The options CLIENT_ID=, CLIENT_SECRET=, and REFRESH_TOKEN= support OAuth authentication. A utility is available to
generate a refresh token if your system administrator does not provide a token for you.

- **MongoDB**: schema storage. Beginning in April 2020, schema representations of MongoDB data are now stored by default to a schema called sas-default-schema. As a best practice, specify a descriptive name for your schema using the `SCHEMA_COLLECTION=` option. To work with a temporary (in memory only) schema, specify `SCHEMA_COLLECTION='$temp$'`.

  Also in the April 2020 update for MongoDB, SAS/ACCESS can import formatted date strings as DATETIME values in SAS.

- **PostgreSQL**: bulk unloading (data retrieval). Bulk unloading support includes these new data connector options: `BULKUNLOAD=`, `BLDELDATFILE=`, `BLDELIMITER=`, `BLESCAPE=`, `BLFORMAT=`, `BLNULL`, `BLPSQLPATH=`, `BLQUOTE=`, and `LDPATH=`.

- **Teradata**: support for the NUMBER data type.

---

**Documentation Enhancements**

Information about preserving SASDAT block sizes is added to "HDFS Data Source" on page 131. Preserving block sizes is needed when backing up or copying SASDAT files in HDFS.

The new “Where to Specify Data Transfer Modes” section clarifies the difference between two separate data transfer mode options.
# Accessing Data

## Introduction

## Terms to Be Familiar With

## Common Tasks for Accessing Data

### Load a Client-Side File

- **Prerequisites**: 4
- **Example**: 4
- **Key Ideas**: 5
- **See Also**: 5

### Load a Server-Side File

- **Prerequisites**: 5
- **Example**: 5
- **Results**: 8
- **Key Ideas**: 11
- **See Also**: 12

### Load a Database Table

- **Prerequisites**: 12
- **Example**: 12
- **Key Ideas**: 13

### Save an In-Memory Table

- **Example**: 14
- **Key Ideas**: 15

### Drop an In-Memory Table

- **Example**: 15
- **Key Ideas**: 15

### Delete a File from a Caslib's Data Source

- **Example**: 16
- **Key Ideas**: 16

### Data Compression

- **Overview of Data Compression**: 16
- **Compressed Tables and the DATA Step**: 17
- **Compressed Tables and the CASUTIL Procedure**: 18
- **Performance Considerations**: 18
- **Interactions**: 19
- **See Also**: 19
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 Data

## Table 1.1  Common Tasks for Accessing Data

<table>
<thead>
<tr>
<th>Task</th>
<th>Sample Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load a SAS data set.</td>
<td><code>proc casutil;</code>&lt;br&gt;<code>load data=libref.member-name</code>&lt;br&gt;<code>casout=&quot;table-name&quot;;</code>&lt;br&gt;<code>run;</code></td>
</tr>
<tr>
<td><strong>TIP</strong>  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.</td>
<td></td>
</tr>
<tr>
<td>Load a client-side data file.</td>
<td><code>proc casutil;</code>&lt;br&gt;<code>load file=&quot;/path/to/file.suffix&quot;</code>&lt;br&gt;<code>casout=&quot;table-name&quot;;</code>&lt;br&gt;<code>run;</code></td>
</tr>
<tr>
<td>List caslibs. This shows you the server-side data sources that SAS Cloud Analytic Services can access.</td>
<td><code>caslib _all_ list;</code></td>
</tr>
<tr>
<td>Add a file-based caslib.</td>
<td><code>caslib data datasource=(srctype=&quot;path&quot;)</code>&lt;br&gt;<code>path=&quot;/data01&quot;;</code></td>
</tr>
<tr>
<td><strong>TIP</strong>  Remember that the specified PATH= must be accessible from the host for the SAS Cloud Analytic Services controller.</td>
<td></td>
</tr>
<tr>
<td>Determine the data files in a caslib that the server can access.</td>
<td><code>proc casutil;</code>&lt;br&gt;<code>list files incaslib=&quot;name&quot;;</code>&lt;br&gt;<code>run;</code></td>
</tr>
<tr>
<td>Load a server-side data file.</td>
<td><code>proc casutil;</code>&lt;br&gt;<code>load casdata=&quot;file-name.suffix&quot;</code>&lt;br&gt;<code>casout=&quot;table-name&quot;;</code>&lt;br&gt;<code>run;</code></td>
</tr>
<tr>
<td>Load a table from database.</td>
<td><code>proc casutil;</code>&lt;br&gt;<code>load casdata=&quot;database-table-name&quot;</code>&lt;br&gt;<code>casout=&quot;table-name&quot;;</code>&lt;br&gt;<code>run;</code></td>
</tr>
<tr>
<td>List in-memory tables.</td>
<td><code>proc casutil;</code>&lt;br&gt;<code>list tables incaslib=&quot;name&quot;;</code>&lt;br&gt;<code>run;</code></td>
</tr>
<tr>
<td>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.</td>
<td><code>libname mycas cas caslib=&quot;name&quot;;</code></td>
</tr>
</tbody>
</table>
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.

```sas
/* 1 */
caslib hps datasource={srctype="hdfs"}
      path="/hps";
proc casutil incaslib="hps" outcaslib="hps";
    load data=sashelp.iris promote;
    contents casdata="iris";
    load file="/data/WorldData.xlsx" casout="worlddata";
    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 3, “CASUTIL Procedure,” on page 69
- “CASLIB Statement” on page 447

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.

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:

5,10,11,10,11,3
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;
The SESSOPTS= option is used with the CASLIB= session option to ensure that the Casuser personal caslib is set as the active caslib.

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

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

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

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 9.

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.

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.

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.

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.

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 10.

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*

<table>
<thead>
<tr>
<th>File Information for catapult.csv in caslib CASUSER(????????).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>------</td>
</tr>
<tr>
<td>catapult.csv</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column Information for catapult.csv in Caslib CASUSER(????????)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
</tr>
<tr>
<td>-------</td>
</tr>
<tr>
<td>5</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>10</td>
</tr>
<tr>
<td>11</td>
</tr>
<tr>
<td>3</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>File Size</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>catapultraw.sashdat</td>
<td>-rwxr-xr-x</td>
<td></td>
<td></td>
<td>24.2KB</td>
<td>01Feb2016:19:27:28</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Label</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>turns</td>
<td>Number of turns</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>first_ft</td>
<td>Feet for first try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>first_in</td>
<td>Inches for first try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>second_ft</td>
<td>Feet for second try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>second_in</td>
<td>Inches for second try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>third_ft</td>
<td>Feet for third try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>third_in</td>
<td>Inches for third try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>NS encoding</th>
<th>Created</th>
<th>Last Modified</th>
<th>Promoted Table</th>
<th>Dupliacted Rows</th>
<th>View</th>
<th>Source Name</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATAPULT</td>
<td>5</td>
<td>10</td>
<td>utf-8</td>
<td>01Feb2016:19:28:25</td>
<td>01Feb2016:19:27:56</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>_T_1AAD1599_FFF36542BBA8</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>Number of Blocks</th>
<th>Active Blocks</th>
<th>Rows</th>
<th>Fixed Data size</th>
<th>Variable Data size</th>
<th>Blocks Mapped</th>
<th>Memory Mapped</th>
<th>Blocked Unmapped</th>
<th>Memory Unmapped</th>
<th>Blocks Allocated</th>
<th>Memory Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>5</td>
<td>5</td>
<td>5</td>
<td>400</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>400</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Label</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>turns</td>
<td>Number of turns</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>first_ft</td>
<td>Feet for first try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>first_in</td>
<td>Inches for first try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>second_ft</td>
<td>Feet for second try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>second_in</td>
<td>Inches for second try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>third_ft</td>
<td>Feet for third try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>third_in</td>
<td>Inches for third try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>first</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>second</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>third</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>
The CASUTIL Procedure

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.
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 file name, 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.
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.

```sas
* Load a database table from Oracle.
* caslib orilib datasource=(
* srctype="oracle",
* uid="DBUSER",
* pwd="secret",
* path="/dbserver.example.com:1521/dbname",
* schema="DBUSER"
*);

proc casutil;
list files;
* Load the table and keep it in the same caslib.
* load casdata="sales" casout="sales" promote
```

See Also

- Chapter 3, “CASUTIL Procedure,” on page 69
- “CASLIB Statement” on page 447
- SAS Cloud Analytic Services: DATA Step Programming
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.

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

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.

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

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.

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 447 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 “Data Connectors” on page 155.
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.

```plaintext
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;

casil orsales datasource=({
  srcype="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 file name 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 SASDAT 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 “Data Connectors” on page 155.

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.

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

Key Ideas

- Include the file name 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 1.1  Creating a Compressed Table with a DATA Step**

```sas
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 1.2  Appending Rows to a Compressed Table**

```sas
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 1.3  Creating a Partitioned and Compressed Table**

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

```sas
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.

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

This example uses the COMPRESS option on page 54 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”
# CAS LIBNAME Engine

**CAS LIBNAME Engine Overview** .................................................. 22

**Getting Started** ........................................................................... 23

**Data Types** .................................................................................. 25
  - Data Type Definitions ................................................................. 25
  - Support for Implicit Declaration of Data Types ......................... 25
  - VARCHAR Data Type ................................................................. 25

**Use of WHERE with the CAS LIBNAME Engine** ............................. 34

**Working with Temporary Columns** ............................................... 35

**Variable Names and Data Set Names in CAS Engine** .................... 36

**Troubleshooting CAS LIBNAME Engine Errors** .......................... 37
  - The Call to Cloud Analytic Services Failed ............................... 37

**Dictionary** .................................................................................. 37
  - CAS LIBNAME Statement .......................................................... 37
  - CASLIB= LIBNAME Statement Option ....................................... 40
  - COMPRESS= LIBNAME Statement Option ................................... 41
  - DATALIMIT= LIBNAME Statement Option ................................... 42
  - HOST= LIBNAME Statement Option ............................................ 43
  - NCHARMULTIPLIER= LIBNAME Statement Option ...................... 44
  - PORT= LIBNAME Statement Option ............................................ 44
  - MAXTABLEMEM= LIBNAME Statement Option ............................. 45
  - READTRANSFERSIZE= LIBNAME Statement Option ....................... 46
  - SESSREF= LIBNAME Statement Option ....................................... 47
  - TAG= LIBNAME Statement Option ............................................. 48
  - TRANSCODE_FAIL= LIBNAME Statement Option ......................... 49
  - UUID= LIBNAME Statement Option ............................................ 49
  - UUIDMAC= LIBNAME Statement Option ....................................... 50
  - WRITETRANSFERSIZE= LIBNAME Statement Option ...................... 50
  - APPEND= Data Set Option ......................................................... 51
  - CASLIB= Data Set Option ........................................................ 53
  - COMPRESS= Data Set Option .................................................... 54
  - COPIES= Data Set Option ........................................................ 54
  - DATALIMIT= Data Set Option .................................................... 55
  - DUPLICATE= Data Set Option ..................................................... 56
  - MAXTABLEMEM= Data Set Option ............................................... 57
  - NCHARMULTIPLIER= Data Set Option ........................................ 58
  - ONDEMAND= Data Set Option ................................................... 58
  - ORDERBY= Data Set Option ....................................................... 59
  - PARTITION= Data Set Option ..................................................... 60
  - PROMOTE= Data Set Option ....................................................... 62
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. 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.

The best practice for connecting to CAS is to first start a CAS session, and then specify the SESSREF=. See the below table for more information about how the LIBNAME Engine connects to CAS.

Table 2.1 How the LIBNAME Engine Connects to CAS

<table>
<thead>
<tr>
<th>Number of Existing CAS Sessions</th>
<th>Specified SESSREF= Statement Option?</th>
<th>Connection Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>None</td>
<td>No</td>
<td>The CAS engine starts a new session when you specify the LIBNAME statement. 1</td>
</tr>
<tr>
<td>One</td>
<td>No</td>
<td>The libref is associated with the session in the SESSREF= statement option. The system option is updated with the name of the most recently started CAS session.</td>
</tr>
<tr>
<td>One</td>
<td>Yes</td>
<td>The libref is associated with the session that you specify in the SESSREF= statement option.</td>
</tr>
<tr>
<td>More than one</td>
<td>No</td>
<td>The libref is associated with the session in the SESSREF= statement option. The system option is updated with the name of the most recently started CAS session.</td>
</tr>
<tr>
<td>More than one</td>
<td>Yes</td>
<td>The libref is associated with the session that is specified in the SESSREF= statement option. Otherwise, the libref is associated with a session according to the SESSREF= system option. You can view the setting with the following code:</td>
</tr>
</tbody>
</table>

1 The CAS engine uses the CASHOST and CASPORT system options to make the connection, and the SESSREF system option to name the session.
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:

```sas
Cas casauto terminate;
```

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

---

**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 assign 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.

```sas
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. In-memory DATA steps 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:

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Example</th>
<th>Missing Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed Length Character</td>
<td>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</td>
<td>data mycas.greets; length greet1 $16; greet1 = &quot;Good day&quot;; run; cas data greets;</td>
<td>all blanks</td>
</tr>
<tr>
<td>VARCHAR(( n ))</td>
<td>Stores a varying-length character string. ( n ) is the maximum number of characters to store. Range 1 - 536,870,911 characters (UTF-8 encoding)</td>
<td>data mycas.info; length lastname varchar(40); lastname=&quot;Adams&quot;; run;</td>
<td>all blanks, or zero length</td>
</tr>
</tbody>
</table>
DOUBLE (numeric) Stores a numeric value, including dates and times, as a floating-point number.
Range: 8 bytes for the CAS Engine\(^1\)

Any one of the following forms:

- .A-.Z
- .

1 If a shorter length is specified, then 8 bytes are used and a note is printed to the SAS log.
2 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 that are supported by the SAS V9 Engine, see “SAS Variables” in SAS Language Reference: Concepts.

SAS Cloud Analytic Services does not support the LENGTH statement for numeric data types and does not support DOUBLE columns that use less than 8 bytes. If you use the LENGTH statement to store numeric data with less than 8 bytes and then load that data into CAS, the server stores the data with 8 bytes. If you have several numeric columns that use less than 8 bytes, the in-memory table size can be much larger than what is required for a .sas7bdat file.

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:

```sas
libname mycas cas;
data mycas.datatypes;
x=1;
y='hello';
run;
```
PROC CONTENTS DATA MYCAS.DATATYPES;
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-name) VARCHAR(length|*)
ARRAY array-name[N] VARCHAR(length|*)
ARRAY array-name[*] 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 that can be 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 VARCHAR Variables” on page 31

* specifies that SAS uses 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 32,767 bytes.

length xyz varchar(*);

Uninitialized or missing 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 VARCHAR Variables” on page 31
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');

Requirements requires a CAS engine libref on the output table

requires additional storage space. See VARCHAR Variable Storage. on page 33

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 in a column. 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, in which length is fixed and measured in bytes.

For example, a VARCHAR(100) can store up to 100 characters, but the actual storage used in any given row depends on the lengths of the individual values in the row and column.

For example, if a VARCHAR variable, vc2, is defined as a VARCHAR(100) variable, this means that it can store up to 100 characters. But because the value “abc” contains only 3 characters and each character uses one byte, only 3 bytes of memory are allocated for the value in that row. A fixed-length CHAR column, on the other hand, takes up the defined number of bytes regardless of the actual value. In the case of the example, the fixed-width CHAR with a defined length of 100 would use all 100 bytes even though the actual value “abc” needs only 3 bytes.
Example: Create a VARCHAR Using the LENGTH Statement

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

Example: Create a VARCHAR Variable Using the ARRAY Statement

```sas
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(n)** – 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:

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

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

```sql
    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. VARCHAR type variables are not supported by the Base SAS V9 engine, so they are automatically converted from VARCHAR types to fixed-length CHAR types.

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.

```sql
    data mycas.new;
    length vc varchar(40);
    vc = '5000';
    num = input(vc,8.);
    run;
    proc contents data=mycas.new; run;
```
The CONTENTS procedure provides metadata about a CAS table including detailed information about the variables in the table.

When a table contains a VARCHAR variable, PROC CONTENTS displays extra length information about the variable that is not normally displayed when a VARCHAR is not present in the table:

- **Len** displays the length specified by the user when the variable is declared. The defined length is given in 2 sub-columns, Chars and Bytes:
  - **Chars** displays the *defined* length in characters, or the length that the user specifies in the VARCHAR(n | *) statement when the variable is declared.
  - **Bytes** displays a length in bytes that SAS calculates based on the defined length and the SAS session encoding. SAS calculates the length by multiplying the defined length by the largest possible number of bytes required to store any one character in the character set encoding. The defined length is the value defined by the user in the VARCHAR(n) statement.

  For example, if your SAS session encoding is UTF-8, then the **Len Bytes** value is calculated as \((n \times 4)\), where \(n\) is the length defined in the VARCHAR(n | *) statement and 4 is the largest possible number of bytes required to store any character in the UTF-8 character set. Similarly, if the local SAS session encoding is Latin1, then the **Len Bytes** value is calculated as \((n \times 1)\), where \(n\) is the length defined in the VARCHAR(n | *) statement and 1 is the largest number of bytes required to store any character in the Latin1 character set.

- **Max Bytes Used** displays the number of bytes that are used to store the longest string value in the column.

```sas
data mycas.new;
  length vc1 varchar(*) /* defined length of vc1, shown in "Len Chars" column */
     vc2 varchar(100) /* defined length of vc2, shown in "Len Chars" column */
     vc3 varchar(10); /* defined length of vc3, shown in "Len Chars" column */
  vc2 = "123456789";  /* actual bytes used for vc2, shown in Max Bytes Used column */
  vc3 = "abc";        /* actual bytes used for vc3, shown in Max Bytes Used column */
run;
```
If you specify VARCHAR(*), then SAS defines the length as the maximum allowable length.

**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 \((n \times 1)\). \(n\) 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 \((n \times 2)\). \(n\) 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 \((n \times 4)\). \(n\) 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.

**VARCHAR Variable Storage**

When CHAR values are stored, they are right-padded with spaces to the specified length. VARCHAR values are not padded. When VARCHAR values are stored, they are stored with descriptor information in each row that takes space along with the data value. The descriptor is a 16-byte prefix that is stored with the data value in each row and that contains information about the length of the data value.

**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>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIB</td>
<td>You cannot use the ATTRIB statement to create VARCHAR variables.</td>
</tr>
<tr>
<td>BY statement</td>
<td>The BY statement uses a fixed width for VARCHAR variables. Using a VARCHAR(*) type in the BY statement might cause unexpected results.</td>
</tr>
<tr>
<td>Formats</td>
<td>The width of VARCHAR formats is measured in bytes rather than characters.</td>
</tr>
<tr>
<td>Functions</td>
<td>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 &quot;Index CHAR and VARCHAR Character Strings&quot; in <em>SAS Cloud Analytic Services: DATA Step Programming</em> for a related example.</td>
</tr>
<tr>
<td>KEY= on SET and MODIFY statements</td>
<td>VARCHAR variables are not supported by the KEY= option in either the SET or MODIFY statements.</td>
</tr>
<tr>
<td>PUT statement (to ODS output)</td>
<td>VARCHAR variables are not supported with the PUT statement when the DATA step writes output using ODS.</td>
</tr>
</tbody>
</table>
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(*);
vcl = "a string that is 32000 bytes long";
vc2 = vc1 || "another string that is 32000 bytes long";
```

---

## Use of WHERE with the CAS LIBNAME Engine

WHERE processing enables you to conditionally select a subset of rows so that SAS 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 attempts to resolve the WHERE expression on the CAS server so that less data is transferred from the server to SAS. Performance is often improved with large tables when the WHERE expression qualifies only a relatively small subset.

There are WHERE expressions that cannot be processed 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. 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.

```sas
 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 The example shows how to use the FILENAME statement with the TEMP option to create a temporary file. The expressions for each column are written to the temporary file in the DATA step that follows. You could specify a path for the file instead of using the TEMP option.
A DATA _NULL_ statement 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.

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 58 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.

---

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:

```sas
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.
Troubleshooting CAS LIBNAME Engine Errors

The Call to Cloud Analytic Services Failed

```plaintext
ERROR: The call to Cloud Analytic Services failed.
ERROR: The action cannot be processed while an action is active.
```

<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
</table>
| A procedure or DATA step with more than one output uses one CAS LIBNAME engine libref for input and the same libref for output, such as output created using the OUT= option or a TABLES statement. | ▪️ Use multiple DATA steps. For best results, output a single table and use the same CAS engine libref that was used for input.  
▪️ Use multiple PROC steps.  
▪️ Create output data with a different engine. |

There are several scenarios that cause the error. One example of how the error occurs is when the CAS LIBNAME engine is used for more than one output table. See the following code sample that demonstrates this specific scenario.

```
libname mycas cas wts=1k;
data mycas.one mycas.two;
  set sashelp.cars;
run;
```

In the above code example, the WRITETRANSFERSIZE= option is set to an artificially small value to trigger the error. However, the solution is to use multiple DATA steps in the program rather than increase the transfer size.

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 active caslib, 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" | ("controller-host-name"<"backup-controller-host-name ">) PORT=network-port-number <options>;`

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

Required Arguments

CAS  
specifies the CAS LIBNAME engine.

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.

LIBNAME Options Supported by the CAS Engine

The following table contains a list of optional LIBNAME statement options supported by the CAS LIBNAME engine.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>CASLIB= on page 40</td>
<td>Specifies the name of the caslib to use for engine operations in the LIBNAME statement.</td>
<td>The active caslib</td>
</tr>
<tr>
<td>COMPRESS= on page 41</td>
<td>Requests that the CAS table is compressed.</td>
<td>NO</td>
</tr>
<tr>
<td>DATALIMIT= on page 42</td>
<td>Specifies the maximum number of bytes of data that can be transferred between the CAS server and SAS during a single-table read.</td>
<td>100M</td>
</tr>
<tr>
<td>HOST= on page 43</td>
<td>Specifies the host name of the CAS controller.</td>
<td>The host name is determined by the session unless the SESSREF= option and the PORT= option are specified.</td>
</tr>
<tr>
<td>MAXTABLEME M= on page 45</td>
<td>Specifies the maximum amount of memory, in bytes, to allocate for an in-memory table.</td>
<td>16M</td>
</tr>
<tr>
<td>Option</td>
<td>Description</td>
<td>Default</td>
</tr>
<tr>
<td>------------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><strong>NCHARMULTIPLIER= on page 44</strong></td>
<td>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.</td>
<td>1 for SBCS environments 1.5 for DBCS environments</td>
</tr>
<tr>
<td><strong>PORT= on page 44</strong></td>
<td>Specifies the network port number on which the controller listens for client connections.</td>
<td>The port is determined by session unless the SESSREF= option and the PORT= option are specified.</td>
</tr>
<tr>
<td><strong>READTRANSFERSIZE= on page 46</strong></td>
<td>Specifies the maximum number of bytes that the CAS engine reads when transferring data from the CAS server.</td>
<td>500MB</td>
</tr>
<tr>
<td><strong>SESSREF= on page 47</strong></td>
<td>Specifies the name of the CAS session to which you want to connect.</td>
<td></td>
</tr>
<tr>
<td><strong>TAG= on page 48</strong></td>
<td>Specifies a multi-level prefix for a CAS table that is used to identify the table to both the CAS server and SAS.</td>
<td></td>
</tr>
<tr>
<td><strong>TRANSCODE_FAIL= on page 49</strong></td>
<td>Specifies how the engine handles transcoding errors.</td>
<td>ERROR</td>
</tr>
<tr>
<td><strong>UUID= on page 49</strong></td>
<td>Connects the CAS engine to the CAS session that is identified in the UUID.</td>
<td></td>
</tr>
<tr>
<td><strong>UUIDMAC= on page 50</strong></td>
<td>Specifies a macro variable that connects a subsequent SAS step to a session that was created or modified by the CAS LIBNAME engine.</td>
<td>If you do not specify this option, the UUID is stored in the <em>IOCASUUID</em> macro variable value.</td>
</tr>
<tr>
<td><strong>Writtetransfersize= on page 50</strong></td>
<td>Specifies the maximum number of bytes that the CAS engine writes when transferring data to the CAS server.</td>
<td>512KB</td>
</tr>
</tbody>
</table>
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.

```sas
  cas casauto host="cloud.example.com" port=5570; /*1*/  
  libname mycas cas; /*2*/  
  
  data mycas.cars;  
    set sashelp.cars; /*3*/  
    run;  
  
  data mycas.somecars;  
    set mycas.cars; /*4*/  
    by make model;  
    keep make model weight origin;  
    run;  
  
  proc print data=mycas.cars2; /*5*/  
   run;  
```

1 Start a CAS session.
2 Assign a libref. The engine finds the CASAUTO session from the SESSREF system option.
3 Use the engine to transfer data from SAS to the server.
4 Run Data step in CAS to subset data.
5 Print output from in-memory table.

See Also

- CASLIB statement,
- COMPRESS= LIBNAME option

CASLIB= LIBNAME Statement Option

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

Valid in: CAS LIBNAME Statement
Category: Data Access
Default: the active caslib
Note: The CASLIB= LIBNAME option accepts a caslib reference name as a string, quoted or not quoted. Caslib reference names do not support named literals.
Syntax

**CASLIB=**<br>
**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 Statement Option**

Requests that the CAS table is compressed.

Valid in:  
CAS LIBNAME Statement

Category:  
Data Access

Default:  
NO

Restriction:  
This option applies to output, in-memory tables only.

Syntax

**COMPRESS=**<br>
**YES | NO**
Required Arguments

**YES**
specifies to compress output, in-memory tables.

**NO**
specifies that compression is not used when creating output, in-memory tables.

Comparisons

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

See Also

“COMPRESS= Data Set Option”

---

**DATALIMIT= LIBNAME Statement 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: CAS LIBNAME Statement

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 | integerK | integerM | integerG | ALL
```

Optional Arguments

- **integer**
specifies the maximum number of bytes to read.

- **integerK**
specifies the maximum number of kilobytes to read.

- **integerM**
specifies the maximum number of megabytes to read.

- **integerG**
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 Statement Option”
- “READTRANSFERSIZE= Data Set Option”
- “WRITETRANSFERSIZE= LIBNAME Statement Option”
- “WRITETRANSFERSIZE= Data Set Option”

HOST= LIBNAME Statement Option

specifies the host name of the CAS controller.

Valid in: CAS LIBNAME Statement
Category: Data Access

Syntax

Form 1: HOST="controller-host-name" | ("controller-host-name"="backup-controller-host-name")

Required Argument

"controller-host-name"

specifies the host name of the CAS controller.

Optional Argument

"backup-controller-host-name"

specifies the host name of the CAS backup controller.

Details

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
See Also

For more information about backup controllers, see Distributed Server: Controller Fault Tolerance.

NCHARMULTIPLIER= LIBNAME Statement 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: CAS LIBNAME Statement
Category: Data Access
Defaults: 1 for SBCS encoding
1.5 for DBCS encoding
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 ≤ 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”

PORT= LIBNAME Statement Option

Specifies the network port number on which the controller listens for client connections.

Valid in: CAS LIBNAME Statement
Category: Data Access
Default: Port 0 is assumed if the SESSREF= option and the PORT= option are not specified.
Note: Otherwise, the engine checks for a value in the CASPORT= system option and then in the _CASPORT_ macro variable.

**Syntax**

```
PORT=integer
```

Required Argument

`integer` specifies the network port number.

---

**MAXTABLEMEM= LIBNAME Statement 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 | integerK | integerM | integerG
```

Required Arguments

`integer` specifies the total number of bytes to allocate.

`integerK` specifies the total number of kilobytes to allocate.

`integerM` specifies the total number of megabytes to allocate.

`integerG` 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 Statement Option

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

Valid in: CAS LIBNAME Statement
Category: Data Access
Alias: RTS
Default: 500MB
Restriction: This option affects only Read access.

Syntax

```
READTRANSFERSIZE=integer | integerK | integerM | integerG
```

Required Arguments

- `integer` specifies the total number of bytes to read.
- `integerK` specifies the total number of kilobytes to read.
- `integerM` specifies the total number of megabytes to read.
- `integerG` 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 Statement Option” on page 42
- “DATALIMIT= Data Set Option” on page 55
- “READTRANSFERSIZE= Data Set Option” on page 63
- “WRITETRANSFERSIZE= LIBNAME Statement Option” on page 50
- “WRITETRANSFERSIZE= Data Set Option” on page 66

SESSREF= LIBNAME Statement Option

Specifies the name of the CAS session to which you want to connect.

Valid in: CAS LIBNAME Statement
Category: Data Access
Tip: Using this option is a preferred alternative to HOST= and PORT=

Syntax

```
SESSREF=session-name
```

Required Argument

```
session-name
```

specifies the name of the CAS session to which you want to connect.
TAG= LIBNAME Statement 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: CAS LIBNAME Statement
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 65
TRANSCODE_FAIL= LIBNAME Statement Option

Specifies how the engine handles transcoding errors.
Valid in: CAS LIBNAME Statement, 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 66

UUID= LIBNAME Statement Option

Connects the CAS engine to the CAS session that is identified in the UUID.
Valid in: CAS LIBNAME Statement
Category: Data Access
Tip: The “SESSREF= LIBNAME Statement Option” on page 47 is the preferred option for connecting to a CAS session.
Example: This example starts a session with the CAS statement and saves the UUID in the MyUUID macro variable. A 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";

< Perform some additional tasks. >
libname mycas cas uuid="&MyUUID";
data mycas.female;
set sashelp.class;
  where sex='F';
run;

Syntax

**UUID=“session-uuid”**

Required Argument

**“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.

**UUIDMAC= LIBNAME Statement Option**

Specifies a macro variable that stores a UUID for any session created by the engine.

**Valid in:** CAS LIBNAME Statement

**Category:** Data Access

**Default:** If you do not specify this option, the UUID is stored in the _IOCASUUID_ macro variable value.

**Syntax**

**UUIDMAC=“macro-variable-name”**

**Required Argument**

**“macro-variable-name”**

specifies a name or string into which the engine saves the UUID of the current CAS session.

**WRITETRANSFERSIZE= LIBNAME Statement Option**

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

**Valid in:** CAS LIBNAME Statement

**Category:** Data Access

**Alias:** WTS

**Default:** 512KB

**Restriction:** This option affects only Write access.
Syntax

WRITETRANSFERSIZE=integer | integerK | integerM | integerG

Required Arguments

integer
  specifies the total number of bytes to read.

integerK
  specifies the total number of kilobytes to read.

integerM
  specifies the total number of megabytes to read.

integerG
  specifies the total number of gigabytes to read.

Comparisons

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

See Also

n “DATALIMIT= LIBNAME Statement Option” on page 42
n “DATALIMIT= Data Set Option” on page 55
n “READTRANSFERSIZE= LIBNAME Statement Option” on page 46
n “READTRANSFERSIZE= Data Set Option” on page 63
n “WRITETRANSFERSIZE= LIBNAME Statement Option” on page 50
n “WRITETRANSFERSIZE= Data Set Option” on page 66

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

Restrictions: An error results if you set both PROMOTE= YES and APPEND= YES. You must drop or delete a global-scope base table before promoting a replacement.

If the target table for an append is partitioned, then the table that you want to append must be partitioned by the same columns. For more information, see “Partitioning and Ordering” in SAS Cloud Analytic Services: Fundamentals.

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 in-memory table. If the in-memory table is global in scope, then APPEND=YES automatically promotes the new rows. If the in-memory table is a session table, APPEND does not automatically promote the result table.

Note: An error can occur if the lengths of the input observations differ from the character length on the in-memory table.

NO
specifies that the rows are not appended to the in-memory table.

FORCE
uses the length and type of the in-memory table to determine the length and type of newly appended observations. Truncation occurs to CHAR columns when the length of the input data exceeds the length of the output table.

When you force append numeric data to an output table with a data type of integer, numeric values that overflow or underflow the integer limits are converted to missing.

Example: Append to a CAS Table That Contains Variables with Different Lengths

libname mycas cas;

data mycas.table;
length x $5; /*1*/
x="hello";
run;

data dataset;
length x $7; /*2*/
x="goodbye";
run;

data mycas.table(append=force); /*3*/
set dataset;
run;

proc print data=mycas.table;
run;

1 The length of the data in the CAS table is 5 bytes.
2 The length of the data in the temporary SAS data set is 7 bytes.
The data from the DATA step output is force appended to the CAS table. Since the length of the data in the CAS table is less than the length of the data in the DATA step output, truncation occurs.

<table>
<thead>
<tr>
<th>Obs</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>good</td>
</tr>
<tr>
<td>2</td>
<td>hello</td>
</tr>
</tbody>
</table>

See Also

“Filter and Append Rows to a Promoted Table” in *SAS Cloud Analytic Services: DATA Step Programming*

---

**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
- **Note:** The CASLIB= option accepts a caslib reference name as a string, quoted or not quoted. Caslib reference names do not support named literals.

**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 447
- “CASLIB= LIBNAME Statement Option” on page 40
- “CASLIB= Session Option” on page 392
- “CASLIB= System Option” on page 472
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 Statement 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=**\textit{integer}

Required Argument

\textit{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. \texttt{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=**\textit{integer | integerK | integerM | integerG | ALL}

Required Arguments

\textit{integer}

specifies the maximum number of bytes to read.

\textit{integerK}

specifies the maximum number of kilobytes to read.

\textit{integerM}

specifies the maximum number of megabytes to read.

\textit{integerG}

specifies the maximum number of gigabytes to read.

\textit{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 Statement Option”
- “READTRANSFERSIZE= LIBNAME Statement Option”
- “READTRANSFERSIZE= Data Set Option”
- “WRITETRANSFERSIZE= LIBNAME Statement 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 nodes.

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

\[ \text{MAXTABLEMEM=} \text{integer} | \text{integerK} | \text{integerM} | \text{integerG} \]

Required Arguments

- \text{integer} specifies the total number of bytes to allocate.
- \text{integerK} specifies the total number of kilobytes to allocate.
- \text{integerM} specifies the total number of megabytes to allocate.
- \text{integerG} 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 encoding
1.5 for DBCS encoding
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 ≤ 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 Statement Option”
- "Migrating Data to UTF-8 Encoding" in Migrating Data to UTF-8 for SAS Viya

ONDEMAND= Data Set Option

Specifies how to evaluate temporary computed columns.

Valid in: DATA step and PROC step
Category: Data Access
Default: NO
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 35

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.
### 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.

```sas
   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 59

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

Restriction: An error results if you set both PROMOTE= YES and APPEND= YES. You must drop or delete a global-scope base table before promoting a replacement.

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 | integerK | integerM | integerG
```

Required Arguments

- `integer` specifies the total number of bytes to read.
- `integerK` specifies the total number of kilobytes to read.
- `integerM` specifies the total number of megabytes to read.
- `integerG` 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 35
**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 Statement Option" on page 48

---

**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 35
**TRANSCODE_FAIL= Data Set Option**

Specifies how the engine handles transcoding errors.

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA step and PROC step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category:</td>
<td>Data Access</td>
</tr>
<tr>
<td>Default:</td>
<td>Error</td>
</tr>
</tbody>
</table>

**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 Statement Option`” on page 49

**WRITETRANSFERSIZE= Data Set Option**

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

<table>
<thead>
<tr>
<th>Valid in:</th>
<th>DATA step and PROC step</th>
</tr>
</thead>
<tbody>
<tr>
<td>Category:</td>
<td>Data Access</td>
</tr>
<tr>
<td>Alias:</td>
<td>WTS</td>
</tr>
<tr>
<td>Default:</td>
<td>512K</td>
</tr>
<tr>
<td>Restriction:</td>
<td>This option only affects Write access.</td>
</tr>
</tbody>
</table>
Syntax

**WRITETRANSFERSIZE=**\textit{integer | integerK | integerM | integerG}

Required Arguments

\textit{integer}
- specifies the total number of bytes to read.

\textit{integerK}
- specifies the total number of kilobytes to read.

\textit{integerM}
- specifies the total number of megabytes to read.

\textit{integerG}
- specifies the total number of gigabytes to read.

Comparisons

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

See Also

- “**DATALIMIT=** LIBNAME Statement Option” on page 42
- “**DATALIMIT=** Data Set Option” on page 55
- “**READTRANSFERSIZE=** LIBNAME Statement Option” on page 46
- “**READTRANSFERSIZE=** Data Set Option” on page 63
- “**WRITETRANSFERSIZE=** LIBNAME Statement Option” on page 50
CASUTIL Procedure

Overview: CASUTIL Procedure .................................................. 69
  What Does the CASUTIL Procedure Do? .............................. 70
  Terminology ................................................................. 70

Syntax: CASUTIL Procedure ................................................... 71
  PROC CASUTIL Statement ................................................. 72
  ALTERTABLE Statement .................................................... 73
  CONTENTS Statement ...................................................... 74
  DELETESOURCE Statement ............................................... 75
  DROPTABLE Statement .................................................... 76
  INDEX Statement .......................................................... 76
  LIST Statement ............................................................ 79
  LOAD Statement ........................................................... 80
  PARTITION Statement ..................................................... 86
  PROMOTE Statement ....................................................... 88
  SAVE Statement ........................................................... 89
  UPDATE Statement ......................................................... 91

Usage: CASUTIL Procedure ................................................... 93
  Enclose Values in Quotation Marks ....................................... 93
  Subdirectories and File Name Matching .................................. 93
  Limitations and Restrictions ............................................. 95

Results: CASUTIL Procedure ................................................ 96
  Procedure Output .......................................................... 96
  ODS Table Names ........................................................ 98

Examples: CASUTIL Procedure ............................................. 98
  Example 1: Load a CSV File into CAS .................................. 98
  Example 2: Append Rows to an In-Memory Table ..................... 100
  Example 3: Promote a Table .............................................. 101
  Example 4: Saving and Loading Encrypted SASHDAT Files .......... 103
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 <options>;
  ALERTABLE CASDATA="table-name" <INCASLIB="caslib"> <LABEL="label"> <RENAME | CASOUT="table-name"> <COLUMNS={column-options-1 <, {column-options-n}>}> <COLUMNORDER="variable-name","variable-name-n"> > <DROP="variable-name","variable-name-n"> > <KEEP="variable-name","variable-name-n"> > CONTENTS CASDATA="table-name" <INCASLIB="caslib"> <options>; 
  DELETESOURCE CASDATA="file-name" <INCASLIB="caslib"> <DATASOURCEOPTIONS=(data-source-options)> <DELETEOPTIONS=(delete-options)> <QUIET>; 
  DROPTABLE CASDATA="table-name" <INCASLIB="caslib"> <QUIET>; 
  INDEX CASDATA="table-name" <INCASLIB="caslib"> <LABEL="label"> <COMPRESS> <PROMOTE> <REPLACE> <REPLICATION|COPIES=integer> <INDEXVARS=((variable-name-1), (variable-name-n))>; 
  LIST <FILES | TABLES> <options>;
  LOAD CASDATA="file-name" | DATA=SAS-data-set | FILE="SAS-file" <options>;
  PARTITION CASDATA="table-name" <INCASLIB="caslib"> <LABEL="label"> <COMPRESS> <PROMOTE> <REPLACE> <REPLICATION|COPIES=integer>
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 <options>;

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.
The ALTERTABLE statement enables you to change your table. For example, you can rename the table, change labels and formats, drop columns, and so on.

Syntax

```
ALTERTABLE CASDATA="table-name" <INCASLIB="caslib" | LABEL="label">
  <RENAME | CASOUT="table-name">
  <COLUMNS={column-options-1 ,,column-options-n}>}
  <COLUMNORDER={"variable-name" ,,"variable-name-n">}
  <DROP={"variable-name" ,,"variable-name-n"}>
  <KEEP={"variable-name" ,,"variable-name-n"}>
```

Required Argument

- **CASDATA="table-name"**
  - specifies the name of the in-memory table to change.

Optional Arguments

- **INCASLIB="caslib"**
  - specifies the caslib with the in-memory table to change. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

- **LABEL="label"**
  - specifies a new label for the table.

- **RENAME | CASOUT="table-name"**
  - specifies a new name for the table.

- **COLUMNS={column-options-1 ,,column-options-n}>**
  - specifies how to alter columns. After you specify the column name, you can drop a column or assign a new format, label, or name. Following are the column options:
    - **NAME="variable-name"**
    - **DROP=TRUE | FALSE**
    - **FORMAT="format-name"**
    - **LABEL="label"**
    - **RENAME="table-name"**

- **COLUMNORDER={"variable-name" ,,"variable-name-n"}>**
  - specifies the order of the columns.

- **DROP={"variable-name" ,,"variable-name-n"}>**
  - specifies the variables to drop from the table.

- **KEEP={"variable-name" ,,"variable-name-n"}>**
  - specifies the variables to keep in the table.
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 101
“Example 4: Saving and Loading Encrypted SASHDAT Files” on page 103

Syntax

```plaintext
CONTENTS CASDATA="table-name" <INCASLIB="caslib"> <options>;
```

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 file name 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 82.

`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 “Platform Data Sources” on page 125 and “Data Connectors” on page 155.

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 file name 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 84.

Alias `VARLIST=`
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<br>CASDATA="file-name" <INCASLIB="caslib" ><br><DATASOURCEOPTIONS = {data-source-options}> <DELETEOPTIONS = {delete-options}> <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.

**DATASOURCEOPTIONS = {data-source-options}**

specifies data source options.

Alias  dataSource, options

See  dataSourceOptions

**DELETEOPTIONS={delete-options}**

specifies the settings for deleting a file from a data source.

Note  The value that you specify for fileType determines the other parameters that apply.

See  deleteOptions

**QUIET=TRUE | FALSE**

suppresses error messages and avoids setting the SYSERR automatic macro variable when the specified table or file is not found.

When QUIET is set to True, attempting to delete a file or table that does not exist returns an OK status and severity. When set to False, attempting to delete a file or table that does not exist returns an error.
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 103

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.

INDEX Statement

The INDEX statement enables you to create indexes on one or more table variables.

Syntax

```
INDEX CASDATA="table-name"
   <INCASLIB="caslib">
   <VARS={ {casinvardesc-1}, {casinvardesc-n} }> 
   <WHERE="where-expression <logical-operator where-expression">>
   <IMPORTOPTIONS=(FILETYPE=file-type <file-type-options>)>
   <DATASOURCEOPTIONS=(data-source-options)>
   <CASOUT="table-name">
   <OUTCASLIB="caslib">
   <LABEL="label-name">
   <COMPRESS> <PROMOTE><REPLACE><REPLICATION|COPIES=integer>
   <INDEXVARS=((variable-name-1),<(variable-name-n)>)
```
Required Argument

**CASDATA=** "table-name"

specifies the name of the in-memory table to change.

Optional Arguments

**INCASLIB=** "caslib"

specifies the caslib with the in-memory table to change. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

**VARS=** \{ {casinvardesc-1}, {<casinvardesc-n>} \}

specifies the variables to use in the action. The `casinvardesc-n` value can be one or more of the following:

- **format=** "string"
  specifies the format to apply to the variable.

- **formattedLength=** `integer`
  specifies the length of format field plus the format precision.

- **label=** "string"
  specifies the descriptive label for the variable.

- **name=** "variable-name"
  specifies the name for the variable.

- **nfd=** `integer`
  specifies the length of the format precision.

- **nfl=** `integer`
  specifies the length of the format field.

**WHERE=** "where-expression"

specifies an expression for subsetting the input 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.

**IMPORTOPTIONS=(FILETYPE="file-type" <file-type-options>)**

specifies the file format and options. If you do not specify the FILETYPE= option, the server attempts to determine the file type based on the file name suffix, such as .sashdat, .csv, and so on. Files with a .txt suffix are imported as a CSV file.

**FILETYPE=** "BASESAS" | "CSV" | "DTA" | "EXCEL" | "HDAT" | "LASR" | "XLS"

specifies the file format.

**file-type-options**

specifies options for importing the data. For more information, see “Platform File Types” on page 147 and “Data Connectors” on page 155.

**DATASOURCEOPTIONS=**(data-source-options)

specifies overrides for the DATASOURCE= options for the input caslib. For more information, see “Platform Data Sources” on page 125 and “Data Connectors” on page 155.
Alias OPTIONS=

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.

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.

LABEL="label-name"
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.

COMPRESS
specifies to compress the output table. This option is supported with the DATA=
argument only.

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.

REPLACE
specifies that an in-memory table with a given name replaces an existing in-
memory table with the same name.

REPLICATION|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.

Default 1

Interaction 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.

INDEXVARS={"variable-name-1", "<variable-name-2>"}
specifies the list of variables to create indexes for in the output data.

Details
If the INDEX statement is used with no optional arguments, then all previous set
indexes are cleared.
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 98

Syntax

LIST <FILES | TABLES> <options>;

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 “Platform Data Sources” on page 125 and “Data Connectors” on page 155.

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">"
  80

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>
  <NCHARMULTIPLIER=n>
  <TRANSCODE_FAIL=ERROR | WARN | SILENT >
  <TIMEOUTSECONDS=n>
  <WRITETRANSFERSIZE=integer | integerK | integerM | integerG>;

Form 3:  LOAD FILE="path-to-client-file" CASOUT="table-name"
  <OUTCASLIB="caslib">
  <COMPRESS>
  <IMPORTOPTIONS=(FILETYPE=file-type <file-type-options>)}
  <LABEL="table-description">
  <PROMOTE | REPLACE>
  <COPIES=integer>
  <TIMEOUTSECONDS=n>
  <WRITETRANSFERSIZE=integer | integerK | integerM | integerG>;

Form 4:  LOAD FILE=fileref CASOUT="table-name"
  <OUTCASLIB="caslib">
  <COMPRESS>
  <IMPORTOPTIONS=(FILETYPE=file-type <file-type-options>)}
LOAD Statement

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=**"path-to-client-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=.

**FILE=**fileref

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.

**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
Interaction This option is ignored if the REPEAT option is also specified.

DATASOURCEOPTIONS=(data-source-options)
specifies overrides for the DATASOURCE= options for the input caslib. For more information, see “Platform Data Sources” on page 125 and “Data Connectors” on page 155.

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. If you do not specify the FILETYPE= option, the server attempts to determine the file type based on the file name suffix, such as .sashdat, .csv, and so on. Files with a .txt suffix are imported as a CSV file.

FILETYPE="BASESAS" | "CSV" | "DTA" | "EXCEL" | "HDAT" | "LASR" | "XLS"
specifies the file format.

file-type-options
specifies options for importing the data. For more information, see “Platform File Types” on page 147 and “Data Connectors” on page 155.

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
specifies the multiplication factor used when transcoding.

When SAS writes data to the CAS server, SAS estimates the number of bytes that are needed to transcode the data to UTF-8 based on the character set of the SAS session. When DBCS characters and some SBCS characters are transcoded to UTF-8, they require additional bytes to represent a character. You can use the NCHARMULTIPLIER= option to specify a byte multiplier when you know the number of bytes that are needed to represent the data in UTF-8.

CAUTION
Data truncation can occur during transcoding. If a data set is not wide enough to store the results that are encoded as UTF-8 when data transcoding occurs, then data might be truncated. To avoid truncation, use the NCHARMULTIPLIER= option or the CASNCHARMULTIPLIER= system option to widen columns so that data is not truncated.
Defaults

For SBCS environments the default is 1

For DBCS environments, the default is 1.5.

Range

0–4

Tip

A best practice is to test reading data before the option is used in a production environment.

See

“CASNCCHARMULTIPLIER= System Option” on page 473.

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.

TIMEOUTSECONDS=n

specifies the client inactivity time-out interval in seconds. If SAS stops transferring data to the server, the server starts the inactivity timer for the specified duration and cancels the request if SAS does not resume data transfer.

Alias

TIMEOUT=

Default

1800 seconds

Range

30–3600 seconds

TRANSCODE_FAIL=ERROR | WARN | SILENT

specifies how the engine handles transcoding errors.

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.

Default

ERROR
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-operator where-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.

Restriction You can specify the WHERE= as a data set option when you use the
LOAD CASDATA= form.

WRITETRANSFERSIZE=integer | integerK | integerM | integerG
specifies the maximum number of bytes the procedure writes when transferring
data to the CAS server.

integer
specifies the total number of bytes to read.

integerK
specifies the total number of kilobytes to read.

integerM
specifies the total number of megabytes to read.

integerG
specifies the total number of gigabytes to read.

Alias WTS=
Details

Summary of Options

Table 3.1 Summary of LOAD Statement Options

<table>
<thead>
<tr>
<th>LOAD Statement Option</th>
<th>LOAD Statement Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND</td>
<td>LOAD DATA=</td>
</tr>
<tr>
<td>COMPRESS</td>
<td>LOAD DATA=</td>
</tr>
<tr>
<td>COPIES=</td>
<td>All LOAD statement forms. For more information, see “Data Redundancy” on page 125.</td>
</tr>
<tr>
<td>DATASOURCEOPTIONS=</td>
<td>LOAD CASDATA= and LOAD DATA=</td>
</tr>
<tr>
<td>GROUPBY=</td>
<td>LOAD CASDATA= and LOAD DATA=</td>
</tr>
<tr>
<td>IMPORTOPTIONS=</td>
<td>LOAD CASDATA= and LOAD FILE=</td>
</tr>
<tr>
<td>INCASLIB=</td>
<td>LOAD CASDATA=</td>
</tr>
<tr>
<td>LABEL=</td>
<td>All LOAD statement forms</td>
</tr>
<tr>
<td>NCHARMULTIPLIER=</td>
<td>LOAD DATA=</td>
</tr>
<tr>
<td>ORDERBY=</td>
<td>LOAD CASDATA= and LOAD DATA=</td>
</tr>
<tr>
<td></td>
<td>Note: ORDERBY= requires GROUPBY=</td>
</tr>
<tr>
<td>OUTCASLIB=</td>
<td>All LOAD statement forms</td>
</tr>
<tr>
<td>PROMOTE</td>
<td>All LOAD statement forms</td>
</tr>
<tr>
<td>REPEAT</td>
<td>LOAD DATA=</td>
</tr>
<tr>
<td>REPLACE</td>
<td>All LOAD statement forms</td>
</tr>
<tr>
<td></td>
<td>Note: Keep in mind that global-scope tables cannot be replaced. Use the DROPTABLE statement before the LOAD statement.</td>
</tr>
<tr>
<td>TIMOUTSECONDS=</td>
<td>LOAD DATA= and LOAD FILE=</td>
</tr>
<tr>
<td>TRANSCODE_FAIL=</td>
<td>LOAD DATA=</td>
</tr>
<tr>
<td>VERS=</td>
<td>LOAD CASDATA=</td>
</tr>
<tr>
<td>WHERE=</td>
<td>LOAD CASDATA=</td>
</tr>
<tr>
<td></td>
<td>Note: You can specify the WHERE= as a data set option when you use the LOAD CASDATA= form.</td>
</tr>
</tbody>
</table>
PARTITION Statement

The PARTITION statement enables you to partition a table.

Syntax

```
PARTITION CASDATA="table-name"
  <INCASLIB="caslib">
  <VARS={ {casinvardesc-1}, {casinvardesc-n} }>
  <WHERE="where-expression <logical-operator where-expression">>
  <PARTITIONBY | GROUPBY=(group-by-variable-1 <group-by-variable-n)>
  <IMPORTOPTIONS=(FILETYPE= file-type <file-type-options>)>
  <DATASOURCEOPTIONS=(data-source-options)>
  <CASOUT="table-name">
  <OUTCASLIB="caslib">
  <LABEL="label-name">
  <COMPRESS><PROMOTE><REPLACE><REPLICATION|COPIES=integer>
  <INDEXVARS=((variable-name-1),((variable-name-n))>
```

Required Argument

`CASDATA="table-name"`

specifies the name of the in-memory table to change.

Optional Arguments

`INCASLIB="caslib"`

specifies the caslib with the in-memory table to change. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

`VARS={ {casinvardesc-1}, {casinvardesc-n} }`

specifies the variables to use in the action. The `casinvardesc-n` value can be one or more of the following:

- `format="string"`
  - specifies the format to apply to the variable.

- `formattedLength=integer`
  - specifies the length of format field plus the format precision.

- `label="string"`
  - specifies the descriptive label for the variable.
PARTITION Statement

name="variable-name"
specifies the name for the variable.

nfd=integer
specifies the length of the format precision.

nfl=integer
specifies the length of the format field.

WHERE="where-expression"
specifies an expression for subsetting the input 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.

PARTITIONBY | GROUPBY=(group-by-variable-1 <group-by-variable-n>)
specifies the names of the variables to use for grouping results.

IMPORTOPTIONS=(FILETYPE="file-type" <file-type-options>)
specifies the file format and options. If you do not specify the FILETYPE= option, the server attempts to determine the file type based on the file name suffix, such as .sashdat, .csv, and so on. Files with a .txt suffix are imported as a CSV file.

FILETYPE="BASESAS" | "CSV" | "DTA" | "EXCEL" | "HDAT" | "LASR" | "XLS"
specifies the file format.

file-type-options
specifies options for importing the data. For more information, see “Platform File Types” on page 147 and “Data Connectors” on page 155.

DATASOURCEOPTIONS=(data-source-options)
specifies overrides for the DATASOURCE= options for the input caslib. For more information, see “Platform Data Sources” on page 125 and “Data Connectors” on page 155.

Alias OPTIONS=

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.

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.
LABEL="label-name"
- 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.

COMPRESS
- specifies to compress the output table. This option is supported with the DATA= argument only.

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.

REPLACE
- specifies that an in-memory table with a given name replaces an existing in-memory table with the same name.

REPLICATION|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.
  Default 1

Interaction
- 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.

INDEXVARS={"variable-name-1", "<variable-name-2>"}
- specifies the list of variables to create indexes for in the output data.

Details
- If the PARTITION statement is used with no optional arguments, then all previous set partitions are cleared.

**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 101

**Syntax**

PROMOTE CASDATA="table-name"
  INCASLIB="caslib"<CASOUT="table-name"> <<OUTCASLIB="caslib">
  <DROP | KEEP>
  <QUIET>;

---

88 Chapter 3 / CASUTIL Procedure
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.

**QUIET**

does not return a message with status and severity. If the QUIET option is not specified, then a message returns with status and severity.

**Alias**

silent

### 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 103

#### Syntax

```
SAVE CASDATA="file-name" <INCASLIB="caslib">
  <CASOUT="table-name"> <OUTCASLIB="caslib">
  <EXPORTOPTIONS={export-options}>
  <IMPORTDATASOURCEOPTIONS={import-data-source-options}>
  <options>;
```
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.

**EXPORTOPTIONS=**{export-options}
specifies the settings for saving a table to a data source.

*Alias* `export`

*Note* The value that you specify for fileType determines the other parameters that apply.

*See* `exportOptions`

**IMPORTDATASOURCEOPTIONS=**{import-data-source-options}
enables you to save an in-memory table to the specified data source.

*See* `dataSourceOptions`

**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=`
The UPDATE statement enables you to update rows in a table.

Syntax

UPDATE CASDATA="table-name"
<INCASLIB="caslib">
<SET={ {var="variable-name", value="string"} }>
<DATASOURCEOPTIONS=(data-source-options)>
<IMPORTOPTIONS=(FILETYPE="file-type" <file-type-options>)>
<GROUPBY=(group-by-variable-1 <group-by-variable-n)>
<ORDERBY=(variable-1 <variable-n)>
<WHERE="where-expression <logical-operator where-expression"

Required Argument

CASDATA="table-name"
specifies the name of the in-memory table to change.
Optional Arguments

**INCASLIB=** "caslib"

specifies the caslib with the in-memory table to change. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

**SET=** \{ \{**var=** "variable-name", **value=** "string"\} \}

specifies the variable name and the value to set. Use the where parameter to limit the updates to specific rows.

- **var=** "variable-name"
  - specifies the column name to update.

- **value=** "string"
  - specifies the value to set the column to. Enclose the value in quotation marks even if the data type is numeric.

**DATASOURCEOPTIONS=** (data-source-options)

specifies overrides for the DATASOURCE= options for the input caslib. For more information, see “Platform Data Sources” on page 125 and “Data Connectors” on page 155.

**Alias OPTIONS=**

**IMPORTOPTIONS=** (FILETYPE= "file-type" <file-type-options>)

specifies the file format and options. If you do not specify the FILETYPE= option, the server attempts to determine the file type based on the file name suffix, such as .sashdat, .csv, and so on. Files with a .txt suffix are imported as a CSV file.

- **FILETYPE=** "BASESAS" | "CSV" | "DTA" | "EXCEL" | "HDAT" | "LASR" | "XLS"
  - specifies the file format.

- **file-type-options**
  - specifies options for importing the data. For more information, see “Platform File Types” on page 147 and “Data Connectors” on page 155.

**GROUPBY=** (group-by-variable-1 <group-by-variable-n>)

specifies the names of the variables to use for grouping results.

**ORDERBY=** (variable-1 <variable-n>)

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables, or it can be combined with variables that are specified in the groupBy parameter when the value of the groupByMode parameter is set to REDISTRIBUTE.

**WHERE=** "where-expression"

specifies an expression for subsetting the input 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.
Usage: CASUTIL Procedure

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 File Name 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;

Using Wildcard Characters for File Name 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 setssessopt=(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.

- File name 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 setssessopt=(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;

<table>
<thead>
<tr>
<th>CAS File Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>-----------------------</td>
</tr>
<tr>
<td>iris.csv</td>
</tr>
<tr>
<td>titanic3.xls</td>
</tr>
<tr>
<td>jia_free_wifi.csv</td>
</tr>
<tr>
<td>setosa.csv</td>
</tr>
</tbody>
</table>

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.

```sas
proc casutil;
  load data=sashelp.iris;
  contents casdata="iris";
quit;
```

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 file name 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. Single-machine servers do not use redundant data blocks.

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.

---

**Table Information for Caslib CASUTIL (dataset)**

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Label</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>Indexed Columns</th>
<th>Created</th>
<th>Last Modified</th>
<th>Promoted Table</th>
<th>Repeated Table</th>
<th>View</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>IRIS</td>
<td>Fisher's Iris Data (IRIS)</td>
<td>150</td>
<td>6</td>
<td>0</td>
<td>2019-09-19T13:27:38-04:00</td>
<td>2019-09-19T13:27:36-04:00</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

**Detail Information for iris in Caslib CASUTIL (dataset)**

<table>
<thead>
<tr>
<th>Node</th>
<th>Number of Blocks</th>
<th>Active Blocks</th>
<th>Rows</th>
<th>Data Size</th>
<th>Variable Data Size</th>
<th>Blocks Mapped</th>
<th>Memory Mapped</th>
<th>Blocks Unmapped</th>
<th>Memory Unmapped</th>
<th>Blocks Allocated</th>
<th>Memory Allocated</th>
<th>Index Size</th>
<th>Compressed Size</th>
<th>Compression Ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>4</td>
<td>2</td>
<td>150</td>
<td>7280</td>
<td>0</td>
<td>2</td>
<td>7582</td>
<td>2</td>
<td>7582</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Column Information for iris in Caslib CASUTIL (dataset)**

<table>
<thead>
<tr>
<th>Column</th>
<th>Label</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
<th>Exposed Width</th>
<th>Exposed Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>SepalLength</td>
<td>Sepal Length</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>PetalLength</td>
<td>Petal Length</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 file name and the Source Caslib field indicates the original caslib.
Variable Data size
This field shows the number of bytes that are used for varying-length character
columns (VARCHAR) and varying-length binary columns (VARBINARY). When
these data types are used in a table, each row includes a 16-byte descriptor for
each varying-length column.

Blocks Mapped
This field shows the number of blocks that are currently mapped into memory.
For small tables, smaller than 16M, this value can be 0. A zero indicates that the
data is in memory only. A nonzero value indicates that some data is either using
blocks from a SASHDAT file or that data is temporarily stored in
CAS_DISK_CACHE.

Note: The 16M value is the default value for the MAXTABLEMEM= session
option. For more information, see the MAXTABLEMEM= on page 402 session
option.

Memory Mapped
This field shows the number of bytes for the blocks that are mapped. The blocks
are mapped from temporary files in CAS_DISK_CACHE or from a SASHDAT
file.

If the table is from a SASHDAT file that is encrypted or is subset with a WHERE
filter, then blocks from the original SASHDAT file are not used. In these cases,
temporary files in CAS_DISK_CACHE are used.

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 are allocated in memory and 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
- when you append or update rows

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.

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 3.2 ODS Tables Produced by the CASUTIL Procedure*

<table>
<thead>
<tr>
<th>ODS Table</th>
<th>Description</th>
<th>Statement Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>CaslibInfo</td>
<td>Caslib information</td>
<td>LIST</td>
</tr>
<tr>
<td>ColumnInfo</td>
<td>Column information</td>
<td>CONTENTS</td>
</tr>
<tr>
<td>FileInfo</td>
<td>CAS file information</td>
<td>LIST with the FILES option</td>
</tr>
<tr>
<td>TableDetails</td>
<td>Detailed table information</td>
<td>CONTENTS</td>
</tr>
<tr>
<td>TableInfo</td>
<td>CAS table information</td>
<td>LIST with the TABLES option, CONTENTS</td>
</tr>
</tbody>
</table>

**Examples: CASUTIL Procedure**

**Example 1: Load a CSV File into CAS**

**Program**

```plaintext
caslib csvfiles task=add type=dnfs
   path="/data/csv/"
   desc="Spreadsheets and CSV source data.";
proc casutil;
   list files;
```
load casdata="County_Population.csv"

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.

<table>
<thead>
<tr>
<th>Name</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>File Size</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spreadsheet10k.xlsx</td>
<td>r-xr-xr-x</td>
<td>nfsnobody</td>
<td>nfsnobody</td>
<td>563.5KB</td>
<td>08Nov2012:07:21:12</td>
</tr>
<tr>
<td>mailorder.csv</td>
<td>r-xr-xr-x</td>
<td>nfsnobody</td>
<td>nfsnobody</td>
<td>117.2KB</td>
<td>29Sep2015:11:40:14</td>
</tr>
<tr>
<td>NST_EST2012_ALLDATA.csv</td>
<td>r-xr-xr-x</td>
<td>nfsnobody</td>
<td>nfsnobody</td>
<td>20.0KB</td>
<td>15Jan2013:08:59:26</td>
</tr>
<tr>
<td>sine.csv</td>
<td>r-xr-xr-x</td>
<td>nfsnobody</td>
<td>nfsnobody</td>
<td>305.3KB</td>
<td>15Jan2013:13:18:14</td>
</tr>
<tr>
<td>SC-EST2011-6RACE-ALL.csv</td>
<td>r-xr-xr-x</td>
<td>nfsnobody</td>
<td>nfsnobody</td>
<td>138.3KB</td>
<td>15Jan2013:09:02:53</td>
</tr>
<tr>
<td>sine10k.csv</td>
<td>r-xr-xr-x</td>
<td>nfsnobody</td>
<td>nfsnobody</td>
<td>582.9KB</td>
<td>09May2013:07:16:16</td>
</tr>
<tr>
<td>UNdata_Export_20130115_002219649.csv</td>
<td>r-xr-xr-x</td>
<td>nfsnobody</td>
<td>nfsnobody</td>
<td>695.4KB</td>
<td>15Jan2013:09:22:20</td>
</tr>
<tr>
<td>2012_nfl_php_data_reg_season1.csv</td>
<td>r-xr-xr-x</td>
<td>nfsnobody</td>
<td>nfsnobody</td>
<td>6.5MB</td>
<td>06Feb2013:11:19:02</td>
</tr>
<tr>
<td>County_Population.csv</td>
<td>r-xr-xr-x</td>
<td>nfsnobody</td>
<td>nfsnobody</td>
<td>1.3MB</td>
<td>14Jan2013:14:15:41</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>NL3 Encoding</th>
<th>Created</th>
<th>Last Modified</th>
<th>Promoted Table</th>
<th>Repeated Table</th>
<th>View</th>
<th>Source Name</th>
<th>Source Caslib</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>COUNTY_POPULATION</td>
<td>3183</td>
<td>97</td>
<td>utf-8</td>
<td>09Apr2016:18:41:18</td>
<td>09Apr2016:10:41:18</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>County_Population.csv</td>
<td>CSVFILES</td>
<td>No</td>
</tr>
</tbody>
</table>
Example 2: Append Rows to an In-Memory Table

Program

```sas
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.
Example 3: Promote a Table

Program

```plaintext
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 88 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.
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.

```sas
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')
    casout="jta_free_wifi2";
quit;

proc mdsummary data=mycas.jta_free_wifi2; /* 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 74 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 104.
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.

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.

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.

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 3.2 on page 105.

Results: CONTENTS Statement for the Jta_free_wifi Table

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>NLS encoding</th>
<th>Created</th>
<th>Last Modified</th>
<th>Promoted Table</th>
<th>Repeated Table</th>
<th>View</th>
<th>Source Name</th>
<th>Source Caslib</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>JTA_FREE_WIFI</td>
<td>42269</td>
<td>21</td>
<td>utf-8</td>
<td>09-Apr-2016 13:56:01</td>
<td>09-Apr-2016 13:56:01</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>Jta_free_wifi_cases</td>
<td>CASUSER:</td>
<td>No</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node</th>
<th>Number of Blocks</th>
<th>Active Blocks</th>
<th>Blocks Mapped</th>
<th>Blocks Unmapped</th>
<th>Memory Mapped</th>
<th>Blocks Allocated</th>
<th>Memory Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>84</td>
<td>42</td>
<td>42359</td>
<td>327383247</td>
<td>19883511</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>スポット ID</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>スポット名（日本語）</td>
<td>varchar</td>
<td>115</td>
<td>55</td>
</tr>
<tr>
<td>スポット名（英語）</td>
<td>varchar</td>
<td>256</td>
<td>256</td>
</tr>
<tr>
<td>スポットデータ</td>
<td>varchar</td>
<td>20</td>
<td>8</td>
</tr>
</tbody>
</table>
### Results: WiFi Hotspot Categories and Counts

#### Output 3.1 WiFi Hotspot Categories and Counts

<table>
<thead>
<tr>
<th>Obs</th>
<th>カテゴリー</th>
<th><em>NObs</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>その他</td>
<td>31989</td>
</tr>
<tr>
<td>2</td>
<td>バス</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>ホテル</td>
<td>1741</td>
</tr>
<tr>
<td>4</td>
<td>商業施設(百貨店、SC、アウトレットモール等)</td>
<td>46</td>
</tr>
<tr>
<td>5</td>
<td>港湾</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>移動中の休憩所(サービスエリア、道の駅等)</td>
<td>181</td>
</tr>
<tr>
<td>7</td>
<td>空港</td>
<td>6</td>
</tr>
<tr>
<td>8</td>
<td>美術館・博物館・寺社仏閣</td>
<td>11</td>
</tr>
<tr>
<td>9</td>
<td>観光スポット(景勝地等)</td>
<td>33</td>
</tr>
<tr>
<td>10</td>
<td>観光案内所</td>
<td>91</td>
</tr>
<tr>
<td>11</td>
<td>鉄道(駅構内)</td>
<td>175</td>
</tr>
<tr>
<td>12</td>
<td>飲食・小売店</td>
<td>7979</td>
</tr>
</tbody>
</table>
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

Restrictions: 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 120 and "Add and Save a User-Defined Format" in SAS Cloud Analytic Services: User-Defined Formats. You cannot use a BY statement in a PROC MDSUMMARY step. Use the MDSUMMARY procedure GROUPBY statement on page 109 instead.

```
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

```plaintext
VAR <variable-list>
```

Without Arguments
If no variables are listed, then the summary statistics are computed for all numeric variables.

Optional Argument
```plaintext
<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.

**Note:**
You cannot use a BY statement in an MDSUMMARY procedure step. Use a GROUPBY statement instead.

**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

```plaintext
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.

Usage: MDSUMMARY Procedure

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. They are listed in the following table.

Table 4.1 Table of Statistic Produced by the MDSUMMARY Procedure

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>CSS</em></td>
<td>CSS</td>
<td>Corrected sum of squares</td>
</tr>
<tr>
<td><em>CV</em></td>
<td>CV</td>
<td>Coefficient of variation</td>
</tr>
<tr>
<td><em>KURTOSIS</em></td>
<td>KURTOSIS</td>
<td>Kurtosis</td>
</tr>
<tr>
<td><em>MAX</em></td>
<td>MAX</td>
<td>Maximum value</td>
</tr>
<tr>
<td><em>MEAN</em></td>
<td>MEAN</td>
<td>Arithmetic mean</td>
</tr>
<tr>
<td><em>MIN</em></td>
<td>MIN</td>
<td>Minimum value</td>
</tr>
<tr>
<td><em>NMISS</em></td>
<td>NMISS</td>
<td>Number of values that are missing</td>
</tr>
</tbody>
</table>
### Examples: MDSUMMARY Procedure

#### Example 1: Compute Descriptive Statistics

**Program**

The following example shows you how to access your data with SAS Cloud Analytic Services (CAS). It shows you how to compute all statistics for each variable, and treat the entire input table as one group. The results are written to an output table.

```sas
/* options cashost="cloud.example.com" casport=5570; */
cas casauto; /* 1 */
libname mycas cas; /* 2 */
proc casutil; /* 3 */
  load data=sashelp.cars replace;
  contents casdata="cars";
  quit;
proc mdsummary data=mycas.cars; /* 4 */
  var MSRP MPG_City;
```

<table>
<thead>
<tr>
<th>Column Name</th>
<th>Statistic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><em>NOBS</em></td>
<td>NOBS</td>
<td>Total number of observations</td>
</tr>
<tr>
<td><em>PRT</em></td>
<td>PRT</td>
<td>p-Value for Student’s t statistics</td>
</tr>
<tr>
<td><em>SKEWNESS</em></td>
<td>SKEWNESS</td>
<td>Skewness</td>
</tr>
<tr>
<td><em>STD</em></td>
<td>STD</td>
<td>Standard deviation</td>
</tr>
<tr>
<td><em>STDERR</em></td>
<td>STDERR</td>
<td>Standard error of the mean</td>
</tr>
<tr>
<td><em>SUM</em></td>
<td>SUM</td>
<td>Sum</td>
</tr>
<tr>
<td><em>T</em></td>
<td>T</td>
<td>Student’s t statistic</td>
</tr>
<tr>
<td><em>USS</em></td>
<td>USS</td>
<td>Uncorrected sum of squares</td>
</tr>
<tr>
<td><em>VAR</em></td>
<td>VAR</td>
<td>Variance</td>
</tr>
</tbody>
</table>
Example 1: Compute Descriptive Statistics

```
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" replace;
   quit;
```

1 The CAS statement creates session Casauto.

2 The LIBNAME statement creates a CAS engine libref.

---

Note: 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.


5 The MDSUMMARY procedure does not print the output. Use the PRINT procedure to print the table Mycas.MdsumStat.

6 The CASUTIL procedure saves the table to the default caslib. The SAVE statement stores the data as UTF-8 when it creates the SASHDAT file.
Results

Output 4.1  Column Information for the Cars Data Set

<table>
<thead>
<tr>
<th>Column</th>
<th>Label</th>
<th>Type</th>
<th>Length</th>
<th>Format Name</th>
<th>Formatted Length</th>
<th>Format Width</th>
<th>Format Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make</td>
<td></td>
<td>char</td>
<td>13</td>
<td></td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Model</td>
<td></td>
<td>char</td>
<td>40</td>
<td></td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td>char</td>
<td>8</td>
<td></td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Origin</td>
<td></td>
<td>char</td>
<td>6</td>
<td></td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DriveTrain</td>
<td></td>
<td>char</td>
<td>5</td>
<td></td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MSRP</td>
<td></td>
<td>double</td>
<td>8</td>
<td>DOLLAR</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Invoice</td>
<td></td>
<td>double</td>
<td>8</td>
<td>DOLLAR</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>EngineSize</td>
<td>Engine Size (L)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cylinders</td>
<td></td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Horsepower</td>
<td></td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MPG_City</td>
<td>MPG (City)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MPG_Highway</td>
<td>MPG (Highway)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Weight</td>
<td>Weight (LBS)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wheelbase</td>
<td>Wheelbase (IN)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Length</td>
<td>Length (IN)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Results: MDSUMMARY Procedure

Output 4.2  Summary of MSRP and City Miles per Gallon

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>Column</em></th>
<th>NObs</th>
<th><em>Mean</em></th>
<th><em>Max</em></th>
<th><em>Min</em></th>
<th><em>Std</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MSRP</td>
<td>428</td>
<td>32774</td>
<td>85514</td>
<td>192465</td>
<td>19431716674</td>
</tr>
<tr>
<td>2</td>
<td>MPG_City</td>
<td>428</td>
<td>20.060747664</td>
<td>60</td>
<td>10</td>
<td>5.2382176396</td>
</tr>
</tbody>
</table>

Additional Information

- Tables that are saved from a caslib are saved in SASHDAT format by default.
- For documentation about the CASUTIL procedure syntax, see Chapter 3, “CASUTIL Procedure,” on page 69.
- 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 example shows you how to access your data with SAS Cloud Analytic Services (CAS) and compute all statistics for each variable (treating the entire input table as one group). It also computes all statistics for every unique combination of the formatted values of variables Make and Type. The results are written to an output table.

```sas
/* options cashost="cloud.example.com" casport=5570; */
cas casauto; /* 3 */
libname mycas cas; /* 2 */
proc casutil; /* 3 */
    load data=sashelp.cars replace;
    contents casdata="cars";
quit;

proc mdsummary data=mycas.cars; /* 4 */
    var MPG_City;
    groupby / out=mycas.carsmpgcityall; /* 5 */
    groupby make type / out=mycas.carsmaketype; /* 6 */
run;

proc print data=mycas.carsmpgcityall; /* 7 */
    var _Column_ _NObs_ _Mean_ _Std_ _Min_ _Max_; 
    title 'Overall City Mileage';
run;

proc print data=mycas.carsmaketype(obs=20); /* 8 */
    var make Type _Column_ _NObs_ _Mean_ _Std_ _Min_ _Max_; 
    title 'City Mileage by Make and Type';
run;
```

1. The CAS statement creates session Casauto.
2. The LIBNAME statement creates a CAS engine libref.
3. The CASUTIL procedure loads the data into the default caslib. 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.
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.

The second GROUPBY statement with the OUT= option creates an in-memory table named Mycas.CarsMakeType. The in-memory table calculates summaries for rows grouped by Make and Type.

The MDSUMMARY procedure does not print output. The PRINT procedure prints table Mycas.MdsumStat.

The PRINT procedure prints the first 20 observations in table Mycas.CarsMakeType.

Results: PROC CASUTIL CONTENTS Statement

Output 4.3  Column Information for the Cars Data Set

<table>
<thead>
<tr>
<th>Column</th>
<th>Label</th>
<th>Type</th>
<th>Length</th>
<th>Format Name</th>
<th>Formatted Length</th>
<th>Format Width</th>
<th>Format Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make</td>
<td></td>
<td>char</td>
<td>13</td>
<td></td>
<td>13</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Model</td>
<td></td>
<td>char</td>
<td>40</td>
<td></td>
<td>40</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td>char</td>
<td>8</td>
<td></td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Origin</td>
<td></td>
<td>char</td>
<td>6</td>
<td></td>
<td>6</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>DriveTrain</td>
<td></td>
<td>char</td>
<td>5</td>
<td></td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MSRP</td>
<td></td>
<td>double</td>
<td>8</td>
<td>DOLLAR</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Invoice</td>
<td></td>
<td>double</td>
<td>8</td>
<td>DOLLAR</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>EngineSize</td>
<td>Engine Size (L)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Cylinders</td>
<td></td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Horsepower</td>
<td></td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MPG_City</td>
<td>MPG (City)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>MPG_Highway</td>
<td>MPG (Highway)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Weight</td>
<td>Weight (LBS)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Wheelbase</td>
<td>Wheelbase (IN)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Length</td>
<td>Length (IN)</td>
<td>double</td>
<td>8</td>
<td></td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Results: MDSUMMARY Procedure

Output 4.4  Overall Summary of City Mileage

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>Column</em></th>
<th><em>NObs</em></th>
<th><em>Mean</em></th>
<th><em>Std</em></th>
<th><em>Min</em></th>
<th><em>Max</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>MPG_City</td>
<td>428</td>
<td>20.060747664</td>
<td>5.23353176386</td>
<td>10</td>
<td>60</td>
</tr>
</tbody>
</table>
### Example 2: Computing Descriptive Statistics with Group-By Variables

#### Output 4.5  Summary of City Mileage by Make and Type

<table>
<thead>
<tr>
<th>Obs</th>
<th>Make</th>
<th>Type</th>
<th><em>Column</em></th>
<th><em>NObs</em></th>
<th><em>Mean</em></th>
<th><em>Std</em></th>
<th>Min.</th>
<th>Max.</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Audi</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>13</td>
<td>18.616384615</td>
<td>2.3642638579</td>
<td>14</td>
<td>23</td>
</tr>
<tr>
<td>2</td>
<td>Audi</td>
<td>Sports</td>
<td>MPG_City</td>
<td>4</td>
<td>19</td>
<td>2.7080128015</td>
<td>15</td>
<td>21</td>
</tr>
<tr>
<td>3</td>
<td>BMW</td>
<td>SUV</td>
<td>MPG_City</td>
<td>2</td>
<td>16</td>
<td>0</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>Chevrolet</td>
<td>SUV</td>
<td>MPG_City</td>
<td>4</td>
<td>15.75</td>
<td>2.3629078131</td>
<td>14</td>
<td>19</td>
</tr>
<tr>
<td>5</td>
<td>Chrysler</td>
<td>Sports</td>
<td>MPG_City</td>
<td>1</td>
<td>17</td>
<td>.</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>6</td>
<td>CMC</td>
<td>Truck</td>
<td>MPG_City</td>
<td>4</td>
<td>15.75</td>
<td>2.217357826</td>
<td>13</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>Infiniti</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>6</td>
<td>17.833333333</td>
<td>0.7527726527</td>
<td>17</td>
<td>19</td>
</tr>
<tr>
<td>8</td>
<td>Isuzu</td>
<td>SUV</td>
<td>MPG_City</td>
<td>2</td>
<td>16</td>
<td>1.4142135624</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>9</td>
<td>Mercedes-Benz</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>16</td>
<td>18.0625</td>
<td>2.2351360883</td>
<td>13</td>
<td>22</td>
</tr>
<tr>
<td>10</td>
<td>Mercedes-Benz</td>
<td>Sports</td>
<td>MPG_City</td>
<td>5</td>
<td>16.2</td>
<td>3.1144823005</td>
<td>13</td>
<td>21</td>
</tr>
<tr>
<td>11</td>
<td>Nissan</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>9</td>
<td>22</td>
<td>3.6742346142</td>
<td>18</td>
<td>28</td>
</tr>
<tr>
<td>12</td>
<td>Pontiac</td>
<td>Sports</td>
<td>MPG_City</td>
<td>1</td>
<td>16</td>
<td>.</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>13</td>
<td>Saab</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>6</td>
<td>20.666666667</td>
<td>0.5163977795</td>
<td>20</td>
<td>21</td>
</tr>
<tr>
<td>14</td>
<td>Saturn</td>
<td>Wagon</td>
<td>MPG_City</td>
<td>1</td>
<td>24</td>
<td>.</td>
<td>24</td>
<td>24</td>
</tr>
<tr>
<td>15</td>
<td>Subaru</td>
<td>Sports</td>
<td>MPG_City</td>
<td>2</td>
<td>19</td>
<td>1.4142135624</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>16</td>
<td>BMW</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>13</td>
<td>19.230769231</td>
<td>0.8320502943</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>17</td>
<td>BMW</td>
<td>Sports</td>
<td>MPG_City</td>
<td>4</td>
<td>18.25</td>
<td>2.6299666397</td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>18</td>
<td>Buick</td>
<td>SUV</td>
<td>MPG_City</td>
<td>2</td>
<td>17</td>
<td>2.8284271247</td>
<td>15</td>
<td>19</td>
</tr>
<tr>
<td>19</td>
<td>Chevrolet</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>15</td>
<td>22.266666667</td>
<td>4.0964660758</td>
<td>14</td>
<td>28</td>
</tr>
<tr>
<td>20</td>
<td>Lexus</td>
<td>SUV</td>
<td>MPG_City</td>
<td>3</td>
<td>15.333333333</td>
<td>2.5166114784</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>

#### 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 3, “CASUTIL Procedure,” on page 69.

- 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

```plaintext
/* options cashost="cloud.example.com" casport=5570; */
cas casauto;            /* 1 */
proc casutil;           /* 2 */
   load data=sashelp.cars replace;
   contents casdata="cars";
quit;
libname mycas cas;      /* 3 */
proc mdsummary data=mycas.cars;      /* 4 */
   var MPG_City;
   groupby make model type / out=mycas.carsmiles;
run;
proc print data=mycas.carsmiles(obs=20); /* 5 */
   var Make Model Type _Column_ _NObs_ _Min_ _Max_;
   title 'City Mileage for Make, Model, and Type';
run;
```

1 The CAS statement creates session Casauto.

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 LIBNAME statement for the CAS engine creates a CAS libref.

4 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.

5 The PRINT procedure prints the first 20 observations in table Mycas.CarsMiles.

Note: 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.
### Results

**Output 4.6** PROC MDSUMMARY Output: City Mileage for Make, Model, and Type

<table>
<thead>
<tr>
<th>Obs</th>
<th>Make</th>
<th>Model</th>
<th>Type</th>
<th><em>Column</em></th>
<th><em>NObs</em></th>
<th><em>Min</em></th>
<th><em>Max</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Audi</td>
<td>A4 3.0 Quattro 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>Audi</td>
<td>A4.18T convertible 2dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>Audi</td>
<td>A6 3.0 Quattro 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>Audi</td>
<td>TT 1.8 convertible 2dr</td>
<td>Sports</td>
<td>MPG_City</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>5</td>
<td>BMW</td>
<td>330xi 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>BMW</td>
<td>525i 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>7</td>
<td>BMW</td>
<td>Z4 convertible 2.5i 2dr</td>
<td>Sports</td>
<td>MPG_City</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>Buick</td>
<td>Rendezvous CX</td>
<td>SUV</td>
<td>MPG_City</td>
<td>1</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>9</td>
<td>Cadillac</td>
<td>SRX V6</td>
<td>SUV</td>
<td>MPG_City</td>
<td>1</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>10</td>
<td>Chevrolet</td>
<td>Cavalier 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>11</td>
<td>Chevrolet</td>
<td>Malibu LT 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>12</td>
<td>Chevrolet</td>
<td>Silverado SS</td>
<td>Truck</td>
<td>MPG_City</td>
<td>1</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>13</td>
<td>Chevrolet</td>
<td>Tahoe LT</td>
<td>SUV</td>
<td>MPG_City</td>
<td>1</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>Chrysler</td>
<td>Concorde LX 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>15</td>
<td>Chrysler</td>
<td>Concorde LXi 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>16</td>
<td>Chrysler</td>
<td>PT Cruiser Limited 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>17</td>
<td>Chrysler</td>
<td>Sebring Touring 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>18</td>
<td>Ford</td>
<td>Crown Victoria LX 4dr</td>
<td>Sedan</td>
<td>MPG_City</td>
<td>1</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>19</td>
<td>Ford</td>
<td>Escape XLS</td>
<td>SUV</td>
<td>MPG_City</td>
<td>1</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>GMC</td>
<td>Yukon XL 2500 SLT</td>
<td>SUV</td>
<td>MPG_City</td>
<td>1</td>
<td>13</td>
<td>13</td>
</tr>
</tbody>
</table>

### 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 3, “CASUTIL Procedure,” on page 69.

- 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.

```sas
/* options cashost="cloud.example.com" casport=5570; */
cas casauto;                                  /* 1 */
libname mycas cas;                           /* 2 */
proc format casfmtlib='fmtlib';              /* 3 */
  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;                            /* 4 */
  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 mdssummary data=mycas.cake;            /* 5 */
```

Chapter 4 / MDSUMMARY Procedure
The CAS statement creates session Casauto.

The LIBNAME statement for the CAS engine creates a CAS engine libref.

Note: 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.

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.

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.

The MDSUMMARY procedure computes summary statistics for cake tasting scores.

The first GROUPBY statement with the OUT= option creates an in-memory table named Mycas.Flav that is grouped by Flavor.

The second GROUPBY statement with the OUT= option creates an in-memory table named Mycas.Flag that is grouped by Flavor and Age.

The PRINT procedure prints the output data sets.

Results

Output 4.7  PROC PRINT Output: Cake Flavors and Participant’s Age Grouped by Flavor

<table>
<thead>
<tr>
<th>Obs</th>
<th>Flavor</th>
<th><em>Column</em></th>
<th><em>NObs</em></th>
<th><em>Min</em></th>
<th><em>Max</em></th>
<th><em>Mean</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>TasteScore</td>
<td>1</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Chocolate</td>
<td>9</td>
<td>72</td>
<td>92</td>
<td>81.4444444444</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>Other Flavor</td>
<td>4</td>
<td>72</td>
<td>91</td>
<td>81.75</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Vanilla</td>
<td>6</td>
<td>73</td>
<td>94</td>
<td>82.1666666667</td>
<td></td>
</tr>
</tbody>
</table>
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 3, “CASUTIL Procedure,” on page 69.

- 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.

```plaintext
/* options cashost="cloud.example.com" casport=5570; */
cas casauto; /* 1 */
libname mycas cas; /* 2 */
proc casutil; /* 3 */
load data=sashelp.cars replace;
  contents casdata="cars";
quit;
```
The CAS statement creates session Casauto.

The LIBNAME statement for the CAS engine creates a CAS engine libref.

Note: 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.

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.

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.

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:
Additional Information

- For documentation about the CASUTIL procedure syntax, see Chapter 3, “CASUTIL Procedure,” on page 69.
- 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.
Data Redundancy

The following table shows how several factors interact with respect to data redundancy. Data redundancy applies to distributed servers only.

Table 5.1 Data Redundancy by Data Access Method, Data Source, and File Type

<table>
<thead>
<tr>
<th>Data Access Method</th>
<th>Caslib Data Source</th>
<th>Redundancy</th>
<th>File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATA step and PROC CASUTIL; LOAD DATA=</td>
<td>The caslib data source isn’t a factor for these data access methods.</td>
<td>Not applicable.</td>
<td>SAS Data Sets</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### Data Access Method

<table>
<thead>
<tr>
<th>Caslib Data Source</th>
<th>SASHDAT files</th>
<th>Other files</th>
<th>File Formats</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROC CASUTIL;</strong></td>
<td><strong>DNFS</strong></td>
<td>Based on COPIES= when the file is loaded.</td>
<td>SASHDAT and CSV files</td>
</tr>
<tr>
<td><strong>LOAD CASDATA=</strong></td>
<td></td>
<td>Based on COPIES= when the file is loaded.</td>
<td>SASHDAT and CSV files</td>
</tr>
<tr>
<td><strong>HDFS</strong></td>
<td></td>
<td>Based on COPIES= when the file was saved.</td>
<td>SASHDAT and CSV files</td>
</tr>
<tr>
<td><strong>PATH</strong></td>
<td></td>
<td>Based on COPIES= when the file is loaded.</td>
<td>SASHDAT, CSV, and PC Files</td>
</tr>
<tr>
<td><strong>Data Connectors</strong></td>
<td>Not applicable.</td>
<td>Based on COPIES= when the file is loaded.</td>
<td>Based on the data connector.</td>
</tr>
</tbody>
</table>

### Dictionary

**Azure Data Lake Storage Data Source**

Specifies an Azure Data Lake Storage Gen 2 directory for loading and saving files with SAS Cloud Analytic Services.

**Valid in:**

- CASLIB statement
- LOAD CASDATA= statement for the CASUTIL procedure
- SAVE statement for the CASUTIL procedure

**Note:**

Support for Azure Data Lake Storage was added in [SAS Viya 3.5](https://support.sas.com/documentation/viya/3.5/).  

**Syntax**

```
CASLIB caslib-reference-name
   DATASOURCE=(SRCTYPE="ADLS" ACCOUNTNAME="string" APPLICATIONID="string" FILESYSTEM="file-system-name" <DNSSUFFIX="network-storage-host-name" > <TIMEOUT=seconds>)
   <caslib-statement-options> <SUBDIRS>;
```
LOAD CASDATA="file-name" CASOUT="table-name"
<DATASOURCEOPTIONS=(…options…)> <load-statement-options>;

PROC CASUTIL;
SAVE CASDATA="table-name" CASOUT="file-name"
<DATASOURCEOPTIONS=(…options…)> <save-statement-options>;

Data Source Arguments

ACCOUNTNAME="account-name"

specifies the name of the account to use on the external storage system.

Valid in

<table>
<thead>
<tr>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
<tr>
<td>PROC CASUTIL: LOAD statement</td>
</tr>
</tbody>
</table>

Requirement

The ACCOUNTNAME= argument is required for the CASLIB statement.

APPLICATIONID="string"

specifies the application ID (a GUID string) for accessing the external storage system.

Valid in

<table>
<thead>
<tr>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
<tr>
<td>PROC CASUTIL: LOAD statement</td>
</tr>
</tbody>
</table>

Example

applicationId="b1fc955d5c-e0e2-45b3-a3cc-a1cf54120f"

DNSSUFFIX="network-storage-host-name"

specifies the network host name where the storage system resides. It is referred to as a suffix because the value is typically appended to the end of the storage account name.

Valid in

<table>
<thead>
<tr>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
<tr>
<td>PROC CASUTIL: LOAD statement</td>
</tr>
</tbody>
</table>

Default

dfs.core.windows.net

Example
dnsSuffix="dfs.core.windows.net"

FILESYSTEM="file-system-name"

specifies the name of the file system within the external storage system that files are to be read from and written to.

Valid in

<table>
<thead>
<tr>
<th>CASLIB statement</th>
</tr>
</thead>
</table>
PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements

PROC CASUTIL: LOAD statement

Requirement: The FILESYSTEM= argument is required for the CASLIB statement.

SRCTYPE="ADLS"
specifies that the data source is Azure Data Lake Storage Gen 2.

Requirement: The SRCTYPE= argument is required for the CASLIB statement.

TIMEOUT=seconds
specifies the maximum completion time for a storage task, expressed as a number of seconds. If the task does not complete within the specified time, the storage system might return an error. Use this option to establish a connection to the external storage system.

Valid in: CASLIB statement

PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements

PROC CASUTIL: LOAD statement

Default: 60

Example: timeout=600

Details

Requirements

Table 5.2 Requirements for Accessing Data in Azure Data Lake Storage Gen 2

<table>
<thead>
<tr>
<th>Requirement</th>
<th>How to meet the requirement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Tenant ID</td>
<td>If all users can use the same tenant ID, then an administrator can configure a server option to set the tenant ID. For more information, see cas.AZURETENANTID= in SAS Viya Administration: SAS Cloud Analytic Services. Users can set a tenant ID individually, or override the server setting by specifying the AZURETENANTID with the SESSOPTS= option for the CAS statement.</td>
</tr>
</tbody>
</table>
Authentication

An ADLS caslib uses device code authentication with Azure. After users authenticate, the credentials are cached in a path that is accessible to the CAS controller.

To cache the credential files for all users in a single directory, an administrator can configure the `cas.AZUREAUTHCACHELOC=` server option.

If the server option is not configured, then the credential file is created in the home directory for the host account that runs the CAS session.

Account, Application ID, and File system

These values are specified when the caslib is added. The LOAD CASDATA=, SAVE, and other statements for the CASUTIL procedure can override these values in the DATASOURCEOPTIONS= option.

Authentication

The first time that an ADLS caslib is accessed, you are prompted with an error message in the client log to authorize access:

```
ERROR: Authorization Pending user-interaction
ERROR: To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code ###### to authenticate.
```

After you log on to the Microsoft web page, a series of windows provide further instruction and result in the required authorization. Subsequent attempts to access data succeed. The authorization steps create a file that is accessible to the CAS controller for the host account that is used to run the CAS session. The file is named .sasadls_userid.json and is used to cache authentication information. It is created in the directory that is specified in the azureAuthCacheLoc CAS server option. If azureAuthCacheLoc was not set when the server was started, the JSON file is created in the home directory for the host account that is used to run the CAS session.

This authentication and authorization step is not needed as long as the Azure authentication JSON file that is created in the location specified in the azureAuthCacheLoc server option is not deleted and the user credentials do not change.

For information about setting the directory to use for caching credentials, see `cas.AZUREAUTHCACHELOC="path"` in SAS Viya Administration: SAS Cloud Analytic Services.

Examples:

Example 1: Add an ADLS Caslib and List the Available Files

``` stata
   cas casauto sessopt=(azureTenantId='b1c24e5c-3625-4593-a430-9552373a0c2f');
   caslib azds datasource=(
```
If the Azure tenant ID is not set as a server option, you can specify `AZURETENANTID` on page 391 with the `SESSOPTS=` option for the `CAS` statement.

Example 2: Load an ORC File from an ADLS Caslib

```sas
/* After setting the azureTenantId in the CAS session and adding an ADLS caslib. */
proc casutil;
   load casdata="somefile.orc" casout="somefile"; /* 1 */
      
   load casdata="subdir/otherfile.orc" casout="otherfile"; /* 2 */
quit;
```

1 By default, files are read from the root directory of the caslib. This is equivalent to the `filesystem` argument that is specified when the caslib is added.

2 If the caslib is added with the `SUBDIRS` option, then you can specify the subdirectory to access in the `CASDATA=` argument.

Example 3: Save a Table as an ORC File in an ADLS Caslib

```sas
/* After setting the azureTenantId in the CAS session and adding an ADLS caslib. */
proc casutil;
   load data=sashelp.class casout="class" outcaslib="azds";
   save casdata="class" casout="class.orc";
quit;
```

You can load data into CAS any way that you prefer. This short example shows the `LOAD DATA=` syntax that reads data from a SAS libref and loads it into the server.

To save data to Azure Data Lake Storage Get 2 as an ORC file, specify the `.orc` file name suffix or specify the `EXPORTOPTIONS=(FILETYPE="orc")` argument with `SAVE` statement.

See Also
- “CASLIB Statement” on page 447
- Chapter 3, “CASUTIL Procedure,” on page 69
HDFS Data Source

Specifies a Hadoop Distributed File System directory for loading and saving files that the SAS Cloud Analytic Services controller can access.

Valid in: CASLIB statement, LOAD CASDATA= statement for the CASUTIL procedure, SAVE statement for the CASUTIL procedure

Applies to: Distributed servers only

Note: SASHDAT files in HDFS cannot be exported from HDFS and used with DNFS or PATH caslibs.

**CAUTION:**

Backing up or copying SASHDAT files must preserve block size. SASHDAT files stored in HDFS are stored as many individual blocks. The blocks are a fixed size and each block begins with a SASHDAT header. To back up or copy SASHDAT files, you must determine the block size and preserve the block size. See “Preserving SASHDAT Block Size” on page 132.

Syntax

**CASLIB caslib-reference-name**

| DATASOURCE=(SRCTYPE="HDFS" <ENCRYPTIONDOMAIN="string" > <ENCRYPTIONPASSWORD="string" > PATH="/directory-path" <caslib-statement-options>); |

**PROC CASUTIL;**

LOAD CASDATA="file-name" CASOUT="table-name"

| DATASOURCEOPTIONS=(<ENCRYPTIONDOMAIN="string" > <ENCRYPTIONPASSWORD="string" >) <load-statement-options>; |

**PROC CASUTIL;**

SAVE CASDATA="table-name" CASOUT="file-name"

| DATASOURCEOPTIONS=(<ENCRYPTIONDOMAIN="string" > <ENCRYPTIONPASSWORD="string" >) <save-statement-options>; |

Data Source Arguments

**ENABLEREMOTESAVE=TRUE | FALSE**

when set to True and the HDFS cluster is remote from the server, the Save action is enabled to save in-memory tables to a remote HDFS cluster. By default, the server does not attempt to save files in a remote HDFS cluster.

Default FALSE

**ENCRYPTIONDOMAIN="string"**

specifies the name for a collection of data that is stored with a common encryption password.

**ENCRYPTIONPASSWORD="string"**

specifies a password for encrypting or decrypting stored data.
HADOOPHOME="path"
specifies the fully qualified path to the directory for the Hadoop software. An example could be /hadoop/hadoop-2.7.1 or a path that includes distribution information such as /opt/cloudera/parcels/CDH-5.14.2-1.cdh5.14.2.p0.3/1ib/hadoop. Specify the HADOOPNAMENODE= argument as well.

HADOOPNAMENODE="host-name"
specifies the host name of the NameNode for the Hadoop cluster. Specify the HADOOPHOME= argument as well.

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.

Requirement The SRCTYPE= argument is required.

Details
Preserving SASHDAT Block Size
The following information applies to backing up SASHDAT files or copying them under the following circumstances:

- Copying from one directory to another directory within a Hadoop cluster.
- Exporting a SASHDAT file from HDFS cluster and then importing the SASHDAT file into HDFS.

If you need to back up SASHDAT files, you must preserve the block size. The following information discusses copying files, but also applies to backing up files.

If you use Hadoop commands to move SASHDAT files, you must preserve the block size. SASHDAT files are stored as several individual blocks in HDFS with a fixed block size. Each block begins with a special SASHDAT header. If you do not preserve the block size when you copy the file, then the block size changes and the blocks do not begin with the SASHDAT header. When the blocks do not begin with the SASHDAT header, the file cannot be read by SAS Cloud Analytic Services.

Example Code 5.1 Determine the Block Size for a SASHDAT
```bash
hdfs dfs -stat '%o' /path/to/file.sashdat
81408
```

Example Code 5.2 Copy a SASHDAT File to a New Directory
```bash
hdfs dfs -D dfs.blocksize=81408 /path/to/file.sashdat /new/hdfs/directory/
```

Example Code 5.3 Export the SASHDAT File to a Local Directory
```bash
hdfs dfs -copyToLocal /path/to/file.sashdat /local/directory/path/
```
Copy the SASHDAT file from /local/directory/path to a location that is accessible to the other Hadoop cluster.

Example Code 5.4 Import the SASHDAT File and Preserve the Block Size
```bash
hdfs dfs -D dfs.blocksize=81408 -copyFromLocal /local/directory/path/filename.sashdat
```
Examples:

Example 1: Add an HDFS Caslib and Specify HADOOPHOME= and HADOOPNAMENODE=

```sas
proc cas;
   accessControl.assumeRole /
      adminRole="superuser";
run;

caslib hdfstmp datasource=(srctype="hdfs" hadoopnamenode="hdfsnn01"
   hadoophome="/hadoop/hadoop-2.7.1/" enableremotesave=true)
   path="/tmp";
```

The preceding example sets the Hadoop NameNode and path to the Hadoop software. The HDFS cluster can be on the same hosts as the server or it can be remote from the server. You must assume an administrative role to add an HDFS caslib and specify the NameNode and Hadoop home directory.

Personal CASUSERHDFS caslibs are not associated with secondary HDFS caslibs that are added this way.

Example 2: Add the /vapublic Directory as an HDFS Data Source for SAS Cloud Analytic Services

```sas
caslib public datasource=(srctype="hdfs") path="/vapublic";
```

Example 3: Load a SASHDAT File into SAS Cloud Analytic Services Using PROC CASUTIL

```sas
proc casutil;
   load casdata="mydata.sashdat" casout="mydata"
      datasourceoptions=(srctype="hdfs" encryptionPassword="secret");
quit;
```

See Also

- SAS Viya for Linux: Deployment Guide
- The SAVE example for the Path data source on page 142 can be performed with an HDFS data source.
Data Source Arguments

PORT=network-port-number
 specifies the publish/subscribe network port number.

SERVER="host-name"
 specifies the host name of the SAS Event Stream Processing server.

SRCTYPE="ESP"
 specifies that the data source is a SAS Event Stream Processing server.

Requirement The SRCTYPE= argument is required.

Authentication Options

AUTHENTICATIONTYPE="KERB" | NONE | OAUTH | SAS
 authenticationType= specifies how to authenticate from SAS Cloud Analytic Services to the ESP server.

KSERVICENAME="string"
 specifies the Kerberos service name.

OAUTHTOKEN="string"
 specifies the OAuth authentication token.

PASSWORD="string"
 specifies the password.

USERNAME="string"
 specifies the user name to use for authenticating to the server.

Details

The CASLIB statement can be used to add or drop a caslib that uses a SAS Event Stream Processing server as a data source. The CASUTIL procedure cannot load data from this type of caslib or save data.

Use the actions in the “Streaming Data Action Set” in SAS Viya: System Programming Guide to transfer data from an ESP window to the CAS server as an in-memory table. After the data transfer, the CASUTIL procedure can be used to manage the in-memory tables and the CAS LIBNAME engine can be used to the in-memory tables.

Example: Add an ESP Server as a Data Source for SAS Cloud Analytic Services

caslib espsrv datasource=(srctype="esp" server="host.example.com" port=55555)
description="SAS ESP Server port 55555"; /* 1*/

proc cas;
builtins.loadActionSet / actionSet="loadStreams"; /* 2*/
    loadStreams.mMetaData / caslib="espsrv"; /* 3*/
run;
The CASLIB statement enables CAS to access an ESP server as a data source. The commonly used publish/subscribe port 55555 is specified as part of the connection.

The loadAction set action is used to enable access to the loadStreams action set in the current CAS session. The Streaming Data (loadStreams) action set is dedicated to accessing data in an ESP server from CAS.

The loadStreams.mMetaData action queries an ESP server to determine the projects, queries, and windows that are available in the ESP server. For the ESP server sample trades project, all the queries and windows are listed.

The loadStreams.loadSnapshot action transfers the current contents of the Trades window to an in-memory table in CAS. The in-memory table name is also Trades.

The CAS LIBNAME engine is used to assign a libref to the Espsrv caslib. The PRINT procedure can then access the in-memory table that is named Trades.

See the following examples for more information about the CAS actions and sample output:

- "Query the Project Metadata" in *SAS Viya: System Programming Guide*
- "Load a Stream Snapshot" in *SAS Viya: System Programming Guide*

---

**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.

Valid in: CASLIB statement, LOAD CASDATA= statement for the CASUTIL procedure, SAVE statement for the CASUTIL procedure

**Syntax**

```cas
CASLIB caslib-reference-name
   DATASOURCE=(SRCTYPE="DNFS" <ENCRIPTIONDOMAIN="string" >
                <ENCRIPTIONPASSWORD="string"> )
                PATH="/directory-path" <caslib-
                statement-options>;
PROC CASUTIL;
   LOAD CASDATA="file-name" CASOUT="table-name"
```
DATASOURCEOPTIONS=(<ENCRYPTIONDOMAIN="string" >
 <ENCRYPTIONPASSWORD="string" >) <load-statement-options>;

PROC CASUTIL;
SAVE CASDATA="table-name" CASOUT="file-name"
DATASOURCEOPTIONS=(<ENCRYPTIONDOMAIN="string" >
 <ENCRYPTIONPASSWORD="string" >) <save-statement-options>;

Data Source Arguments

ENCRIPTIONDOMAIN="string"
specifies the name for a collection of data that is stored with a common encryption password.

ENCRIPTIONPASSWORD="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 customer does not want to deploy a Hadoop cluster for HDFS 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.

Example: Add the /net/fileserver Directory as a DNFS Data Source for SAS Cloud Analytic Services

caslib dnfsds datasource=(srctype="dnfs") path="/net/fileserver/";

See Also

The SAVE example for the Path data source on page 142 can be performed with a DNFS data source.
SAS LASR Analytic Server Data Source

Specifies the connection options for loading data from the SAS LASR Analytic Server into SAS Cloud Analytic Services.

Valid in: CASLIB statement, LOAD CASDATA= statement of the CASUTIL procedure

Syntax

CASLIB caslib-reference-name
   DATASOURCE=(SRCTYPE="LASR" SERVER="host-name" PORT=integer <data-source-options>) <caslib-statement-options>;

PROC CASUTIL;
   LOAD CASDATA="file-name" CASOUT="table-name"
   <DATASOURCEOPTIONS=( data-source-options)>
   IMPORTOPTIONS=(FILETYPE="LASR" file-type-options) <load-statement-options>;

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 5.3 Summary of Arguments for SAS LASR Analytic Server**

<table>
<thead>
<tr>
<th>Argument</th>
<th>Valid Data Source Options in the CASLIB Statement</th>
<th>Valid Import Options in the CASUTIL Procedure</th>
</tr>
</thead>
<tbody>
<tr>
<td>COMPPGM=</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>COMPVARS=</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>FILETYPE=</td>
<td></td>
<td>Required</td>
</tr>
<tr>
<td>METALIB=</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>PARALLELMODE=</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>PASSWORD=</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>PORT=</td>
<td>Required</td>
<td>●</td>
</tr>
<tr>
<td>PRESERVEORDER=</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>SRCTYPE=</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>SERVER=</td>
<td>Required</td>
<td></td>
</tr>
<tr>
<td>SIGNER=</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>TAG=</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>USERNAME=</td>
<td>●</td>
<td>●</td>
</tr>
<tr>
<td>VARCHARS=</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>VARS=</td>
<td>●</td>
<td></td>
</tr>
<tr>
<td>WHERE=</td>
<td>●</td>
<td></td>
</tr>
</tbody>
</table>

### 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 SERVERPARAM statement of the IMSTAT procedure.

The value can also be set with SAS Management Console and then restarting the SAS LASR Analytic Server.

Examples:

Example 1: Add a SAS LASR Analytic Server as a Data Source For SAS Cloud Analytic Services

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.

```sas
caslib valasr datasource=(srctype="lasr"
tag="vapublic"
/* ...additional data source options... */)
```
Specifying the `TAG=` option is recommended.

Example 2: Load SAS LASR Analytic Server Data into SAS Cloud Analytic Services Using PROC CASUTIL

```sas
proc casutil;
  list files; /* 1 */

  load casdata="epa_cars" casout="epa_cars"
      importoptions=(
          filetype="lasr"
          varschars="true"
          vars=("model_year" "vehicle_manufacturer_name") /* 2 */
      );

  contents casdata="epa_cars";
quit;
```

1 The `LIST FILES` statement shows the tables that are in SAS LASR Analytic Server.

2 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.

**Valid in:**
- `CASLIB` statement
- `LOAD CASDATA=` statement for the `CASUTIL` procedure
- `SAVE` statement for the `CASUTIL` procedure

**Syntax**

**CASLIB `caslib-reference-name`**

```
DATASOURCE=(SRCTYPE="PATH" <ENCRYPTIONDOMAIN="string" >
              <ENCRYPTIONPASSWORD="string">) PATH="/directory-path" <caslib-statement-options>;
```

**PROC CASUTIL**

```
LOAD CASDATA="file-name" CASOUT="table-name"
DATASOURCEOPTIONS=(<ENCRYPTIONDOMAIN="string" >
                    <ENCRYPTIONPASSWORD="string">) <options> <load-statement-options>;
```

**PROC CASUTIL**

```
SAVE CASDATA="table-name"
DATASOURCEOPTIONS=(<ENCRYPTIONDOMAIN="string" >
                    <ENCRYPTIONPASSWORD="string">) <options> <save-statement-options>;
```
Data Source Arguments

**ENCRIPTIONDOMAIN="string"**
- specifies the name for a collection of data that is stored with a common encryption password.

**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.

Examples:

Example 1: Add the data01 Directory as a Path Data Source for SAS Cloud Analytic Services

```
caslib pathds datasource=(srctype="path") path="\fileserver\data01";
```

Example 2: Save a SASHDAT File into a Path Data Source Using PROC CASUTIL

```
proc casutil;
  load data=sashelp.class casout="class";
  save casdata="class" casout="male.sashdat" where="sex eq 'M'" datasourceoptions=(encryptionPassword="secretm");
  save casdata="class" casout="female.sashdat" where="sex eq 'F'" datasourceoptions=(encryptionPassword="secretf");
  list files;
  load casdata="female.sashdat" casout="female" datasourceoptions=(encryptionPassword="secretf");
  quit;
libname mycas cas;
proc print data=mycas.female;
run;
```

1 The SAVE statement is used to write the male.sashdat file to directory associated with the active caslib. The ENCRYPTIONPASSWORD= option is used to set the password SecretM.

2 The LOAD CASDATA= statement reads the female.sashdat file from the directory associated with the active caslib and loads it into memory. The in-
memory table is named Female. The ENCRYPTIONPASSWORD= option is used to decrypt the data.

3 The LIBNAME statement assigns a CAS LIBNAME engine libref, Mycas. Because the CASLIB= option is not specified, the active caslib, PathDs, is used. The PRINT procedure can access the in-memory table that is named Female.

---

**S3 Data Source**

Specifies an Amazon Web Services (AWS) S3 location for loading and saving files that the SAS Cloud Analytic Services controller can access.

- **Valid in:** CASLIB statement, LOAD statement for the CASUTIL procedure, SAVE statement for the CASUTIL procedure
- **Applies to:** SASHDAT files and CSV files
- **Restriction:** You cannot save files larger than 5 TB to the AWS S3 environment. The largest object currently allowed in the AWS S3 environment is 5 TB.

**Syntax**

```plaintext
CASLIB caslib-reference-name
DATASOURCE=(<AWSCONFIGPATH="string"> <AWSPROFILE="string"> <AWSALT_PROFILE="string"> <AWSACCESSKEYID="string"> <SECRETACCESSKEY="string"> <SESSIONTOKEN="string"> <KMSCONTEXT="string"> <KMSKEYID="string"> <SSEHEXKEY="string"> <SSETEXTKEY="string"> <OBJETKEY="string"> <USESSL=TRUE | FALSE> <USEIAMROLE=TRUE | FALSE>);}

PROC CASUTIL;
LOAD CASDATA="file-name" CASOUT="table-name"
<DATASOURCEOPTIONS=(<ACCESSKEYID="string" SECRETACCESSKEY="string" <SESSIONTOKEN="string"> <ENCRYPTIONDOMAIN="string"> <ENCRYPTIONPASSWORD="string"> <USESSL=TRUE | FALSE> <USEIAMROLE=TRUE | FALSE>)> <load-statement-options>;

PROC CASUTIL;
SAVE CASDATA="table-name" CASOUT="file-name"
<DATASOURCEOPTIONS=(<ENCRYPTIONDOMAIN="string"> <ENCRYPTIONPASSWORD="string"> )> <save-statement-options>;
```

**Options for the S3 Data Source**

- **ACCESSKEYID="string"** specifies the access key ID that you use to access the Amazon Web Services (AWS) S3 environment.

  **Requirement** This option is required for the S3 data source in the CASLIB statement, unless you specify USEIAMROLE=.
AWSCONFIGPATH="string"
specifies the path and file name of an AWS CLI configuration file that contains
connection parameters to access AWS S3.

Default .aws/config

AWSCONFIGPROFILE="string"
specifies a named profile to use from an AWS CLI config files instead of the
default profile.

AWSCREDENTIALSPATH="string"
specifies the path and file name of an AWS CLI credentials file.

Default .aws/credentials

AWSCREDENTIALSPROFILE="string"
specifies a named profile to use from an AWS CLI credentials file instead of the
default profile.

BUCKET="bucket-name"
specifies the name of the S3 bucket that you are accessing.

Requirement This option is required for the S3 data source in the CASLIB
statement.

Tip You can use PROC S3 to manipulate data in an S3 bucket.

ENCRYPTIONDOMAIN="string"
specifies the name for a collection of data that is stored with a common
encryption password.

ENCRYPTIONPASSWORD="string"
specifies a password for encrypting or decrypting stored data.

KMSCONTEXT="string"
specifies the AWS Key Management Services encryption context for the
cryptographic operation.

KMSKEYID="string"
specifies the AWS Key Management Services ID.

OBJECTPATH="directory-path"
specifies a subdirectory within an S3 bucket.

REGION="region-value"
specifies the region for the AWS S3 environment that you are accessing. Here
are the supported region values:

- AsiaPacificIndia EU_London
- AsiaPacificSeoul EU_Paris
- AsiaPacificSingapore SA_SaoPaulo
- AsiaPacificSydney US_East_Virginia
- AsiaPacificTokyo US_East_Ohio
- Canada US_Gov
- ChinaBeijing US_Gov_FIPS
- ChinaNingxia US_Standard
- EU_Frankfurt US_West_California
- EU_Ireland US_West_Oregon
Requirement This option is required for the S3 data source in the CASLIB statement.

SECRETACCESSKEY= "string"
specifies the secret access key that you use to access the AWS S3 environment.

Requirement This option is required for the S3 data source in the CASLIB statement, unless you specify USEIAMROLE=.

SESSIONTOKEN= "string"
specifies the session token that you use with ACCESSKEYID= and SECRETACCESSKEY= to access the AWS S3 environment.

A session token is used typically with federated accounts. Do not specify this option if your credentials do not include a session token.

SRCTYPE= "S3"
specifies that the data source is an Amazon Web Services S3 environment.

Requirement This option is required for the S3 data source in the CASLIB statement.

SSEHEXKEY= "string"
specifies an encryption key in the form of a 64-digit hexadecimal value.

SSETEXTKEY= "string"
specifies a plain-text version of the key to use with server-side encryption.

USEIAMROLE= TRUE | FALSE
specifies whether to use credentials that are associated with an IAM role. The IAM role is associated with an EC2 instance on an S3 platform.

Interaction If you specify USEIAMROLE=, then do not specify ACCESSKEYID=, SECRETACCESSKEY=, or SESSIONTOKEN=.

Default FALSE

USESSL= TRUE | FALSE
specifies whether to use an SSL connection to an S3 bucket.

Default TRUE

Example: Add the S3 Data Source to the Mys3 Caslib

caslib mys3 datasource=(srctype="s3"
    accesskeyid="LASKDJFLKJE9EXAMPLE"
    secretaccesskey="kJflkusXRrtUMi/M8zbWofa/bPxrGoaKDEXAMPLE"
    region="EU_Ireland"
    bucket="myBucket"
    objectpath="/asr/projB"
    usessl=true);
Platform File Types

Dictionary

Delimited Files (CSV)
Specifies the file type options for loading data from delimited files.
Valid in: LOAD CASDATA= statement for the CASUTIL procedure
Example: Load a Latin1 encoded CSV file into SAS Cloud Analytic Services.

proc casutil;
  load casdata="iris.csv" casout="iris"
    importoptions=(filetype="csv" encoding="latin1");
run;

Syntax

PROC CASUTIL;
  LOAD CASDATA="file-name" CASOUT="table-name"
  IMPORTOPTIONS=(FILETYPE="CSV" file-type-arguments) <load-statement-options>;

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.
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
- S3

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.

**Input Data 6.1 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 6.1 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 6.1  Column Information from the CONTENTS Statement**

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
<th>Format Width</th>
<th>Format Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>sexe</td>
<td>varchar</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>nom</td>
<td>varchar</td>
<td>7</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>âge</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>la taille</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>poids</td>
<td>double</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

**Output 6.2  Three Rows from the Class_Fr Table**

<table>
<thead>
<tr>
<th>Obs</th>
<th>sexe</th>
<th>nom</th>
<th>âge</th>
<th>la taille</th>
<th>poids</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Masculin</td>
<td>André</td>
<td>14</td>
<td>69.0</td>
<td>112.5</td>
</tr>
<tr>
<td>2</td>
<td>Masculin</td>
<td>Benoît</td>
<td>14</td>
<td>63.5</td>
<td>102.5</td>
</tr>
<tr>
<td>3</td>
<td>Masculin</td>
<td>Kévin</td>
<td>12</td>
<td>57.3</td>
<td>83.0</td>
</tr>
</tbody>
</table>

### Parquet Files

Specifies the file type options for loading data from Parquet files.

Valid in:  CASLIB statement, LOAD CASDATA= statement for the CASUTIL procedure, SAVE statement for the CASUTIL procedure

**Note:** Support for Parquet was added in SAS Viya 3.5.

### Syntax

```
CASLIB caslib-reference-name
   DATASOURCE=(SRC_TYPE="PARQUET" <ENCRYPTIONDOMAIN="string">
               <ENCRYPTIONPASSWORD="string">
               PATH="/directory-path" <caslib-statement-options>);
PROC CASUTIL;
LOAD CASDATA="file-name" CASOUT="table-name"
<DATASOURCEOPTIONS=(<ENCRYPTIONDOMAIN="string">
                    <ENCRYPTIONPASSWORD="string">)>>
```
IMPORTOPTIONS=(FILETYPE="PARQUET" file-type-arguments) <load-statement-options>

PROC CASUTIL;
SAVE CASDATA="table-name" CASOUT="file-name"
<DATASOURCEOPTIONS=(<ENCRYPTIONDOMAIN="string">)
<ENCRYPTIONPASSWORD="string">) <save-statement-options>;

Options for the Parquet File Type

ENCRIPTIONDOMAINT="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.

ENCRIPTIONPASSWORD="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.

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.

FILETYPE="PARQUET"
specifies the file type.

Requirement The FILETYPE= argument is required.

SRCTYPE="PARQUET"
specifies that the data source is a Parquet file.

Details

Parquet File Types for Reads
Parquet files can be read from caslibs with these platform file types.

- DNFS
- PATH
- S3
Instead of specifying the `ENCRYPTIONPASSWORD=` option on a file-by-file basis, you can use it as a file type source option when you add a caslib. For an example, see Encrypt Tables in a Caslib on page 462.

Supported Parquet Data Types

The table below shows the data types that you can load from Parquet into CAS. It also shows the resulting data type for the data after it has been loaded into CAS, including the mapping between Parquet native and logical types with those in CAS. (Parquet logical data types extend the various Parquet native data types.) The length of the data format in CAS is based on the length of the source data.

Table 6.1  Supported Amazon Redshift Data Types

<table>
<thead>
<tr>
<th>Parquet Native Data Type</th>
<th>Parquet Logical Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character Data Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE_ARRAY</td>
<td>UTF8</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>FLBA¹</td>
<td>NONE</td>
<td>CHAR</td>
</tr>
<tr>
<td>Numeric Data Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>BYTE_ARRAY⁴</td>
<td>NONE</td>
<td>BINARY</td>
</tr>
<tr>
<td>BYTE_ARRAY</td>
<td>NONE</td>
<td>VARBINARY</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INT32</td>
<td>INT_32</td>
<td>INT32</td>
</tr>
<tr>
<td>INT64</td>
<td>INT_64</td>
<td>INT64</td>
</tr>
<tr>
<td>Date and Time Data Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>INT32</td>
<td>DATE</td>
<td>DATE</td>
</tr>
<tr>
<td>INT64</td>
<td>TIME_MICROS²</td>
<td>TIME</td>
</tr>
<tr>
<td>INT64</td>
<td>TIMESTAMP_MICROS³</td>
<td>DATETIME</td>
</tr>
</tbody>
</table>

1 Fixed_LENGTH_BYTE_ARRAY
2 Parquet TIME values are UNIX Epoch. The Parquet interface to CAS adjusts to or from the SAS Epoch.
3 Parquet TIMESTAMP values are UNIX Epoch. The Parquet interface to CAS adjusts to or from the SAS Epoch.
4 The Parquet FIXED_LENGTH_BYTE_ARRAY cannot currently be annotated with a logical data type that differentiates between a UTF8 string and binary data. Therefore, when writing a .parquet data set, all CAS table columns of the BINARY data type must be converted to a Parquet BYTE_ARRAY with a logical type of NONE. When such .parquet data sets are read back into CAS, the column type is VARBINARY.
Examples:

Example 1: Add a PATH Caslib to Access Parquet Files in a Local or Shared Network Storage Location

```plaintext
caslib mycaslib desc='Parquet Caslib'
  datasource=(srctype='path'
    path="/mytest/customer");
```

Example 2: Load an Encrypted Parquet File into SAS Cloud Analytic Services

```plaintext
proc casutil;
  load casdata="cars.parquet";
run;
```

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.

```plaintext
proc casutil;
  load casdata="cars.sashdat"
    importoptions=(filetype="hdat" encryptionPassword="pasquotank");
run;
```

**Syntax**

```plaintext
PROC CASUTIL;
  LOAD CASDATA="file-name" CASOUT="table-name"
  IMPORTOPTIONS=(FILETYPE="HDAT" file-type-arguments) <load-statement-options>;
```

**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
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 462.

See Also

The SAVE example for the Path data source on page 142 shows how to work with SASHDAT files.
Data Connectors

Working with SAS Data Connectors ........................................... 156
  Accessing and Loading Data ............................................. 156
  Case Sensitivity with Object Names ................................... 156
  Security Considerations .................................................. 156
  Data Type Considerations ............................................... 156

Quick Reference for Data Connector Syntax ............................. 157

Where to Specify Data Connector Options ............................... 160

Using Wildcards for Pattern Matching .................................... 161
  Overview of Wildcards .................................................... 161
  Using Wildcards with SAS Macros ...................................... 161

Using Data Transfer Modes with Data Connectors for DBMS Data Sources ......................................................... 162
  Overview ................................................................. 162
  How to Implement Data Transfer Modes ............................... 162
  Where to Specify Data Transfer Modes ............................... 162
  About Loading Data When Using Multinode Data Transfer ...... 162

Dictionary ............................................................................. 164
  Amazon Redshift Data Connector ........................................ 164
  DB2 Data Connector ......................................................... 179
  Google BigQuery Data Connector ....................................... 187
  Hadoop Data Connector .................................................... 194
  Impala Data Connector ..................................................... 211
  JDBC Data Connector ....................................................... 221
  Microsoft SQL Server Data Connector ............................... 228
  MongoDB Data Connector .................................................. 237
  MySQL Data Connector ..................................................... 248
  ODBC Data Connector ...................................................... 257
  Oracle Data Connector ...................................................... 264
  ORC Data Connector ......................................................... 272
  PC Files Data Connector .................................................... 276
  PostgreSQL Data Connector .............................................. 280
  Salesforce Data Connector .................................................. 290
  SAP HANA Data Connector ............................................... 298
  SAS Data Sets Data Connector .......................................... 305
  Snowflake Data Connector ................................................. 310
  Spark Data Connector ....................................................... 323
  SPD Engine Files Data Connector ...................................... 337
  Teradata Data Connector .................................................... 349
  Vertica Data Connector ..................................................... 362

155
Working with SAS Data Connectors

Accessing and Loading Data

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.

Case Sensitivity with Object Names

How a data source supports case sensitivity is reflected in how its corresponding data connector supports case sensitivity.

In general, for all data connectors that support case sensitivity, the data connector quotes everything it passes down. Therefore, if a data source supports case sensitivity, everything is case sensitive, and its data connector has no option to turn off case sensitivity.

For a data source such as Microsoft SQL Server that does not support case sensitivity, case also does not matter for the Microsoft SQL Server data connector. Although Teradata is case insensitive with respect to object name searches, the original case in which the object was saved is retained for display.

Security Considerations

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.
Data Type Considerations

If you save the contents of a CAS table to a table in an external data source, the resulting database table might contain different data types than the CAS table. Any formats that have been applied in SAS Viya that cannot be mapped to an equivalent format in your data source are not preserved. Verify that your data is what you expect after writing data from SAS Viya to a DBMS or other data source. For specifics about data type conversions between SAS Viya and your data source, refer to the data type information for your data connector.

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 447.

<table>
<thead>
<tr>
<th>Data Connector</th>
<th>srcType=Type</th>
<th>Option Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Amazon Redshift</td>
<td>redshift</td>
<td>“Amazon Redshift Data Connector” on page 164</td>
<td>caslib RScaslib desc='Amazon Redshift Caslib'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataSource={srctype='redshift', server='RSserver', username='user1', password='myPwd', database='rsdatabase'};</td>
<td></td>
</tr>
<tr>
<td>Cloud Data Exchange</td>
<td>cde</td>
<td>“Data Agent CAS Library Options” in Cloud Data Exchange for SAS Viya: Administrator's Guide</td>
<td>action addCaslib / caslib=&quot;cde_caslib&quot;</td>
</tr>
<tr>
<td></td>
<td></td>
<td>datasourc={srcType=&quot;clouddex&quot;, username=&quot;myuser1&quot;, password=&quot;myPwd&quot;, port=&quot;1337&quot;, server=&quot;data-agent-host.url&quot;, catalog=&quot;GRIDLIB&quot;, schema=&quot;model&quot; conopts=&quot;dsn=dsn_tera&quot;};</td>
<td></td>
</tr>
<tr>
<td>DB2</td>
<td>db2</td>
<td>“DB2 Data Connector” on page 179</td>
<td>caslib mycaslib desc='DB2 Caslib'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>datasource={srctype='db2' username='myusr1' password='myPwd' database=&quot;sample&quot;};</td>
<td></td>
</tr>
<tr>
<td>Data Connector</td>
<td>srcType</td>
<td>Option Syntax</td>
<td>Example</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------</td>
<td>---------------------------------------------------</td>
<td>------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Google BigQuery</td>
<td>bigquery</td>
<td>“Google BigQuery Data Connector”</td>
<td>caslib gbqcaslib desc='Google BigQuery Caslib'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataSource=(srctype='bigquery'</td>
<td>credfile='~/userfiles/BigQuery/xxx-yyyy-8e99c10a22537.json' project='project1');</td>
</tr>
<tr>
<td>Hadoop</td>
<td>hadoop</td>
<td>“Hadoop Data Connector” on page 194</td>
<td>caslib hvcaslib desc='Hadoop Caslib'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataSource=(srctype='hadoop', dataTransferMode='serial', hadoopjarpath='/data/cdh58/sdm/lib', hadoopconfigdir='/data/cdh58/sdm/conf', username='myuser1', server='hive01.example.com', schema='default');</td>
<td></td>
</tr>
<tr>
<td>Impala</td>
<td>impala</td>
<td>“Impala Data Connector” on page 211</td>
<td>caslib imcaslib desc='Impala Caslib'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataSource=(srctype='impala', username='myuser1', server='impala01.example.com');</td>
<td></td>
</tr>
<tr>
<td>JDBC</td>
<td>jdbc</td>
<td>“JDBC Data Connector” on page 221</td>
<td>caslib mycaslib desc='JDBC Caslib'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataSource=(srctype='jdbc', url='jdbc:myDriver://&lt;connection-options&gt;', username='myuser1', password='myPwd', class='JDBC-driver-class', classpath='&lt;path-to-JDBC-driver-class&gt;');</td>
<td></td>
</tr>
<tr>
<td>Microsoft SQL Server</td>
<td>sqlserver</td>
<td>“Microsoft SQL Server Data Connector” on page 228</td>
<td>caslib mysqlcaslib desc='Microsoft SQL Server Caslib'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataSource=(srctype='sqlserver', username='myuser1', password='myPwd', sqlserver_dsn='SQLserverDSN-name');</td>
<td></td>
</tr>
<tr>
<td>MongoDB</td>
<td>mongodb</td>
<td>“MongoDB Data Connector” on page 237</td>
<td>caslib mycaslib desc='MongoDB Caslib'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataSource=(srctype='mongodb', server='mymachine.com', username='myuser1', password='myPwd', db='database-name', schema_collection='mySchema');</td>
<td></td>
</tr>
<tr>
<td>MySQL</td>
<td>mysql</td>
<td>“MySQL Data Connector” on page 248</td>
<td>caslib mycaslib desc='MySQL Caslib'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataSource=(srctype='mysql', host='mySQLserver', username='myuser1', password='myPwd', database='&lt;database-name&gt;', schema='myschema');</td>
<td></td>
</tr>
<tr>
<td>ODBC</td>
<td>odbc</td>
<td>“ODBC Data Connector” on page 257</td>
<td>caslib mycaslib desc='ODBC Caslib'</td>
</tr>
<tr>
<td></td>
<td></td>
<td>dataSource=(srctype='odbc', username='myuser1', password='myPwd', database='dbodbc');</td>
<td></td>
</tr>
<tr>
<td>Data Connector</td>
<td>srcType=</td>
<td>Option Syntax</td>
<td>Example</td>
</tr>
<tr>
<td>----------------</td>
<td>----------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>Oracle</td>
<td>oracle</td>
<td>“Oracle Data Connector” on page 264</td>
<td>caslib mycaslib desc=&quot;Oracle Caslib&quot; datasync=(srctype=&quot;oracle&quot;, username=&quot;myuser1&quot;, password=&quot;myPwd&quot;, path=&quot;/machine.lnx.com:5570/exadat&quot;);</td>
</tr>
<tr>
<td>ORC</td>
<td>path</td>
<td>“ORC Data Connector” on page 272</td>
<td>caslib mycaslib desc=&quot;ORC Caslib&quot; datasync=(srctype=&quot;path&quot; path=&quot;/mytest/customer&quot;);</td>
</tr>
<tr>
<td>PC Files</td>
<td>path</td>
<td>“PC Files Data Connector” on page 276</td>
<td>caslib mycaslib desc='PCFiles Caslib' datasync=(srctype='path' filetype='xls' path=&quot;/path/mytest/customers.xls&quot; getnames='true' ranges=&quot;A10-D14&quot;);</td>
</tr>
<tr>
<td>PostgreSQL</td>
<td>postgres</td>
<td>“PostgreSQL Data Connector” on page 280</td>
<td>caslib mycaslib desc='PostgreSQL Caslib' datasync=(srctype='postgres' server='PGserver' username='myuser1' password='myPwd' database=&quot;PGdatabase-name&quot;);</td>
</tr>
<tr>
<td>Salesforce</td>
<td>sforce</td>
<td>“Salesforce Data Connector” on page 290</td>
<td>caslib mycaslib desc='Salesforce Caslib' datasync=(srctype='sforce' username='myuser1' password='myPwd&quot;);</td>
</tr>
<tr>
<td>SAP HANA</td>
<td>hana</td>
<td>“SAP HANA Data Connector” on page 298</td>
<td>caslib hacaslib sessref=mysess datasync=(srctype='hana', server='hana server', username='myuser1', password='myPwd', tableType='column' schema='hana schema' instance='00');</td>
</tr>
<tr>
<td>SAS Data Sets</td>
<td>path</td>
<td>“SAS Data Sets Data Connector” on page 305</td>
<td>caslib mycaslib desc='SASDataSets Caslib' datasync=(srctype='path' path=&quot;/mytest/customer&quot;);</td>
</tr>
<tr>
<td>Snowflake</td>
<td>snowflake</td>
<td>“Snowflake Data Connector”</td>
<td>caslib snowcaslib desc='Snowflake Caslib' datasync=(srctype='snowflake' server='SFserver' username='myuser1' password='myPwd' database=&quot;sfdatabase-name&quot;);</td>
</tr>
<tr>
<td>Data Connector</td>
<td>srcType=</td>
<td>Option Syntax</td>
<td>Example</td>
</tr>
<tr>
<td>---------------</td>
<td>---------</td>
<td>---------------</td>
<td>---------</td>
</tr>
<tr>
<td>SPD Engine Files</td>
<td>spde</td>
<td>“SPD Engine Files Data Connector” on page 337</td>
<td>caslib mycaslib desc='SPD Engine Files Caslib' dataSource=(srctype='spde', hadoopjarpath=&quot;/data/cdh58/sdm/lib&quot;, hadoopconfigdir=&quot;/data/cdh58/sdm/conf&quot;, username=&quot;myuser1&quot;, mdfpath=&quot;/user/MyMDFFPath&quot;, hdfs=true, host='default&quot;);</td>
</tr>
<tr>
<td>Teradata</td>
<td>teradata</td>
<td>“Teradata Data Connector” on page 349</td>
<td>caslib TDcaslib desc='Teradata Caslib' datasource=(srctype='teradata' username='myuser1' password='myPwd' server='teradataServer&quot;);</td>
</tr>
<tr>
<td>Vertica</td>
<td>vertica</td>
<td>“Vertica Data Connector” on page 362</td>
<td>caslib VTcaslib desc='Vertica Caslib' datasource=(srctype='vertica' server='VTserver' username='myuser1' password='myPwd' database='VTdatabase-name' schema='myschema&quot;);</td>
</tr>
</tbody>
</table>

**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 statement */
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

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:

<table>
<thead>
<tr>
<th>Character</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>%</td>
<td>matches any number of characters.</td>
</tr>
<tr>
<td>_</td>
<td>matches a single character. You can include more than one '_' in a pattern.</td>
</tr>
</tbody>
</table>
| \         | escapes a wildcard character so that it is treated as a literal character
             instead of a wildcard in a pattern. |
If you have file names 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**: for data connectors, which includes multinode
- **parallel**: for 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 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.

For details about CAS nodes, see these documents.

- [SAS Viya Administration: SAS Cloud Analytic Services](#)
- [SAS Cloud Analytic Services: Fundamentals](#)
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 \texttt{NUMREADNODES=} and \texttt{NUMWRITENODES=} options.

To specify how many nodes to use for serial data transfer using multiple worker nodes, you specify a specific value.

- Set \texttt{NUMREADNODES=} and \texttt{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 \texttt{NUMREADNODES=}0 and \texttt{NUMWRITENODES=}0, then all available nodes are used for multinode data transfer.

Data connect accelerators use the \texttt{DATATRANSFERMODE=} option to specify serial or parallel data transfer. When you specify \texttt{DATATRANSFERMODE=\textquoteright PARALLEL}, data transfer uses parallel mode. When you specify \texttt{DATATRANSFERMODE=\textquoteright 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 \texttt{NUMREADNODES=} and \texttt{NUMWRITENODES=} are different from 1, then data transfer uses multiple nodes to perform the transfer.

Where to Specify Data Transfer Modes

Two different SAS programming options have the same name: \texttt{DATATRANSFERMODE=}.

- You can specify the \texttt{DATATRANSFERMODE=} file import option in such SAS procedures as \texttt{PROC CAS} and \texttt{PROC CASUTIL}. Use this option when you are working with SAS tables (SAS7BDAT files) as specified by the \texttt{PATH} type of \texttt{CASLIB}. For SAS tables, the default value is \texttt{DATATRANSFERMODE=AUTO}. For any other file types, \texttt{DATATRANSFERMODE=} is not an available option, so the result is effectively the same as the \texttt{SERIAL} value.

- You can specify the dataTransferMode= caslib attribute with the SrcType attribute to specify using optional SAS software. The default \texttt{DATATRANSFERMODE=SERIAL} value specifies that SAS Data Connector technology is used to connect to a supported remote data source. The \texttt{DATATRANSFERMODE=PARALLEL} value specifies that SAS Data Connect Accelerator technology is used to connect to the SAS In-Database Embedded Process in a supported remote data source. For most source types, \texttt{DATATRANSFERMODE=} is not an available option. Therefore, the result is effectively the same as the \texttt{SERIAL} value. For caslibs with a source type of Hadoop, Spark, SPD Engine Files, or Teradata, you can specify the \texttt{DATATRANSFERMODE=} option.
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

Amazon Redshift Data Connector

Enables you to transfer data between Amazon Redshift and CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Notes: All users with SAS/ACCESS Interface to Amazon Redshift can use SAS Data Connector to Amazon Redshift.

Support for this data connector was added in SAS Viya 3.3.

Examples: Establish a connection between your Amazon Redshift database and SAS Cloud Analytic Services.

```sas
caslib redshiftcaslib desc='Amazon Redshift Caslib' dataSource={{srctype='redshift'
                   server='RSserver'
                   username='user1'
                   password='myPwd'
                   database="rsdatabase-name"});
```

Overriding the user and password values.

```sas
proc casutil;
   load casdata="rsexamp" casout="myRSdata" incaslib="redshiftcaslib"
   outcaslib="casuser" dataSourceOptions={
       username='user5'
       password='myPwd'});
quit;
```
Syntax

Data Connector Options for Amazon Redshift

Data connector options are used in the context of different statements that connect your data in Amazon Redshift with CAS. 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 160.

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [see Requirement] PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>AUTHDOMAIN=</td>
</tr>
<tr>
<td>Requirement</td>
<td>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.</td>
</tr>
</tbody>
</table>

**AWSCONFIG=“path-and-AWS-configuration-file-name”**

specifies the location of the Amazon Web Services (AWS) configuration file.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement PROC CASUTIL: SAVE statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>~/.aws/config</td>
</tr>
<tr>
<td>Requirement</td>
<td>To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.</td>
</tr>
</tbody>
</table>

**BLOPTIONS=“option-1<, option-2<, ..., option-N>“**

specifies options to pass to the bulk-load and bulk-unload facility. These options affect how data is transferred and processed.

For bulk loading, any options that you specify are appended to the COPY command that is executed by the S3 bulk load tool.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement PROC CASUTIL: SAVE statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>BULKLOADOPTIONS=</td>
</tr>
</tbody>
</table>
To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

Support for this option was added in SAS Viya 3.4.

BLOPTIONS="ROUNDEC"

specifies the name of the Amazon S3 bucket to use when bulk loading data. The bucket name that you specify must already exist.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default none

To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

Support for this option was added in SAS Viya 3.4.

BUCKET="Amazon-S3-bucket-name"

specifies the name of the Amazon S3 bucket to use when bulk loading data. The bucket name that you specify must already exist.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default none

To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

Support for this option was added in SAS Viya 3.4.

BULKLOAD=TRUE | FALSE

specifies whether to use bulk loading to insert a large number of rows from CAS into an external table.

Valid in CASLIB statement

PROC CASUTIL: SAVE statement

Default FALSE

To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

Support for this option was added in SAS Viya 3.4.

BULKUNLOAD=TRUE | FALSE

specifies whether to retrieve (fetch) a large number of rows from an external table into CAS.

Interaction Bulk unloading works only when BULKUNLOAD= is enabled and when NUMREADNODES= is not set to 1.

Valid in CASLIB statement

Default FALSE

Note Support for this option was added in SAS Viya 3.5.

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
CHARMULTIPLIER=value
specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Default: Active caslib

Valid in
- CASLIB statement
- PROC CASUTIL: LOAD and SAVE statements

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.

Default: UTF-8

Valid in
- CASLIB statement

COMPRESS=TRUE | FALSE
specifies whether to compress data using the GZIP format.

Valid in
- CASLIB statement
- PROC CASUTIL: SAVE statement

Default: FALSE

Requirement: To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

Note: Support for this option was added in SAS Viya 3.4.

CREDENTIALSFILE="path-and-filename"
specifies the path and name of the AWS credentials file to use for bulk loading data.

Valid in
- CASLIB statement
- PROC CASUTIL: SAVE statement
Default: none

**Requirement:** To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

**Note:** Support for this option was added in SAS Viya 3.4.

**CREDSPROFILE=** "profile-name"

specifies the profile name to use when there is more than one profile in the AWS credentials file.

Valid in: CASLIB statement

**Default:** none

**Requirement:** To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

**Note:** Support for this option was added in SAS Viya 3.4.

**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

**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]

**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
DEFAULTDIR="directory-path"
specifies where all temporary files are created during bulk loading.

The value that you specify is prepended onto the file name. Provide the complete directory path, including the directory separator character.

Valid in CASLIB statement
PROC CASUTIL: SAVE statement
Alias DEFAULTDIRECTORY=
Default temporary-file-directory that is specified by the UTILLOC= system option
Requirement To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.
Note Support for this option was added in SAS Viya 3.4.

DELETEDATAFILES=TRUE | FALSE
specifies whether to delete only the data file or all files that are created during bulk loading. The files are deleted from the local machine and from the S3 bucket.

Valid in CASLIB statement
PROC CASUTIL: SAVE statement
Default TRUE
Requirement To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.
Note Support for this option was added in SAS Viya 3.4.

DELIMITER="character"
specifies an override of the default delimiter character that separates columns of data during bulk loading. You might specify a value for DELIMITER= if your character data contains the default delimiter character.

Valid in CASLIB statement
PROC CASUTIL: SAVE statement
Default bell character (ASCII 0x07)
Requirement To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.
Note Support for this option was added in SAS Viya 3.4.

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.

**ESCAPE=** *character* | "OFF"

specifies a single character to use for character escape sequences. Select a character that is not used anywhere in your data.

You can disable the ESCAPE= character setting by specifying ESCAPE="OFF".

**INSERTBUFF**= "number-of-rows"

specifies the number of rows per block of data to save to a table in the data source.

**KEY**= "value"

specifies the Amazon Web Services access key that is used with key-based access control. If you are using temporary token credentials, this is the temporary access key ID.

**NCHARMULTIPLIER**= *value*

specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.
Valid in | CASLIB statement
---|---
PROC CASUTIL: LOAD and SAVE statements

Default | 3

**NUMREADBUFFERS=number**
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in | CASLIB statement
---|---
PROC CASUTIL: LOAD statement

Default | 4
Range | 1-10

**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 lets a data connector 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 | CASLIB statement
---|---
PROC CASUTIL: LOAD statement

Default | 1

**NUMWRITTENODES=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 lets a data connector 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 | CASLIB statement
---|---
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, DELETESOURCE, LIST, LOAD, and SAVE statements
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.

**PROFILE=**"profile-name"

specifies the profile to use when there is more than one profile in the AWS configuration file. When more than one profile is used, the profile name precedes the set of name-value pairs in square brackets. Here is an example configuration file with two profiles:

```
[default]
region=us-west-2
output=text

[analyst]
region=us-east-1
output=json
```

Valid in **CASLIB statement**

**PROC CASUTIL: SAVE statement**

Default default

Requirement To specify this option, you must also set `BULKLOAD=TRUE` or `BULKUNLOAD=TRUE`.

Note Support for this option was added in SAS Viya 3.4.

**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.

**REGION=“region”**
specifies the AWS region from which S3 data is being loaded. Use the regions values that are available for the Amazon S3 service.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: SAVE statement</td>
<td></td>
</tr>
</tbody>
</table>

Default: none

Requirements: To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

You must specify the region when using bulk loading with the Amazon Redshift data connector.

Note: Support for this option was added in SAS Viya 3.4.

**S3CONFIG=“local-path-and-configuration-file-name”**
specifies the location of a local configuration file for Amazon S3. For more information about Amazon S3 configuration files, see the Amazon Web Services documentation.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: SAVE statement</td>
<td></td>
</tr>
</tbody>
</table>

Default: none

Requirements: To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

Note: Support for this option was added in SAS Viya 3.4.

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
<td></td>
</tr>
</tbody>
</table>

Default: none

Interaction: If you specify values for SCHEMA= and DATABASE=, the value for SCHEMA= is used to qualify the table name.
SECRET="secret-access-key"
specifies the secret access key or temporary secret access key that is used to bulk load data.

An Amazon Web Services (AWS) secret access key is associated with the key ID that you specify with the KEY= option.

Valid in | CASLIB statement
---|---
PROC CASUTIL: SAVE statement

Default | none

Requirement | To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

Note | Support for this option was added in SAS Viya 3.4.

Tip | To increase security, you can encode the key value by using PROC PWENCODE.

SERVER="server-identifier"
specifies the server identifier for the DBMS server.

Valid in | CASLIB statement [required]

Default | none

SLICECOLUMN="column-name"
specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

Valid in | CASLIB statement

Default | none

Interaction | Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

SRCTYPE="redshift"
specifies that the data source is an Amazon Redshift database.

Valid in | CASLIB statement [required]

Default | none

TOKEN="temporary-token"
specifies a temporary token that is associated with the temporary credentials that you specify with the KEY= and SECRET= options.

Valid in | CASLIB statement

PROC CASUTIL: SAVE statement
Default none

Requirement To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

Note Support for this option was added in SAS Viya 3.4.

**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

**USESSL=TRUE | FALSE**

specifies whether to use SSL encryption for connections to Amazon S3.

DefaultValue= TRUE

Requirement To specify this option, you must also set BULKLOAD=TRUE or BULKUNLOAD=TRUE.

Note Support for this option was added in SAS Viya 3.4.

**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, DELETESOURCE, LIST, LOAD, and SAVE statements**

Aliases UID=

USER=

Default none
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

Bulk Loading and Unloading Amazon Redshift Data

Bulk loading and unloading is the fastest way to move data between your Amazon Redshift database and the CAS server. To use the bulk-load facility to transfer data to Amazon Redshift, specify BULKLOAD=TRUE. To use the bulk-unload facility to transfer data into CAS, specify BULKUNLOAD=TRUE. Bulk loading and unloading in Amazon Redshift uses the Amazon Simple Storage Services (S3) tool to transfer large amounts of data.

These data connector options are also related to bulk loading or unloading:

- AWSCONFIG=
- BLOPTIONS=
- BUCKET=
- COMPRESS=
- CREDENTIALSFILE=
- CREDENTIALSPROFILE=
- DEFAULTDIR=
- DELETEDATAFILES=
- DELIMITER=
- ESCAPE=
- KEY=
- PROFILE=
- REGION=
- S3CONFIG=
- SECRET=
- TOKEN=
- USESSL=

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 7.2 Supported Amazon Redshift Data Types

<table>
<thead>
<tr>
<th>Amazon Redshift Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>CHAR($n$)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>VARCHAR($n$)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td><strong>Numeric Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>INT64</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DECIMAL($p$, $s$)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INT32</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SERIAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>INT32</td>
</tr>
<tr>
<td><strong>Date and Time Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(formatted as DATEw.)</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP($p$)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(without time zone)</td>
<td>(formatted as DATETIMEw.d)</td>
</tr>
</tbody>
</table>

**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 30.

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";
  contents casdata="%upcase(class_from_RScaslib)" incaslib="casuser"; \3
```

quirt;

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.

DB2 Data Connector

Enables you to transfer data between DB2 and CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Notes: All users with SAS/ACCESS Interface to DB2 can use SAS Data Connector to DB2. Support for this data connector was added in SAS Viya 3.3.

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(db2examp)" incaslib="db2caslib"
   dataSourceOptions=(username='user5' password='pwd5');

Syntax

Data Connector Options for DB2

Data connector options are used in the context of different statements that connect your data in DB2 with CAS. 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 160.

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 the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in  
CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

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.
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
Default  none

Example  conopts="QueryTimeout=0"

DB="database-name"
specifies the database name.

Valid in  CASLIB statement [Required]
Default  none
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

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
Default  Calculated automatically based on row size

INVALIDDATE="ERROR" | "SILENT" | "WARN"
specifies how to treat invalid date values when loading DATE or DATETIME values. Here are the possible values:

ERROR specifies that when an invalid DATE or DATETIME value is detected, loading the record fails, and an error is printed in the SAS log.

SILENT specifies that SAS converts an invalid DATE or DATETIME value to a missing value, and no message is printed in the SAS log.

WARN specifies that SAS converts an invalid DATE or DATETIME value to a missing value and prints a warning in the SAS log.
<table>
<thead>
<tr>
<th>Alias</th>
<th>INVALID_DATE=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid in</td>
<td>CASLIB statement</td>
</tr>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD statement</td>
</tr>
<tr>
<td>Defaults</td>
<td>When loading records individually, the default is ERROR.</td>
</tr>
<tr>
<td></td>
<td>When loading multiple records, the default is WARN.</td>
</tr>
<tr>
<td>Interaction</td>
<td>Use the READBUFF= option to control how many records are retrieved in a Load operation.</td>
</tr>
</tbody>
</table>

**NCHARMULTIPLIER=value**

specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD and SAVE statements</td>
</tr>
<tr>
<td>Default</td>
<td>3</td>
</tr>
</tbody>
</table>

**NUMREADBUFFERS=number**

specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD statement</td>
</tr>
<tr>
<td>Default</td>
<td>4</td>
</tr>
<tr>
<td>Range</td>
<td>1-10</td>
</tr>
</tbody>
</table>

**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 lets a data connector 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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD statement</td>
</tr>
<tr>
<td>Default</td>
<td>1</td>
</tr>
</tbody>
</table>

**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 lets a data connector 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**

<table>
<thead>
<tr>
<th>CASLIB statement</th>
<th>PROC CASUTIL: SAVE statement</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default</strong></td>
<td>1</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>CASLIB statement [see Requirement]</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: LOAD statement</td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>
**SLICECOLUMN="column-name"**
specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

Valid in: CASLIB statement

Default: none

Interaction: Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

**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

**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, DELETESOURCE, LIST, LOAD, and SAVE statements**

**Aliases:** UID=
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 7.3  Supported DB2 Data Types**

<table>
<thead>
<tr>
<th>DB2 Data Type</th>
<th>CAS Data Type</th>
<th>DB2 Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td></td>
<td>Character</td>
<td></td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>CHAR</td>
<td>CLOB</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>GRAPHIC(n)</td>
<td>VARCHAR</td>
<td>LONG VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>LONG VARGRAPHIC</td>
<td>VARCHAR</td>
<td>VARCHAR(n)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>Numeric</td>
<td></td>
<td>Numeric</td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>INT64</td>
<td>DECIMAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>INT32</td>
<td>INTEGER</td>
<td>INT32</td>
</tr>
<tr>
<td>Date and Time</td>
<td></td>
<td>Date and Time</td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DOUBLE</td>
<td>DATE</td>
<td>(formatted as DATEw.)</td>
</tr>
</tbody>
</table>
### DB2 Data Type | CAS Data Type
---|---
TIME | DOUBLE (formatted as TIMEw.d)
TIMESTAMP | DOUBLE (formatted as DATETIMEw.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 30.

**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.

```plaintext
caslib db2lib
    datasource=(srctype='db2',
               username='myuser1',
               password='myPwd',
               db="myDB2data");
```
Example 2: Load DB2 Data into SAS Cloud Analytic Services Using PROC CASUTIL

```sas
proc casutil;
  list tables incaslib="casuser";        \1
  load casdata="MYDB2DATA" incaslib="db2caslib" outcaslib="casuser"
    casout="DB2data_from_db2caslib";      \2
  list tables incaslib="casuser";
  contents casdata="%upcase(class_from_db2caslib)" incaslib="casuser"; \3
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`.
3. Note: You must list DB2 table names with capitalization that matches that in the DB2 database.
4. List the tables in Casuser again to see the newly created table, `DB2data_from_db2caslib`, that you loaded.
5. List information about the newly loaded table, including column names, data types, and so on.

Google BigQuery Data Connector

Enables you to transfer data between Google BigQuery and CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Notes: All users with SAS/ACCESS Interface to Google BigQuery can use SAS Data Connector to Google BigQuery. Support for this data connector was added in SAS Viya 3.4.

Example: Establish a connection between your Google BigQuery database and SAS Cloud Analytic Services.

```sas
caslib gbqcaslib desc='Google BigQuery Caslib'
  dataSource=(srctype='bigquery'
    credfile='/u/authfiles/BigQuery/xxx-yyyy-8e99c10a22537.json'
    project='project1');
```

Syntax

Data Connector Options for Google BigQuery

Data connector options are used in the context of different statements that connect your data in Google BigQuery with CAS. 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 160.

Here are the data connector options for Google BigQuery:

**BULKLOAD=TRUE | FALSE**

specifies whether to use bulk loading to insert a large number of rows from CAS into an external table.

Valid in  
CASLIB statement

Default  
TRUE

**CLIENT_ID="ID-value"**

specifies the client ID value for the OAuth client that is used to connect to Google BigQuery using the OAuth authentication method.

Valid in  
CASLIB statement

Default  
none

Interaction  
Use this option in conjunction with the CLIENT_SECRET= and REFRESH_TOKEN= options.

**CLIENT_SECRET=authentication-secret**

specifies the client secret value for the OAuth client that is used to connect to Google BigQuery using the OAuth authentication method.

Valid in  
CASLIB statement

Default  
none

Interaction  
Use this option in conjunction with the CLIENT_ID= and REFRESH_TOKEN= options.

**CREDFILE="path-and-filename"**

specifies the path and name of the Google Cloud Platform credentials file.

Valid in  
CASLIB statement [required]

Default  
none

**MAX_BINARY_LEN='value'**

specifies the maximum length in bytes to allocate for binary values.

Valid in  
CASLIB statement

Default  
none

**MAX_CHAR_LEN='value'**

specifies the maximum number of characters to allocate for string values.

Valid in  
CASLIB statement
**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 lets a data connector 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 CASLIB statement

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 lets a data connector 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 CASLIB statement

PROC CASUTIL: SAVE statement

Default 1

**PROJECT=**"project-ID"
specifies the project ID for a Google BigQuery project.

Valid in CASLIB statement [required]

PROC CASUTIL: LOAD statement

Default none

Example project='project1'

**PROXY=**
specifies the URL of a proxy server that is used to connect to Google BigQuery.

Default none

Example proxy="mycompany/proxy.htm"

**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
REFRESH_TOKEN="token-value"
specifies the refresh token value for the OAuth client that is used to connect to
Google BigQuery using the OAuth authentication method.

Valid in: CASLIB statement
Default: none
Interaction: This option is used in conjunction with the CLIENT_ID= and
CLIENT_SECRET= options.

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: LOAD statement
Default: none

SLICECOLUMN="column-name"
specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively
small number of distinct values that are fairly evenly distributed. Columns with
continuous numeric values can work as well. SAS uses the MOD function,
applied to the values in the specified column, to assign data to a partition. Each
partition is assigned to a node that performs the Read operation for its slice of
data.

Valid in: CASLIB statement
Default: none
Interaction: Multinode data Read is not on by default. Use NUMREADNODES=
to specify a number.

SRCTYPE="bigquery"
specifies that the data source is a Google BigQuery database.

Valid in: CASLIB statement [required]
Default: none

TABLE_QUALIFIER="qualifier"
specifies a table qualifier when you require a value other than the default project
qualifier.

Valid in: CASLIB statement
### PROC CASUTIL: CONTENTS, LOAD statements

<table>
<thead>
<tr>
<th>Default</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>QUALIFIER=</td>
</tr>
</tbody>
</table>

#### Details

**Supported Google BigQuery Data Types**

The following table shows the data types that can be loaded from Google BigQuery 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 7.4 Supported Google BigQuery Data Types**

<table>
<thead>
<tr>
<th>Google BigQuery Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>STRING</td>
<td>VARCHAR</td>
</tr>
<tr>
<td></td>
<td>The default length is based on MAX_CHAR_LEN.</td>
</tr>
<tr>
<td><strong>Numeric Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>FLOAT64</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INT64</td>
<td>INT64</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>DOUBLE</td>
</tr>
<tr>
<td><strong>Date and Time Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td></td>
<td>(formatted as DATEw.)</td>
</tr>
<tr>
<td>DATETIME</td>
<td>DOUBLE</td>
</tr>
<tr>
<td></td>
<td>(formatted as DATETIMEw.d)</td>
</tr>
<tr>
<td>TIME</td>
<td>DOUBLE</td>
</tr>
<tr>
<td></td>
<td>(formatted as TIMEw.)</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DOUBLE</td>
</tr>
<tr>
<td></td>
<td>(formatted as DATETIMEw.d)</td>
</tr>
<tr>
<td><strong>Other Data Types</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Google BigQuery Data Type

<table>
<thead>
<tr>
<th>Google BigQuery Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY</td>
<td>VARCHAR</td>
</tr>
<tr>
<td></td>
<td>The default length is based on MAX_CHAR_LEN.</td>
</tr>
<tr>
<td>GEOGRAPHY</td>
<td>VARCHAR</td>
</tr>
<tr>
<td></td>
<td>The default length is based on MAX_CHAR_LEN.</td>
</tr>
<tr>
<td>RECORD(^1)</td>
<td>Varies, based on the data type for the data in the RECORD object.</td>
</tr>
</tbody>
</table>

\(^1\) Also referred to as a STRUCT data type.

### 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 30.

### 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.

### 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 30.
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 Google BigQuery Caslib and Load Data into SAS Cloud Analytic Services

Use the CASLIB statement to establish a connection between your Google BigQuery source data and a caslib, BQcaslib. All of the options supplied in this example are required in the CASLIB statement, except the SCHEMA= option.

In this example, the Google BigQuery data is stored at the location designated in the credentials file.

```
caslib BQcaslib desc='Google BigQuery Caslib'
   dataSource=(srctype='bigquery',
               credfile="/u/myuser/authfiles/BigQuery/xxx-yyyy-8e99c10a22537.json"
               project="projectA"
               schema="mySchema");
```

```
proc casutil;
  list files incaslib="BQcaslib";
load casdata="myBQdata" outcaslib="BQcaslib" incaslib="casuser"
   casout="class_from_BQcaslib";
list files incaslib="casuser";
contents casdata="%upcase(class_from_BQcaslib)" incaslib="casuser";
quit;
```

1 Add a caslib, BQcaslib, that connects your Google BigQuery database with CAS.
2 List the tables in BQcaslib before loading your data.
3 Load the table myBQdata from Casuser into caslib BQcaslib. Call the new CAS table class_from_BQcaslib.

Note: You must specify Google BigQuery table names using the capitalization that is used in the database.

4 List the tables in Casuser to see the newly created CAS table, class_from_BQcaslib, that you loaded.
List information about the newly loaded table, including column names, data types, and so on.

Example 2: Load RECORD Data into CAS

This example shows how to load a Google BigQuery table called Sample that contains records. Although VARLIST= is not a data connector option, it is helpful to show how to refer to record data.

```
caslib gbqlib desc='Google BigQuery Caslib'
   dataSource=(
      srctype='bigquery'
      credfile='/u/authfiles/BigQuery/xxx-yyyy-8e99c0a22537.json'
      project='project1');    /* 1 */

proc casutil;
   contents casdata="sample" incaslib="gbqlib";        /* 2 */
   load casdata="mySample" incaslib="gbqlib" outcaslib="casuser" casout="sample"
      varlist={"Sample.StringType"};     /* 3 */
quit;
```

1. Add a caslib, Gbqlib, that connects your Google BigQuery database with CAS.
2. List the contents of the Sample data set.
3. Load the StringType record from the Sample table in Gbqlib into caslib Casuser. Call the new CAS table MySample.

---

**Hadoop Data Connector**

Enables you to load data from Hive into 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:** To transfer data in parallel requires a separate license to SAS In-Database Technologies for Hadoop and a massively parallel processing (MPP) system. 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.

```
echo JAVA_HOME=/usr/java/latest/jre
export LD_LIBRARY_PATH=$JAVA_HOME/lib/amd64/server:$LD_LIBRARY_PATH
```

- **Supports:** Hadoop, HDMD, Spark
- **Note:** All users with SAS/ACCESS Interface to Hadoop can use the serial SAS Data Connector to Hadoop.
- **Tip:** 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.
- **Examples:** Load Hadoop source data and add a caslib to it.

```
caslib hvcaslib sessref=mysess
   dataSource=(srctype='hadoop',
```
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

Data connector options are used in the context of different statements that connect your data in Hadoop with CAS. 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 160.

**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 | "nB" | "nK" | "nM" | "nG" | "nT"**

specifies the buffer size that is used to send data to or receive data from SAS embedded processes. If you set this value in PROC CASUTIL, then this value
overrides a value that you set in the CASLIB statement. Increasing the size might result in better performance with a trade-off of increased memory usage.

You can specify this value as an integer (bytes) or with a suffix, such as "4M" to indicate four megabytes. Supported suffixes are "B" (bytes), "K" (kilobytes), "M" (megabytes), "G" (gigabytes), and "T" (terabytes). Suffixes are case insensitive.

Valid in    CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default 1048576

Restriction This option applies only with parallel data transfer for Hadoop (DATATRANSFERMODE="parallel").

Requirement You must specify a value that is greater than 0.

Supports HDMD, Hive, parallel data transfer

Tip If you experience memory errors during data transfer, reduce the value of BUFFERSIZE=.

**BULKLOAD=TRUE | FALSE**

specifies whether to use bulk loading to insert a large number of rows from CAS into an external table.

Valid in    CASLIB statement

PROC CASUTIL: SAVE statement

Default TRUE

Note Support for Hadoop was added in SAS Viya 3.5.

Restriction Support for this option applies only when DATATRANSFERMODE='serial'.

**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 the number of bytes per character 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 option applies to data that is encoded using a single-byte character set. Because all Hive tables are UTF-8 tables, this value does not affect Hive.

Valid in    CASLIB statement
# PROC CASUTIL: LOAD and SAVE statements

<table>
<thead>
<tr>
<th>Default</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>1.0–5.0</td>
</tr>
<tr>
<td>Supports</td>
<td>HDMD, parallel data transfer, serial data transfer</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
</table>

## PROC CASUTIL: CONTENTS, LIST, LOAD statements

<table>
<thead>
<tr>
<th>Default</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example</td>
<td>conopts=&quot;QueryTimeout=0&quot;</td>
</tr>
</tbody>
</table>

**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 or save the data in parallel using embedded processing. If this fails, a note is written to the SAS log and serial processing is then attempted.
- **PARALLEL** specifies to load or save the data in parallel by using the SAS Data Connect Accelerator to your database or data source.
- **SERIAL** specifies to load or save the data serially by using the SAS Data Connector to your database or data source.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
</table>

**Aliases**  
DATATRANSFER=  
DTM=

<table>
<thead>
<tr>
<th>Default</th>
<th>SERIAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>Restriction</td>
<td>The DATATRANSFERMODE= option that is part of a caslib definition is not the same as the DATATRANSFERMODE= file import option. Although they share the same name and values, they are functionally different and therefore are not interchangeable. See “Where to Specify Data Transfer Modes” for details.</td>
</tr>
<tr>
<td>Requirement</td>
<td>To use the PARALLEL option, you must have a license for the SAS Data Connect Accelerator for Hadoop, which is part of SAS In Database Technologies.</td>
</tr>
</tbody>
</table>
Interaction
If NUMREADNODES= or NUMWRITENODES= is a value other than 1 and DATATRANSFERMODE="AUTO", SAS attempts first to transfer the data in parallel. 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 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 for all Hive string data types. For example, if dbMaxText=100, all Hive string data types are loaded as VARCHAR(100).

Valid in
CASLIB statement

PROC CASUTIL: LOAD statement

Alias
DBMAX_TEXT=

Default
1024

Range
1 (minimum)

Supports
Hive, parallel data transfer, serial data transfer

**HADOOPCONFIGDIR="configuration-files-directory"**
specifies the Hadoop configuration files directory. These are the configuration files that are obtained by running the Hadoop tracer tool on the target cluster.

Valid in
CASLIB statement

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 you can obtain by running the Hadoop tracer tool on the target cluster. These files are delimited by colons for Linux.

If you specify PLATFORM="spark", you must include the Spark JAR files in the hadoopJarPath—after the path to the Hadoop JAR files if you are running Apache Hadoop and before the path to the Hadoop JAR files for Cloudera Hadoop.

If you are accessing Hive tables that require a customized Hive SerDe, you must include the SerDe JAR files in the hadoopJarPath.

Valid in
CASLIB statement [required]
**HDFSDATADIR=**"Hadoop-HDFS-data-directory"

specifies the Hadoop HDFS directory to use to store table data when saving a CAS table to HDMD.

<table>
<thead>
<tr>
<th>Default</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>HDMD, Hive, parallel data transfer, serial data transfer</td>
</tr>
</tbody>
</table>

Valid in

CASLIB statement

PROC CASUTIL: SAVE statement

**HDFSMETADIR=**"Hadoop-HDFS-metadata-directory"

specifies the Hadoop HDFS directory to use that contains one or more SASHMDM files.

<table>
<thead>
<tr>
<th>Default</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>HDMD, parallel data transfer, serial data transfer</td>
</tr>
</tbody>
</table>

Valid in

CASLIB statement

PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements

**HDFSTEMPDIR=**"Hadoop-HDFS-temporary-directory"

specifies the Hadoop HDFS directory to use to store temporary data.

<table>
<thead>
<tr>
<th>Default</th>
<th>&quot;/TMP&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>HDMD, Hive, parallel data transfer, serial data transfer</td>
</tr>
</tbody>
</table>

Valid in

CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

**INSERTBUFF="number-of-rows"**

specifies the number of rows per block of data to save to a table in the data source.

<table>
<thead>
<tr>
<th>Default</th>
<th>Calculated automatically based on row size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>HDMD, Hive, serial data transfer</td>
</tr>
</tbody>
</table>

Valid in

CASLIB statement

PROC CASUTIL: SAVE statement

**MAXLOADTHREADS="maximum-number-of-threads"**

specifies the maximum number of threads to use per worker node.

<table>
<thead>
<tr>
<th>Default</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Supports</td>
<td>HDMD, Hive, serial data transfer</td>
</tr>
</tbody>
</table>

Valid in

CASLIB statement

PROC CASUTIL: LOAD statement
**MAXSAVETHREADS=*/maximum-number-of-threads*/

specifies the maximum number of threads to use per worker node.

*Valid in*
- CASLIB statement
- PROC CASUTIL: SAVE statement

*Default*
24

*Supports*
HDMD, Hive, parallel data transfer

**NCHARMULTIPLIER=*/value*/

specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

*Valid in*
- CASLIB statement
- PROC CASUTIL: LOAD and SAVE statements

*Default*
3

*Supports*
HDMD, Hive, parallel data transfer, serial data transfer

**NUMREADBUFFERS=*/number*/

specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

*Valid in*
- CASLIB statement
- PROC CASUTIL: LOAD statement

*Default*
4

*Range*
1-10

*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 lets a data connector 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*
- CASLIB statement
- 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 lets a data connector 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
- CASLIB statement
- PROC CASUTIL: SAVE statement

Default
1

Supported platforms
- 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, DELETESOURCE, 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.

Supported platforms
- HDMD, Hive, parallel data transfer, serial data transfer

PLATFORM="mapred" | "spark"
specifies the platform on which to run the SAS Embedded Process job as a MapReduce job or on Spark.

Valid in
- CASLIB statement
- PROC CASUTIL: LOAD and SAVE statements

Default
mapred

Supported platforms
- HDMD, Hive, parallel data transfer
**PORT**=<em>port value</em>
specifies the Hive JDBC port number.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>10000</td>
</tr>
<tr>
<td>Range</td>
<td>1–65535</td>
</tr>
<tr>
<td>Supports</td>
<td>Hive, parallel data transfer, serial data transfer</td>
</tr>
</tbody>
</table>

**POST_TABLE_OPTS**=
allows additional database-specific options to be placed after the table name when creating a table.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Supports</td>
<td>Hive, serial data transfer</td>
</tr>
</tbody>
</table>

**PRE_TABLE_OPTS**=
allows additional database-specific options to be placed before the table name when creating a table.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Supports</td>
<td>Hive, serial data transfer</td>
</tr>
</tbody>
</table>

**PROPERTIES**="<em>Hive-JDBC-properties-value</em>"
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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Supports</td>
<td>Hive, parallel data transfer, serial data transfer</td>
</tr>
</tbody>
</table>
| Example           | properties=
  "hive.exec.dynamic.partition.mode=nonstrict;
hive.exec.dynamic.partition=true" |

**READBUFF**="<em>number-of-rows</em>"
specifies the number of rows to fetch per block of data retrieved.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: LOAD statement</td>
<td></td>
</tr>
<tr>
<td>Aliases</td>
<td>RAS=</td>
</tr>
<tr>
<td>Default</td>
<td>Calculated automatically based on row size</td>
</tr>
</tbody>
</table>
Restriction This option is not valid with parallel data transfer (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, DELETESOURCE, LIST, LOAD, and SAVE statements

Default 'default'

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

**SERVER="server-identifier"**

specifies the name of the node in the Hadoop cluster where the Hive service is running.

Valid in CASLIB statement

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

**SLICECOLUMN="column-name"**

specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each
partition is assigned to a node that performs the Read operation for its slice of data.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Interaction</td>
<td>Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.</td>
</tr>
<tr>
<td>Supports</td>
<td>serial data transfer</td>
</tr>
</tbody>
</table>

**SRCTYPE=hadoop**
specifies that the data source is Hadoop (Hive or HDMD data source).

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [required]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Supports</td>
<td>HDMD, Hive, parallel data transfer, serial data transfer</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>0 (no message)</td>
</tr>
<tr>
<td>Restriction</td>
<td>This option applies only to parallel data transfer (DATATRANSFERMODE=&quot;PARALLEL&quot;).</td>
</tr>
<tr>
<td>Supports</td>
<td>HDMD, Hive, parallel data transfer</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Interaction</td>
<td>If you use this option, options that alter the JDBC URI (such as PROPERTIES=) are ignored.</td>
</tr>
<tr>
<td>Supports</td>
<td>Hive, parallel data transfer, serial data transfer</td>
</tr>
</tbody>
</table>

**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.

**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.

**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 160.
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 (Hadoop) 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.
When an HDMD table is loaded using the SAS Data Connect Accelerator for Hadoop, here is how the raw data length of CHAR and VARCHAR columns in the resulting CAS table is determined.

**Table 7.5 Raw Data Length of CHAR and VARCHAR Columns**

<table>
<thead>
<tr>
<th>If the byte length is...</th>
<th>...and the character encoding ID of the column is...</th>
<th>...and the HDMD encoding is...</th>
<th>...the raw byte length equals:</th>
</tr>
</thead>
<tbody>
<tr>
<td>specified</td>
<td>not specified</td>
<td>specified</td>
<td>the byte length</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td>X</td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td>the character length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>multiplied by either</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARMULPLIER= or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NCHARMULTIPLIER=</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>based on the value of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>character encoding ID</td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td>X</td>
<td></td>
<td>the character length</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>multiplied by either</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>CHARMULPLIER= or</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>NCHARMULTIPLIER=</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>based on the value of the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>HDMD ENCODING = option</td>
</tr>
</tbody>
</table>

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 options that are specific to HDMD that are on a Hive caslib.

Table 7.6  Options for Specifying a Hive or HDMD Caslib for Hadoop

<table>
<thead>
<tr>
<th>HDFSMETADIR= Option</th>
<th>SERVER= Option</th>
<th>URI= Option</th>
<th>Caslib Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>specified</td>
<td>specified</td>
<td>specified</td>
<td>HDMD</td>
</tr>
<tr>
<td>not specified</td>
<td>specified</td>
<td></td>
<td>not specified</td>
</tr>
<tr>
<td>not specified</td>
<td>specified</td>
<td></td>
<td>not specified</td>
</tr>
</tbody>
</table>

| not specified       | specified      | specified   | Hive, using URI= |
| not specified       | specified      |             | not specified   |
| not specified       | specified      |             | Hive, using the configuration that is found in HadoopConfigPath |
| not specified       | 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 CAS. This table 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 7.7  Supported Hive and HDMD Data Types for Hadoop

<table>
<thead>
<tr>
<th>Hive Data Type</th>
<th>HDMD Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character Data Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>VARCHAR¹</td>
<td>VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>STRING</td>
<td>VARCHART</td>
<td>VARCHART</td>
</tr>
</tbody>
</table>

Numeric Data Types

<p>| ARRAY           | ERROR (not supported) |</p>
<table>
<thead>
<tr>
<th>Hive Data Type</th>
<th>HDMD Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td>INT64</td>
</tr>
<tr>
<td>BINARY</td>
<td>BINARY</td>
<td>ERROR (supported as VARCHAR for serial data transfer, not supported for parallel data transfer)</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>ERROR (supported as INT32 for serial data transfer, not supported for parallel data transfer)</td>
<td></td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DECIMAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INT</td>
<td>INT32</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>INT32</td>
<td></td>
</tr>
<tr>
<td>INTERVAL</td>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td>MAP</td>
<td>ERROR (not supported)</td>
<td></td>
</tr>
<tr>
<td>REAL</td>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td>INT32</td>
</tr>
<tr>
<td>STRUC</td>
<td>ERROR (not supported)</td>
<td></td>
</tr>
<tr>
<td>TINYINT</td>
<td>TINYINT</td>
<td>INT32</td>
</tr>
<tr>
<td>UNIONTYPE</td>
<td>ERROR (not supported)</td>
<td></td>
</tr>
</tbody>
</table>

**Date and Time Data Types**

<table>
<thead>
<tr>
<th>DATE</th>
<th>DATE</th>
<th>DOUBLE</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME</td>
<td>TIME</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>TIMESTAMP (DATETIMES)²</td>
<td></td>
<td>DOUBLE</td>
</tr>
</tbody>
</table>

1 Any VARCHAR data types that are greater than 65535 are automatically promoted to a string type upon CREATE.
2 SAS Data Connect Accelerator for Hadoop supports this data type up to the Year 4000 with millisecond precision.
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 30.

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.

```casc
  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 hdmdlib.

```casc
  caslib hdmdlib desc='hdmd caslib'
  dataSource=(srctype='hadoop',
              dataTransferMode='serial',
```
Example 3: Load a Hive Table into SAS Cloud Analytic Services Using PROC CASUTIL

```sas
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 hvcaslib 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

```sas
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.

---

**Impala Data Connector**

Enables you to load data from Impala into CAS.
Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Note: All users with SAS/ACCESS Interface to Impala can use SAS Data Connector to Impala.

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

Data connector options are used in the context of different statements that connect your data in Impala with CAS. 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 160.

`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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [see Requirement]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Alias</th>
<th>AUTHDOMAIN=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>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.</td>
</tr>
</tbody>
</table>
### BULKLOAD=TRUE | FALSE
specifies whether to use bulk loading to insert a large number of rows from CAS into an external table.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: SAVE statement</td>
</tr>
<tr>
<td>Default</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, LOAD, and LIST statements</td>
</tr>
<tr>
<td>Default</td>
<td>Active caslib</td>
</tr>
</tbody>
</table>

### CHARMULTIPLIER=value
specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD and SAVE statements</td>
</tr>
<tr>
<td>Default</td>
<td>1.0</td>
</tr>
<tr>
<td>Range</td>
<td>1.0–5.0</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>UTF-8</td>
</tr>
</tbody>
</table>

### 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

**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.
NCHARMULTIPLIER=value
specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

NUMREADBUFFERS=number
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

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 lets a data connector 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.

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 lets a data connector 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.
**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]**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASS=</td>
<td>none</td>
</tr>
<tr>
<td>PWD=</td>
<td></td>
</tr>
</tbody>
</table>

**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**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Default</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS=</td>
<td>Calculated automatically based on row size</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
<td></td>
</tr>
</tbody>
</table>

**Default** none

**SERVER=“server-identifier”**
specifies the server identifier for the DBMS server.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [required]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

**SLICECOLUMN=“column-name”**
specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

**SRCTYPE=“impala”**
specifies that the data source is an Impala database.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [required]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**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

<table>
<thead>
<tr>
<th>CASLIB statement [see Requirement]</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>

Alias

| 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 160.

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

The Impala data connector supports these data types for loading data from Impala into CAS. This table 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>
<thead>
<tr>
<th>Impala Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>STRING</td>
<td>VARCHAR</td>
</tr>
<tr>
<td><strong>Numeric Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>INT64</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DECIMAL( p, s )</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INT32</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>INT32</td>
</tr>
<tr>
<td>TINYINT</td>
<td>INT32</td>
</tr>
<tr>
<td><strong>Date and Time Data Types</strong></td>
<td></td>
</tr>
</tbody>
</table>
### Impala Data Type

<table>
<thead>
<tr>
<th>Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIMESTAMP(p)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(without time zone)</td>
<td>(formatted as DATETIMEw.d)</td>
</tr>
</tbody>
</table>

### 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 30.

### 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**

```r
  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**

```r
  proc casutil;
  list files incaslib="Impalacaslib";  1
  load casdata="myIMdata" incaslib="Impalacaslib" outcaslib="casuser" casout="IMdata_from_Impalacaslib";
  list tables incaslib="casuser";       3
```
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.

JDBC Data Connector

Enables you to transfer data between a JDBC data source and CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see parameters for details)

Notes: All users with SAS/ACCESS Interface to JDBC can use SAS Data Connector to JDBC. Support for this data connector was added in SAS Viya 3.4.

See: Chapter 8, "Data Types," on page 371

Example: Add a caslib that connects to a JDBC data source.

caslib jdcaslib sessref=mysess
dataSource=(srctype='jdbc',
   url="jdbc:mysql://<connection-options>"
   username='myuser1'
   password='myPwd'
   class="JDBC-driver-class"
   classpath="path-to-JDBC-driver-class");

Syntax

Data Connector Options for JDBC

Data connector options are used in the context of different statements that connect your data in JDBC with CAS. 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 160.

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

Default Active caslib

CHARMULTIPLIER=value
specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

Valid in CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default 1.0

Range 1.0–5.0

CLASS="class-name"
specifies the JDBC driver class name.

Valid in CASLIB statement [required]

Alias DRIVERCLASS=

Default none

CLASSPATH="JAR-file-path"
specifies the path to the JAR files that are used by the JDBC interface. Multiple locations should be separated with a colon (:) for Windows or a semicolon (;) for UNIX.

Valid in CASLIB statement [required]

Default none

Interaction This value overrides the value of the SAS_ACCESS_JAR_PATH environment variable, if it is set.
**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 JDBC 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

**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

**DBMAXTEXT=maximum-string-column-length**
specifies the maximum length for all string data types. For example, if dbMaxText=100, all string data types are loaded as VARCHAR(100).

Valid in  CASLIB statement

Default  Calculated automatically based on row size

**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

Default  Calculated automatically based on row size

**LOGIN_TIMEOUT=**
specifies the amount of time in seconds to wait before ending a login attempt.

Valid in  CASLIB statement

**NCHARMULTIPLIER=value**
specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

Valid in  CASLIB statement

Default  3
NUMREADBUFFERS=number
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in  
CASLIB statement
PROC CASUTIL: LOAD statement

Default  4
Range  1-10

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, DELETESOURCE, 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.

QUERY_TIMEOUT
specifies the amount of time in seconds to wait for a query response.

Valid in  
CASLIB statement

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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>

| Default | none |

**SRCTYPE=**"jdbc"

specifies that the data source is accessed through JDBC.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [required]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

**URL=**"jdbc:driver-name://driver-connection-options"

specifies the JDBC connection string.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [required]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: SAVE statement</td>
</tr>
</tbody>
</table>

| 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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [see Requirement]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>

**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 \texttt{USERNAME=} and \texttt{PASSWORD=} values or by specifying an \texttt{AUTHENTICATIONDOMAIN=} value.

Details

Use Options

Data connector options are used in the context of different statements that connect your data in JDBC 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 160.

JDBC Naming

The data connector can load JDBC tables with names up to 128 characters or with column names that are up to 128 characters.

Authentication to a JDBC Database

Credentials are required to access the data in a JDBC 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 \texttt{CASLIB} statement and specify a value for either \texttt{AUTHENTICATIONDOMAIN=} or both \texttt{USERNAME=} and \texttt{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 \texttt{USERNAME=} and \texttt{PASSWORD=} to override the original credentials. You can also override authentication domain credentials with a different \texttt{AUTHENTICATIONDOMAIN=} value.

Although it is not typical, it is possible to supply \texttt{USERNAME=} and \texttt{PASSWORD=} in separate statements. For example, you might supply a \texttt{USERNAME=} value in the \texttt{CASLIB} statement and then supply a \texttt{PASSWORD=} value when you specify a \texttt{LOAD} statement that accesses the data. In this situation, you must supply the \texttt{PASSWORD=} value for each statement that accesses the data.

Load a Connection to a JDBC 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 \texttt{CASLIB} statement.

Supported JDBC Data Types

This table shows the data types that can be loaded from JDBC 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 7.9  Supported JDBC Data Types

<table>
<thead>
<tr>
<th>JDBC Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>STRING</td>
<td>VARCHAR</td>
</tr>
<tr>
<td><strong>Numeric Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>INT64</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DECIMAL(p, s)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INT32</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>INT32</td>
</tr>
<tr>
<td>TINYINT</td>
<td>INT32</td>
</tr>
<tr>
<td><strong>Date and Time Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP(p) (without time zone)</td>
<td>DOUBLE (formatted as DATETIME w.d)</td>
</tr>
</tbody>
</table>

**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 30.

**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 a JDBC Connection to PostgreSQL via JDBC and Load a PostgreSQL Table

```plaintext
caslib jdcaslib desc='JDBC Caslib'
   dataSource=(srctype='jdbc',
               url="jdbc:postgresql://<PostgreSQL-server>:<PostgreSQL-port>/
                   <PostgreSQL-database>",
               username='myuser1',
               password='myPwd'
               class="org.postgresql.Driver"
               classpath="/u/jdbc/postgres/39.2.2:/opt/machine/class");

proc casutil;
   list files incaslib="jdcaslib"; 1
   load casdata="myDBdata" incaslib="jdcaslib" outcaslib="casuser"
      casout="DBdata_from_jdcaslib";
   list tables incaslib="casuser"; 3
   contents casdata="DBdata_from_jdcaslib" incaslib="casuser";
   quit;
```

1 Create the caslib Jdcaslib that connects to a PostgresQL database via JDBC.
2 List the tables in Jdcaslib before loading your data.
3 Load the table myDBdata from your JDBC data source into caslib Casuser. Call the new table DBdata_from_jdcaslib.
4 List the tables in caslib Casuser to see the newly created DBdata_from_jdcaslib table that you loaded.
5 List information about the newly loaded table, including column names, data types, and so on.

Microsoft SQL Server Data Connector

Enables you to transfer data between Microsoft SQL Server and CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Notes: All users with SAS/ACCESS Interface to Microsoft SQL Server can use SAS Data Connector to Microsoft SQL Server.

Support for this data connector was added in SAS Viya 3.3.
Examples:

Establish a connection between your Microsoft SQL Server database and SAS Cloud Analytic Services.

```plaintext
caslib sqlservercaslib desc='Microsoft SQL Server Caslib'
   srcType='sqlserver'
   username='user1'
   password='myPwd'
   sqlserver_dsn="configured-DSN-name"
   catalog='*');
```

Overriding the user and password values.

```plaintext
proc casutil;
   load casdata="mycas.sqlsvreamp" casout="mySQLsvrdata" incaslib="sqlservercaslib"
   outcaslib="casuser"
   dataSourceOptions={
      username='user5'
      password='myPwd'});
quit;
```

Syntax

Data Connector Options for Microsoft SQL Server

Data connector options are used in the context of different statements that connect your data in Microsoft SQL Server with CAS. 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 160.

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [see Requirement]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>

**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 Microsoft 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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [Required]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Active caslib</td>
</tr>
</tbody>
</table>

**CHARMULTIPLIER=value**
specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1.0</td>
</tr>
<tr>
<td>Range</td>
<td>1.0–5.0</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Shell encoding</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Example</td>
<td>conopts=&quot;QueryTimeout=0&quot;</td>
</tr>
</tbody>
</table>
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

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

NCHARMULTIPLIER=value
specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

Valid in  CASLIB statement
   PROC CASUTIL: LOAD and SAVE statements
Default  3

NUMREADBUFFERS=number
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in  CASLIB statement
   PROC CASUTIL: LOAD statement
Default  4
Range  1-10

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 lets a data connector 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  CASLIB statement
   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 lets a data connector 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  CASLIB statement

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, DELETESOURCE, 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.
SLICECOLUMN="column-name"

specifies the column to use when partitioning data for a multinode data Read. Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

VALID IN  CASLIB statement

Default  none

Interaction  Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

SQLSERVER_DSN="Microsoft-SQL-Server-datasource-name"

specifies the data source name.

With this option, you can use a Microsoft SQL Server data source that you have previously configured in an odbc.ini file.

VALID IN  CASLIB statement [required]

Aliases  DB=

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.

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, DELETESOURCE, 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>
<thead>
<tr>
<th>Microsoft SQL Server Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT(n)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>CHARACTER(n)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>NCHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>Microsoft SQL Server Data Type</td>
<td>CAS Data Type</td>
</tr>
<tr>
<td>--------------------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>TEXT</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>BIGINT</td>
<td>INT64</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INT32</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>INT32</td>
</tr>
<tr>
<td>TINYINT</td>
<td>INT32</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DECIMAL(p, s)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>MONEY</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SMALLEY</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DATE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(formatted at DATEw.)</td>
<td></td>
</tr>
<tr>
<td>TIME(p)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(without time zone)</td>
<td></td>
</tr>
<tr>
<td>DATETIME(p)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(without time zone)</td>
<td></td>
</tr>
<tr>
<td>DATETIME2(p)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(formatted as DATETIMEw.)</td>
<td></td>
</tr>
</tbody>
</table>

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 30.
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 options in this example are required in the CASLIB statement except for PASSWORD=.

In this example, the Microsoft SQL Server data is stored at the location designated by the SQLSERVER_DSN= option.

```sql
caslib SQLservercaslib desc='Microsoft SQL Server Caslib'
   dataSource=(srctype='sqlserver',
               username='myuser1',
               password='myPwd',
               sqlserver_dsn='dbsqlserver');
```

Example 2: Load Microsoft SQL Server Data into SAS Cloud Analytic Services Using PROC CASUTIL

```sql
proc casutil;
   list files incaslib="SQLservercaslib";  \ 1
   load casdata="mySQLsvrdata" incaslib="SQLservercaslib" casout="casuser"
      casout="class_from_SQLservercaslib"; \ 2
   list files incaslib="casuser";
   contents casdata="class_from_SQLservercaslib" incaslib="casuser";
   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.
MongoDB Data Connector

Enables you to transfer data between MongoDB and CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Notes: All users with SAS/ACCESS Interface to MongoDB can use SAS Data Connector to MongoDB.

Support for the MongoDB data connector was added in SAS Viya 3.5.

Examples: Establish a connection between your MongoDB database and SAS Cloud Analytic Services.

caslib mycaslib desc='MongoDB Caslib'
    dataSource=(srctype='mongodb'
        server='mymachine.com'
        username='user1'
        password='myPwd'
        db="database-name"
        schema_collection="mySchema")

Overriding the user and password values.

proc casutil;
   load casdata="examp" casout="myDBdata" incaslib='mycaslib'
       outcaslib='casuser'
       dataSourceOptions=(
           username='user5'
           password='myPwd5');
quit;

Syntax

Data Connector Options for MongoDB

Data connector options are used in the context of different statements that connect your data in MongoDB with CAS. 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 160.

For more information about schema derivation and MongoDB data specifics, see “SAS/ACCESS Interface to MongoDB” in SAS/ACCESS for Nonrelational Databases: Reference.

Here are the data connector options for MongoDB:

**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

Default  
Active caslib

**CHARMULTIPLIER=value**

specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

Valid in  
CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default  
1.0

Range  
1.0–5.0

**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**

**Alias** DB=

**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

**MAX_CHAR_LEN=value**
specifies the maximum column width for results that are returned from an external database.

**Valid in** CASLIB statement

**PROC CASUTIL: LOAD statement**

**Default** 32767

**Range** 1–32767

**NUMREADBUFFERS=number**
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

**Valid in** CASLIB statement

**PROC CASUTIL: LOAD statement**

**Default** 4

**Range** 1-10

**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 lets a data connector 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** CASLIB statement

**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 lets a data connector 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  CASLIB statement
          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, DELETESOURCE, 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 value to use for the MongoDB data connector.

Valid in  CASLIB statement

Default  27017

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, DELETESOURCE, LIST, LOAD, and SAVE statements |

Default none

SCHEMA COLLECTION=’collection-name’ specifies that a connection to MongoDB should reference the stored schema file for the specified collection.

If the specified collection does not already have a stored schema file, then a scan is performed and the schema file is created.

To generate a temporary schema that is held only in memory, specify SCHEMA_COLLECTION=’$temp$’. For more information, see “Creating a Temporary Schema” on page 245.

Valid in CASLIB statement

Default sas-default-schema

Requirements You must have Write permission for the location where your schema is stored if you specify a new schema name.

- Journaling must be enabled for MongoDB before you can save a JSON file with a schema description.

- Schema names must be unique across MongoDB collections.

Restriction There can be only one schema per collection. However, there can be multiple collections (and therefore multiple schemas) in a database.

Tip Creating a schema in this way creates a MongoDB collection with the same name as the schema alongside your operational data collections. Consider using the SCHEMA_DB= option to keep the schema collection separate from your operational data.

See “Working with Schemas for MongoDB Data” in SAS/ACCESS for Nonrelational Databases: Reference

SCHEMA_DB=’database-name’ specifies the MongoDB database where a schema is stored.

Valid in CASLIB statement

Alias SCHEMA_DATABASE=

Default DATABASE= value
The user name specified in the CASLIB statement must have Write access to the database that is specified in the SCHEMA_DB= option (unless you use the SCHEMA_USER= and the SCHEMA_PASSWORD= options).

**SCHEMA_PASSWORD=**`string`

specifies the password to use to access a schema.

- **Alias**: SCHEMA_PWD=
- **Valid in**: CASLIB statement
- **Default**: PASSWORD= value

**SCHEMA_PORT=**`port-value`

specifies the port to use to access a schema.

- **Valid in**: CASLIB statement
- **Default**: PORT= value

**SCHEMA_SERVER=**`<server-name>`

specifies the server name or IP address of the server that stores the schema mapping.

- **Valid in**: CASLIB statement
- **Default**: SERVER= value
- **Requirement**: Use quotation marks if `server-name` contains spaces or nonalphanumeric characters or if the name is an IP address.

**SCHEMA_SSL_CA=**"`path-and-filename`"

specifies the full path name to the certificate authority file for the MongoDB schema location.

- **Valid in**: CASLIB statement
- **Default**: SSL_CA= value
- **Interaction**: If you use SCHEMA_SSL_CA= to connect to the MongoDB schema file, you must also use the SSL_CA= option to connect to the MongoDB database.

**SCHEMA_SSL_CERT=**"`path-and-filename`"

specifies the full path to the certificate file for the MongoDB schema location.

- **Valid in**: CASLIB statement
- **Default**: SSL_CERT= value
- **Interaction**: If you use SCHEMA_SSL_CERT= to connect to the MongoDB schema file, you must also use the SSL_CERT= option to connect to the MongoDB server.

**SCHEMA_UID=**`user-ID`

specifies the user ID to use to access a schema.
Valid in CASLIB statement

Default USER= value

**SERVER=“server-identifier”**
specifies the server identifier for the DBMS server.

Valid in CASLIB statement [required]

Default none

**SLICECOLUMN=“column-name”**
specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

Valid in CASLIB statement

Default none

Interaction Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

**SRCTYPE=“mongodb”**
specifies that the data source a MongoDB database.

Valid in CASLIB statement [required]

Default none

**SSL_CA=“path-and-file-name”**
specifies the full path to the certificate authority file to use to access MongoDB.

Valid in CASLIB statement

Default none

Interaction If you use the SSL_CA= option to connect to your MongoDB database, then you must also specify the SCHEMA_SSL_CA= option to connect to a schema file.

**SSL_CERT=“path-and-file-name”**
specifies the full path to the certificate file to use to access MongoDB.

Valid in CASLIB statement

Default none

Interaction If you use the SSL_CERT= option to connect to your MongoDB database, then you must also specify the SCHEMA_SSL_CERT= option to connect to a schema file.
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, DELETESOURCE, 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
About the MongoDB Data Connector
The MongoDB data connector enables you to establish a connection between the MongoDB database and CAS. The MongoDB data connector enables you to load data into CAS and to create, modify, and delete tables in the MongoDB database.

Note:  Support for the MongoDB data connector was added in SAS Viya 3.5.

For additional information about working with a MongoDB database and working with SAS/ACCESS schemas for MongoDB, see “SAS/ACCESS Interface to MongoDB” in SAS/ACCESS for Nonrelational Databases: Reference.
Importance of Specifying the SCHEMA_COLLECTION= Data Connector Option

It is recommended that you always specify the SCHEMA_COLLECTION= option in your CASLIB statements for MongoDB. This requires that a connection refer to a JSON file that describes the schema for the specified collection. If a schema file for the given collection does not exist, then a scan is performed and the file is created. Specifying the SCHEMA_COLLECTION= option improves data connector performance, because a new scan is not required each time that you access MongoDB. Specifying SCHEMA_COLLECTION= also ensures that all users who connect using the same SCHEMA_COLLECTION= value are accessing the same stored schema.

If you do not specify the SCHEMA_COLLECTION= option, the default schema value, sas-default-schema, is assumed. If this default schema already exists, you might be accessing data that you do not intend to access. For example, another user might have created this schema since your last SAS session. If the default schema does not already exist, SAS/ACCESS scans the MongoDB data to generate a new stored schema for sas-default-schema. This is a resource intensive task.

Creating a Temporary Schema

If you want to work with a schema without storing the schema file, you can create a temporary schema. To do this, specify SCHEMA_COLLECTION='$temp$'. You might wish to do this if you do not have permission to Write to a MongoDB data source, or if you want to perform testing. However, be aware that although the schema file is not stored, any changes that you make to the source data, such as adding tables or deleting data are passed onto the MongoDB data.

Deleting MongoDB Data

You can delete specific rows from a table in the MongoDB schema using the DELETE statement for PROC FEDSQL. Specify table names as they are listed in the output from a LIST statement in PROC CASUTIL.

In this example, the first DELETE statement deletes all rows where Surname is 'Bloggs' from the Sample_name table. The next DELETE statement deletes the second entry in the array data in table Sample_creditcards.

```
caslib mycaslib desc='MongoDB Caslib'
dataSource=(srctype='mongodb'
  server='mymachine.com'
  username='user1'
  password='myPwd'
  db='database-name'
  schema_collection="mySchema");

proc fedsql;
  delete from mycaslib."sample_name" where "surname"='Bloggs';
  delete from mycaslib."sample_creditcards" where "_index"=1;
quit;
```

Supported MongoDB Data Types

The following table shows the data types that can be loaded from MongoDB 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 7.11  Supported MongoDB Data Types

<table>
<thead>
<tr>
<th>MongoDB Data Type</th>
<th>CAS Data Type</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data Types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JAVASCRIPT</td>
<td>CHAR</td>
<td>Default format is $w$.</td>
</tr>
<tr>
<td>REGEX</td>
<td>CHAR</td>
<td>Default format is $w$.</td>
</tr>
<tr>
<td>STRING</td>
<td>CHAR</td>
<td>Default format is $w$.</td>
</tr>
<tr>
<td><strong>Numeric Data Types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BOOL</td>
<td>NUMERIC</td>
<td>Default format is w. In SAS, the value 0 is False, and any other numeric value is interpreted as True.</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DOUBLE</td>
<td>Default format is w.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>Default format is w.</td>
</tr>
<tr>
<td>INT</td>
<td>NUMERIC</td>
<td>Default format is w.</td>
</tr>
<tr>
<td>LONG</td>
<td>NUMERIC</td>
<td>Default format is w.</td>
</tr>
<tr>
<td><strong>Other Data Types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>ARRAY</td>
<td>Varies</td>
<td>Converted to a child table in SAS with variables of the appropriate type.</td>
</tr>
<tr>
<td>BINDATA</td>
<td>CHAR</td>
<td>Contains a hexadecimal representation of the binary data. Default format is $w$.</td>
</tr>
<tr>
<td>DATE</td>
<td>DATETIME</td>
<td>Date values become SAS dates, formatted based on your specifications.</td>
</tr>
<tr>
<td>NULL</td>
<td>CHAR</td>
<td>MongoDB NULL values are read in as missing character values.</td>
</tr>
<tr>
<td><strong>MongoDB Data Type</strong></td>
<td><strong>CAS Data Type</strong></td>
<td><strong>Notes</strong></td>
</tr>
<tr>
<td>-----------------------</td>
<td>------------------</td>
<td>----------</td>
</tr>
<tr>
<td>OBJECT</td>
<td>Varies</td>
<td>JSON objects typically become child tables with variables of the appropriate data type. However, if a JSON object that is not accounted for in the current schema map is encountered, then that object is returned as a missing value. This might occur if a JSON object is added after the initial scan to determine the schema for a collection.</td>
</tr>
<tr>
<td>OBJECTID</td>
<td>CHAR</td>
<td>Default format is $w.</td>
</tr>
</tbody>
</table>

**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 30.

**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: Add a MongoDB Database as a Data Source and Load Data into SAS Cloud Analytic Services**

caslib mycaslib desc='MongoDB Caslib'
   dataSource=(srctype='mongodb',
                host='MongoDBserver',
                username='user1',
                password='myPw')
Establish a connection between your MongoDB source data and a caslib, mycaslib.

List the tables in Mycaslib before loading your data.

Load the table myDBdata from Mycaslib into caslib Casuser. Call the new table class_from_mycaslib.

List the tables in Casuser to see the newly created table, class_from_mycaslib, that you loaded.

List information about the newly loaded table, including column names, data types, and so on.

MySQL Data Connector

Enables you to transfer data between MySQL and CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Notes: All users with SAS/ACCESS Interface to MySQL can use SAS Data Connector to MySQL.
Support for the MySQL data connector was added in SAS Viya 3.4.

Examples: Establish a connection between your MySQL database and SAS Cloud Analytic Services.

caslib mycaslib desc='MySQL Caslib'
   dataSource=(
      srctype='mysql'
      host='mySQLserver'
      username='user1'
      password='myPwd'
      database="database-name"
      schema='myschema'\n    );

Overriding the user and password values.

proc casutil;
   load casdata="examp" casout="myDBdata" incaslib='mycaslib'
      casout='class_from_mycaslib';
   list files incaslib='casuser';
   contents casdata='class_from_mycaslib' incaslib='casuser';
quit;
Syntax

Data Connector Options for MySQL

Data connector options are used in the context of different statements that connect your data in MySQL with CAS. 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 160.

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [see Requirement]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>

**BASEDIR="path-to-installation-directory"**

specifies the path to the MySQL installation directory. This option sets the BASEDIR environment variable for MySQL.

**CHARMULTIPLIER=value**

specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD and SAVE statements</td>
</tr>
</tbody>
</table>

**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.

**HOST=\"host-name\"**

specifies the MySQL host name.

Valid in  | CASLIB statement [required]  
---|---  
Alias | SERVER=  
Default | none  

**MAX_BINARY_LEN=\"length\"**

specifies the maximum length of a binary variable when you create a table.

Valid in  | CASLIB statement  
---|---  
Default | none  

**MAX_CHAR_LEN=\"bytes\"**

specifies the number of characters allowed for CHAR values when you create a table.

Valid in  | CASLIB statement  
---|---  
Default | 2048  

**MAX_TEXT_LEN=\"number-of-characters\"**

specifies the maximum number of characters allowed in text data types when you create a table.

Valid in  | CASLIB statement  
---|---  
Default | 8190
NCHARMULTIPLIER=value
specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

Valid in  CASLIB statement
         PROC CASUTIL: LOAD and SAVE statements
Default 3

NUMREADBUFFERS=number
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in  CASLIB statement
         PROC CASUTIL: LOAD statement
Default 4
Range   1-10

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 lets a data connector 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  CASLIB statement
         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 lets a data connector 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  CASLIB statement
         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, DELETESOURCE, 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 MySQL port number.

Valid in CASLIB statement

Default none

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, DELETESOURCE, LIST, LOAD, and SAVE statements

Default none
If you specify values for SCHEMA= and DATABASE=, the value for SCHEMA= is used to qualify the table name.

**SLICECOLUMN="column-name"**
specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

Valid in CASLIB statement

Default none

Interaction Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

**SOCKET="file-name"**
specifies the path and file name of the socket file that listens for connections to MySQL. If you provide this value, the MySQL server creates the socket file in the data directory unless you specify an absolute path to a different directory. On Windows, this option specifies a pipe name to use when listening for connections that use a named pipe.

Valid in CASLIB statement

Default /tmp/mysql.sock [UNIX]

MySQL (not case sensitive) [Windows]

**SRCTYPE="mysql"**
specifies that the data source a MySQL 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, DELETESOURCE, 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 MySQL Data Types

The following table shows the data types that can be loaded from MySQL 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>
<thead>
<tr>
<th>MySQL Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data</strong></td>
<td></td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>TEXT</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>TINYTEXT</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>MEDIUMTEXT</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>LONGTEXT</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>JSON</td>
<td>VARCHAR</td>
</tr>
<tr>
<td><strong>Binary Data</strong></td>
<td></td>
</tr>
<tr>
<td>MySQL Data Type</td>
<td>CAS Data Type</td>
</tr>
<tr>
<td>------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>BLOB</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>TINYBLOB</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>MEDIUMBLOB</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>LONGBLOB</td>
<td>VARCHAR</td>
</tr>
</tbody>
</table>

**Numeric Data**

| ENUM             | VARCHAR      |
| SET              | VARCHAR      |
| INT              | INTEGER (INT32) |
| TINYINT          | INTEGER (INT32) |
| SMALLINT         | INTEGER (INT32) |
| BIGINT (INT64)   | BIGINT (INT64) |
| DECIMAL          | DOUBLE       |
| DOUBLE           | DOUBLE       |

**Date and Time Data**

| FLOAT            | DOUBLE       |
| DATE             | DOUBLE       |
| (formatted as DATEw.) |
| TIME             | DOUBLE       |
| (formatted as TIMEw.d) |
| TIMESTAMP        | DOUBLE       |
| (formatted as DATETIMEw.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 30.
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 MySQL Database as a Data Source for SAS Cloud Analytic Services

Use the CASLIB statement to establish a connection between your MySQL source data and a caslib, MySQLcaslib. All of the options supplied in this example are required in the CASLIB statement, except the PASSWORD= option.

In this example, the MySQL data is stored at the location designated by the SERVER= and DATABASE= options.

```plaintext
caslib mycaslib desc='MySQL Caslib'
dataSource=(srctype='mysql',
    host='MySQLserver',
    username='user1',
    password='myPwd',
    database='myDatabase',
    schema='mySchema');
```

Example 2: Load MySQL Data into SAS Cloud Analytic Services Using PROC CASUTIL

```plaintext
proc casutil;
    list files incaslib="mycaslib";  
    load casdata="myDBdata" incaslib="mycaslib" outcaslib="casuser" casout="class_from_mycaslib";  
    list files incaslib="casuser";  
    contents casdata="class_from_mycaslib" incaslib="casuser";
quit;
```

1 List the tables in Mycaslib before loading your data.
2 Load the table myDBdata from Mycaslib into caslib Casuser. Call the new table class_from_mycaslib.
3 List the tables in Casuser to see the newly created table, class_from_mycaslib, that you loaded.
List information about the newly loaded table, including column names, data types, and so on.

**ODBC Data Connector**

Enables you to transfer data between an ODBC data source and CAS.

Valid in: CASLIB statement, PROC CASUTIL: LOAD statement

Note: All users with SAS/ACCESS Interface to ODBC can use SAS Data Connector to ODBC.

Example: Establish a connection to ODBC data. (Note: The GLOBAL option is restricted to administrators.)

```plaintext
caslib odbccaslib desc="SQLviaODBCtoCaslib"
   datasource=(srctype="odbc"
       username="user1"
       password="password1"
       odbc_dsn="dbodbc") global;
```

**Syntax**

**Data Connector Options for ODBC**

Data connector options are used in the context of different statements that connect your data in ODBC with CAS. 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 160.

**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]

<table>
<thead>
<tr>
<th>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</th>
</tr>
</thead>
</table>

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="*" lets all catalogs be referenced.

Valid in: CASLIB statement  
PROC CASUTIL: CONTENTS, LOAD, and LIST statements  
Default: Active caslib

CHARMULTIPLIER=value

specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in: CASLIB statement  
PROC CASUTIL: LOAD and SAVE statements  
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
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

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

NCHARMULTIPLIER=value
specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

Valid in CASLIB statement
PROC CASUTIL: LOAD and SAVE statements
Default 3

NUMREADBUFFERS=number
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in CASLIB statement
PROC CASUTIL: LOAD statement
Default 4
Range 1-10

**ODBC_DSN=**"DSN-name"

specifies the data source name for your DBMS.

Valid in  CASLIB statement [Required]

Aliases DATABASE=

**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, DELETESOURCE, 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, DELETESOURCE, 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

**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, DELETESOURCE, 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 7.13  Data Type Conversions When Loading Data from ODBC Into CAS

<table>
<thead>
<tr>
<th>ODBC Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_BINARY</td>
<td>CHAR</td>
</tr>
<tr>
<td>SQL_CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>SQL_LONGVARBINARY</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_LONGVARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_VARBINARY</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_WCHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>SQL_WLONGVARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_WVARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_BIGINT</td>
<td>BIGINT (INT64)</td>
</tr>
<tr>
<td>SQL_BIT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_FLOAT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>ODBC Data Type</td>
<td>CAS Data Type</td>
</tr>
<tr>
<td>----------------------</td>
<td>--------------------------</td>
</tr>
<tr>
<td>SQL_INTEGER</td>
<td>INTEGER (INT32)</td>
</tr>
<tr>
<td>SQL_NUMERIC</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_SMALLINT</td>
<td>INTEGER (INT32)</td>
</tr>
<tr>
<td>SQL_TINYINT</td>
<td>INTEGER (INT32)</td>
</tr>
</tbody>
</table>

**Date and Time**

<table>
<thead>
<tr>
<th>SQL_TYPE_DATE</th>
<th>DOUBLE (formatted as DATE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_TYPE_TIME</td>
<td>DOUBLE (formatted as TIME)</td>
</tr>
<tr>
<td>SQL_TYPE_TIMESTAMP</td>
<td>DOUBLE (formatted as DATETIME)</td>
</tr>
</tbody>
</table>

**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 30.

**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 447.

```
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.

---

**Oracle Data Connector**

Enables you to load data from Oracle into CAS.

**Valid in:** CASLIB statement, PROC CASUTIL statements (see options for details)

**Note:** All users with SAS/ACCESS Interface to Oracle can use SAS Data Connector to Oracle.

**Examples:**

```
caslib orcaslib desc='Oracle Caslib'
  datasource=(srctype='oracle'
                username='myusr1'
                password='myPwd'
```
Override USER= and PASSWORD= values.

```plaintext
proc casutil;
  load casdata=%upcase(orexamp) incaslib="orcaslib"
  outcaslib="casuser"
  dataSourceOptions=(username='user5' password='pwd5');
```

### Syntax

#### Data Connector Options for Oracle

Data connector options are used in the context of different statements that connect your data in Oracle with CAS. 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 160.

**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 the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

- **Valid in**
  - CASLIB statement
  - PROC CASUTIL: LOAD and SAVE statements

- **Default**
  - 1.0
**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, LIST, LOAD statements</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>PROC CASUTIL: SAVE statement</th>
</tr>
</thead>
</table>

**Default**

none

---

**INSERTBUFF**="number-of-rows"

specifies the number of rows per block of data to save to a table in the data source.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: SAVE statement</td>
</tr>
</tbody>
</table>

**Default**

Calculated automatically based on row size

---

**NCHARMULTIPLIER**=value

specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD and SAVE statements</td>
</tr>
</tbody>
</table>

**Default**

3

---

**NUMREADBUFFERS**=number

specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD statement</td>
</tr>
</tbody>
</table>

**Default**

4
NUMREADNODES=\texttt{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 lets a data connector 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 \texttt{CASLIB statement}

\texttt{PROC CASUTIL: LOAD statement}

Default 1

NUMWRITENODES=\texttt{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 lets a data connector 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 \texttt{CASLIB statement}

\texttt{PROC CASUTIL: SAVE statement}

Default 1

ORA\_ENCODING="\texttt{<encoding-name>}"\texttt{"

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.

\textbf{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 \texttt{CASLIB statement}

\texttt{PROC CASUTIL: LOAD statement}

Default UNICODE

PASSWORD="\texttt{password}"

specifies the DBMS password for a user. You typically specify \texttt{USERNAME=} and \texttt{PASSWORD=} values (or an \texttt{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 \texttt{USERNAME=} and \texttt{PASSWORD=} value in a statement other than the \texttt{CASLIB} statement, these credentials override any credentials that were specified when the caslib was added.
Valid in CASLIB statement [see Requirement]

**PROC CASUTIL: CONTENTS, DELETESOURCE, 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 [required]

**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, DELETESOURCE, LIST, LOAD, and SAVE statements**
SLICECOLUMN="column-name"

specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

Valid in      CASLIB statement
Default       none
Interaction   Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

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, DELETESOURCE, LIST, LOAD, and SAVE statements
Aliases       UID=
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**

The Oracle data connector supports these data types for loading data from Oracle into CAS. This table 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>
<thead>
<tr>
<th>Oracle Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td><strong>Numeric Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>ARRAY</td>
<td>ERROR (not supported)</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>ERROR (not supported)</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td><strong>Additional Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>large object types (BLOB, CLOB)</td>
<td>ERROR (not supported)</td>
</tr>
<tr>
<td>MAP</td>
<td>ERROR (not supported)</td>
</tr>
<tr>
<td>STRUCT</td>
<td>ERROR (not supported)</td>
</tr>
<tr>
<td>UNION</td>
<td>ERROR (not supported)</td>
</tr>
</tbody>
</table>

**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 30.

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.

```plaintext
CASLIB oralib
    datasource=(srctype='oracle',
                username='myuser1',
                password='myPwd',
                path="myORAdata");
```

Example 2: Load Oracle Data into SAS Cloud Analytic Services Using PROC CASUTIL

```plaintext
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.

---

**ORC Data Connector**

Enables you to load data from Apache Optimized Row Columnar (ORC) tables into CAS.

**Valid in:** CASLIB statement

**Applies to:** CAS procedure, CASUTIL procedure

**Restrictions:**
- Only Linux for x64 is supported.
- Partitioned and ACID tables in Hive are not supported.
- Complex data types (LIST, MAP, STRUCT, UNION) are not supported.
- The root object type of an ORC table is a STRUCT. (A STRUCT is the only schema that is supported.) The root object type contains a field for each of the top-level columns.

**Requirements:**
- This data connector supports only ORC files. You must specify FILETYPE="ORC" or a file name that ends with .orc as the file type. Otherwise, the file is not found.
- When connecting to Azure Data Lake Storage Gen2, you must set the AZURETENANTID= server option or the AZURETENANTID= session option. You should also review and set the AZUREAUTHCACHELOC= server option if you want a different value from the default.

**Supports:**
- You can use this with a path or a Microsoft Azure Data Lake Storage Gen2 caslib.

**Notes:**
- To access data in your home directory instead of in a local or shared network storage location, use the CASUSER caslib.
- All users with SAS Viya can use SAS Data Connector to ORC.
- Support for the ORC data connector was added in SAS Viya 3.5.

**Tip:**
- To support ORC tables that Hive creates, the ORC data connector can access a directory of tables as a single logical table. The name of the directory must be specified as the table name. It must also be a subdirectory or subfolder under the defined caslib. To access a directory as a logical table, all tables in the directory must be ORC tables and they must also all have the same schema.

**Examples:**
- Add a PATH caslib to access ORC tables in a local or shared network storage location.

  ```
  caslib mycaslib desc='ORC Caslib'
  datasource=(srctype='path'
      path="/mytest/customer");
  ```
- Add an Azure Data Lake Storage caslib to access ORC tables in Azure Data Lake Storage Gen2.

  ```
  caslib mycaslib desc='ORC in ADLS Caslib'
  ```
Load an ORC table into SAS Cloud Analytic Services.

```sas
proc casutil;
    load casdata="cars.orc"
        importoptions=(filetype="orc");
run;
```

See: For details about the related SAS engine, see SAS® Viya® 3.5 Engines for Open-Source File Formats: Reference.

### Syntax

#### Data Connector Options for ORC

Data connector options are used in the context of different statements that connect your data in ORC tables with CAS. 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 160.

You can use this data connector with both the path and Azure Data Lake Storage casibs. The syntax and options for each differ.

**COMPRESS=YES | NO | ZLIB**

specifies the type of compression to use when a new ORC table is created.

- **Valid in:** CAS action: EXPORTOPTIONS= option on the SAVE action
- **Default:** ZLIB (when no value is specified)
- **Example:** `compress="ZLIB"`

**FILETYPE="ORC"**

specifies the file type.

**SRCTYPE="ADLS" | "PATH"**

specifies that the data source is an ORC table.

### Details

#### Overview

Apache ORC is an open-source file format that was originally developed for use in Hadoop. To access ORC tables requires a DATASOURCE= option on the addCaslib action with SRCTYPE="ADLS" or SRCTYPE="PATH" as the specified value.

#### Supported ORC Data Types

As shown in the table below, the ORC data connector supports the data types for loading data from an ORC table into SAS Cloud Analytic Services. The table 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.
Complex data types (LIST, MAP, STRUCTURE, UNION) are not supported. Tables that have these complex types cannot be accessed through the ORC data connector.

Table 7.15   Supported ORC Data Types

<table>
<thead>
<tr>
<th>ORC Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR(^1)</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR(^2)</td>
</tr>
<tr>
<td>STRING</td>
<td>VARCHAR(^2)</td>
</tr>
<tr>
<td><strong>Numeric Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>BIGINT (64-bit)</td>
<td>INT64</td>
</tr>
<tr>
<td>BINARY</td>
<td>VARCHAR(^2)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>BOOLEAN (1-bit)</td>
<td>INT32</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>FLOAT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INT (32-bit)</td>
<td>INT32</td>
</tr>
<tr>
<td>SMALLINT (16-bit)</td>
<td>INT32</td>
</tr>
<tr>
<td>TINYINT (8-bit)</td>
<td>INT32</td>
</tr>
<tr>
<td><strong>Date and Time Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DOUBLE(^3)</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DOUBLE(^3)</td>
</tr>
<tr>
<td><strong>Other Data Types</strong></td>
<td></td>
</tr>
</tbody>
</table>
Values with a length of less than or equal to 255 are a CHAR data type in CAS. Values with a length of greater than 255 are a VARCHAR data type in CAS. Values with lengths that are shorter than the specified length value are padded with spaces.

The maximum allowed length in the data store is 65,535. If no length exists in the ORC file, no VARCHAR length in CAS is defined.

An ORC type of DATE and TIMESTAMP is stored as a DOUBLE in CAS with a format of DATE9. and DATETIME21.2, respectively.

These include LIST, MAP, STRUCT, and UNION.

Converting Data from ORC to CAS

Character-based columns (CHAR, VARCHAR, STRING) that contain date, time, or timestamp values are not automatically converted to their respective SAS format types. They remain as character-string values.

Examples:

Example 1: Access an ORC Data Set Using the PATH CASLIB and the CAS Procedure

cas mysess user=&SYSUSERID
proc cas;
   /* Define a PATH type CASLIB called "casuser" and */
   /* point it to the location of the ORC data set. */
   addcaslib /
      caslib="casuser"
      datasource={srctype="path"}
      path="/mytest1/wky/orcdata";
   /* Use the CAS Procedure to load the ORC data set. */
   loadtable /
      caslib="casuser"
      path="car_sales"
   /* If the file name does not end with .orc, */
   /* the filetype="orc" must be specified. */
   importoptions={filetype="orc"}
   casout={name="xyz" replace=TRUE};
run;
quit;

Example 2: Create a Microsoft Azure Data Lake Storage Caslib and Save an ORC Data Set with the CAS Procedure

proc cas;
   addcaslib /
      datasource={srctype="adls"
         accountName="myAccount"
         filesystem="mylib"}
PC Files Data Connector

Enables you to load data from files of various file types into CAS.

Valid in: CASLIB statement
Applies to: CAS procedure
Note: All users with SAS/ACCESS Interface to PC Files can use SAS Data Connector to PC Files.

Syntax

Data Connector Options for PC Files

Data connector options are used in the context of different statements that connect your data in PC Files with CAS. 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 160.

```
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

```
FILETYPE="DTA" | "EXCEL" | "JMP" | "SPSS" | "XLS"
```

specifies the file type. Specify FILETYPE="EXCEL" to work with XLSX files (.xlsx file name suffix).

Valid in CASLIB statement
Default XLS
<table>
<thead>
<tr>
<th><strong>Restriction</strong></th>
<th>Use XLSX for national language characters. These characters are not displayed correctly for XLS files.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Requirement</strong></td>
<td>You must specify this option for any non-XLS file type.</td>
</tr>
</tbody>
</table>

**GETNAMES=TRUE | FALSE**

Uses the values in the first line of the file as variable names when GETNAMES=TRUE.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

**GETSHEETNAMES=TRUE | FALSE**

Specifies whether to create a CAS table with a list of all sheet names in an Excel workbook.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**Interaction**

When set to TRUE, this option overrides the SHEET= option.

**PATH="table-path-and-name"**

Specifies the location and the name of a PC Files source table.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

**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. The range can be absolute (for XLS and XLSX files) or named (for XSLX). For details, refer to *SAS/ACCESS® Interface to PC Files: Reference for Viya*.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none (imports the entire worksheet)</td>
</tr>
</tbody>
</table>

**Requirement**

To use this option, you must specify a range that represents a rectangle in the worksheet.

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none (imports the entire worksheet)</td>
</tr>
</tbody>
</table>

**Interaction**

Setting GETSHEETNAMES=TRUE overrides the value for any other option.
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 for loading data from PC Files into CAS. This table shows the resulting data type for the data after it has been loaded into SAS Cloud Analytic Services. This table 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>
<thead>
<tr>
<th>PC Files Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DTA(^1)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>JMP</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SPSS</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>XLS</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>XLSX</td>
<td>VARCHAR</td>
</tr>
</tbody>
</table>

\(^1\) When string data is being exported, this data type supports Unicode and more than 32,767 variables.
Examples:

Example 1: Use GETNAMES= in a Caslib to Import Table Column Names

This example uses testlib caslib to import table column names from an XLSX file with GETNAMES=.

```sas
proc cas;
  session mysess;
  parmlist={
    casLib="testlib",
    importOptions={fileType="xlsx", getnames="true",
                  sheet="test_sheet"} path="mytest.xlsx"
    casout={caslib="testlib",name="test_out"}
  };
  loadtable result=r / parmlist;
run;
```

Example 2: Load a Table into CAS

In this example, the Titanic table is loaded into SAS Cloud Analytic Services. Data was obtained from http://biostat.mc.vanderbilt.edu/DataSets.

```sas
options validvarname=any;
proc casutil;
  load file="/path/to/titanic3.xls" casout="titanic3"
    importOptions=(filetype="xls" getnames=true);
quit;
```

Example 3: 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.

```sas
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;
```
PostgreSQL Data Connector

Enables you to transfer data between PostgreSQL and CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Note: All users with SAS/ACCESS Interface to PostgreSQL can use SAS Data Connector to PostgreSQL.

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="pgexamp" casout="myPGdata" incaslib="postgrescaslib"
    outcaslib="casuser"
    dataSourceOptions=(
        username='user5'
        password='myPwd');
quit;

Syntax

Data Connector Options for PostgreSQL

Data connector options are used in the context of different statements that connect your data in PostgreSQL with CAS. 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 160.

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]
### 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.

### BLDELDATFILE=YES | NO
specifies whether to delete only the data file or all files that the data connector creates for the PSQL facility.

### Requirement
To specify this option, you must also specify the BULKUNLOAD= option.

### Default
YES

### Note
Support for this option is added in SAS Viya 3.5.

### BLDELIMITER='single-character'
specifies an override for the default delimiter character during bulk data retrieval (unloading).

### Default
, (comma)

### Requirement
To specify this option, you must also specify the BULKUNLOAD= option.

### Note
Support for this option is added in SAS Viya 3.5.

### BLESCAPE='single-character' | 'OFF'
specifies the single character to use for escape sequences.

### Default
\ (backslash character)

### Requirement
To specify this option, you must also specify the BULKUNLOAD= option.

### Note
Support for this option is added in SAS Viya 3.5.

### BLFORMAT=TEXT | CSV
specifies the format of the external table data: plain text (TEXT) or comma-separated values (CSV).

### Restriction
To specify this option, you must also specify the BULKUNLOAD= option.

### Default
TEXT

### Note
Support for this option is added in SAS Viya 3.5.

### BLNULL='\N' | empty-value
specifies the string that represents a null value.

### Default
'\N' for TEXT files or unquoted empty value for CSV files
BULKUNLOAD=TRUE | FALSE
specifies whether to retrieve (fetch) a large number of rows from an external table into CAS.

Interaction
Bulk unloading works only when BULKUNLOAD= is enabled and when NUMREADNODES= is not set to 1.

Valid in CASLIB statement

Default FALSE

Note Support for this option was added in SAS Viya 3.5.

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 the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.
This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in: CASLIB statement

<table>
<thead>
<tr>
<th>Default</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>1.0–5.0</td>
</tr>
</tbody>
</table>

**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: Shell encoding

**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

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]

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.
**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

**LDPATH=<LD-LIBRARY-path>**
specifies a full path to append to the existing LD_LIBRARY_PATH value for your system.

This option pertains only to bulk unloading (data retrieval).

**Default** none

**Restriction** To specify this option, you must also specify the BULKUNLOAD= option.

**Note** Support for this option is added in SAS Viya 3.5.

**NCHARMULTIPLIER=** value
specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

**Valid in** CASLIB statement

**PROC CASUTIL: LOAD and SAVE statements**

**Default** 3

**NUMREADBUFFERS=** value
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

**Valid in** CASLIB statement

**PROC CASUTIL: LOAD statement**

**Default** 4

**Range** 1-10

**NUMREADNODES=** value
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 lets a data connector 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** CASLIB statement

**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 lets a data connector 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** CASLIB statement

**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, DELETESOURCE, 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 ODBC SYSINI is not set, set the ODBCINI environment variable to the full path of the directory that contains the configured odbc.ini file.

**REDBUFF=\"number-of-rows\"**

specifies the number of rows to fetch per block of data retrieved.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD statement</td>
</tr>
<tr>
<td>Default</td>
<td>Calculated automatically based on row size</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

**SERVER=\"server-identifier\"**

specifies the server identifier for the DBMS server.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [required]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

**SLICECOLUMN=\"column-name\"**

specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>
Interaction  Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

**SRCTYPE=**"postgres"

specifies that the data source is a PostgreSQL database.

Valid in  CASLIB statement [required]

Default  none

**SSLMODE=**ALLOW | DISABLE | PREFER | REQUIRE | VERIFY-CA | VERIFY-FULL

specifies the support mode for a TLS encrypted connection to your data source.

Here are the possible values:

- **ALLOW**: Encryption is not required, but encryption is accepted if the DBMS server requires it.
- **DISABLE**: Do not use encryption when connecting to the database.
- **PREFER**: Encryption is not required, but encryption should be used if it is available.
- **REQUIRE**: Encryption is required when connecting to the database.
- **VERIFY-CA**: Encryption is required, and an authentication certificate might be required. Whether an authentication certificate is required depends on your database policy.
- **VERIFY-FULL**: Encryption and an authentication certificate are required.

Valid in  CASLIB statement

Default  PREFER

See  For more information, see Details in “SSLMODE= LIBNAME Statement Option” in SAS/ACCESS for Relational Databases: Reference.

**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

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, DELETESOURCE, 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>
<thead>
<tr>
<th>PostgreSQL Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIT(n)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>CHARACTER(n)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>TEXT</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>INTEGER (INT32)</td>
<td>INTEGER (INT32)</td>
</tr>
<tr>
<td>BIGINT (INT64)</td>
<td>BIGINT (INT64)</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>INTEGER (INT32)</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DECIMAL(p, s)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>PostgreSQL Data Type</td>
<td>CAS Data Type</td>
</tr>
<tr>
<td>----------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>NUMERIC</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SERIAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>BIGSERIAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DATE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td></td>
<td>(formatted as DATEw.)</td>
</tr>
<tr>
<td>TIME(p)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(without time zone)</td>
<td>(formatted as TIMEw.d)</td>
</tr>
<tr>
<td>TIMESTAMP(p)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(without time zone)</td>
<td>(formatted as DATETIMEw.d)</td>
</tr>
<tr>
<td>JSON, JSONB</td>
<td>VARCHAR</td>
</tr>
</tbody>
</table>

**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 30.

**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.

```cas
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

```sas
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.

Salesforce Data Connector

Enables you to transfer data between Salesforce and CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Notes: All users with SAS/ACCESS Interface to Salesforce can use SAS Data Connector to Salesforce.
Support for the Salesforce data connector was added in SAS Viya 3.5.

**Examples:**
Establish a connection between your Salesforce database and SAS Cloud Analytic Services.

```plaintext
caslib mycaslib desc='Salesforce Caslib'
   dataSource=(srctype='sforce'
                username='user1'
                password='myPwd');
```

Overriding the user and password values.

```plaintext
proc casutil;
   load casdata="sfexamp" casout="mySFdata" incaslib="mycaslib"
      outcaslib="casuser"
      dataSourceOptions=(
         username='user5'
         password='myPwd');
quit;
```

**Syntax**

**Data Connector Options for Salesforce**

Data connector options are used in the context of different statements that connect your data in Salesforce with CAS. 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 160.

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>AUTHDOMAIN=</td>
</tr>
<tr>
<td>Requirement</td>
<td>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.</td>
</tr>
</tbody>
</table>

**AUTHENDPOINT="authorization-endpoint"**

specifies the authorization endpoint to use.

| Valid in                  | CASLIB statement                                                   |
PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements

Default None

Example authendpoint="http://wsware.com/myendpoint"

CHARMULTIPLIER=value
specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

Valid in CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default 1.0

Range 1.0–5.0

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"

MUTUAL_AUTH=YES | NO
specifies whether to enforce mutual authentication.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements

Default NO

Requirement When MUTUAL_AUTH=YES, you must set the path to your local certificate (CERT_PATH=) and a password (CERT_PASS).

NCHARMULTIPLIER=value
specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.
Valid in CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default 3

**NUMREADBUFFERS=** *number*

specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Default 4

Range 1-10

**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, DELETESOURCE, LIST, LOAD, and SAVE statements

Aliases PIX=

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

**SHOW_METADATA=** **YES | NO**

specifies whether to display metadata tables and columns.

Valid in CASLIB statement

PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
<table>
<thead>
<tr>
<th>SHOW_RECYCLED=</th>
<th>NO</th>
<th>[specifies whether to display deleted table rows in the recycle bin.]</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid in</td>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alias</td>
<td>SHOW_DELETED=</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SRC_TYPE=sforce</th>
<th>[specifies that the data source is a Salesforce database.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid in</td>
<td>CASLIB statement [required]</td>
</tr>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE_NARROW_CHARACTER_TYPES=</th>
<th>TRUE</th>
<th>FALSE</th>
<th>[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.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid in</td>
<td>CASLIB statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>FALSE</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USE_NATIVE_NAMES=</th>
<th>YES</th>
<th>NO</th>
<th>[specifies whether to restore native naming conventions for custom tables and columns.]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Valid in</td>
<td>CASLIB statement</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Alias</td>
<td>NATIVE_NAMES=</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirement</td>
<td>When USE_NATIVE_NAMES=YES, the __s suffix is required on component fields to update them.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Restriction</td>
<td>The __c suffix must not be used in queries.</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>USERNAME=</th>
<th>user-name</th>
<th>[specifies the database or data source user name.]</th>
</tr>
</thead>
</table>
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, DELETESOURCE, 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 Salesforce Data Types**

The following table shows the data types that can be loaded from Salesforce 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 7.18  Supported Salesforce Data Types**

<table>
<thead>
<tr>
<th>Salesforce Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADDRESS</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>ANYTYPE</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>BASE64</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>COMBOBOX</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>DATACATEGORYGROUPREFERENCE</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>EMAIL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>ENCRYPTEDSTRING</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>ID</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>JUNCTIONIDLIST</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>Salesforce Data Type</td>
<td>CAS Data Type</td>
</tr>
<tr>
<td>---------------------</td>
<td>---------------</td>
</tr>
<tr>
<td>LOCATION</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>MASTERRECORD</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>MULTIPICKLIST</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>PHONE</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>PICKLIST</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>REFERENCE</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>STRING</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>TEXTAREA</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>URL</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>BYTE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>CURRENCY</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>PERCENT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>TIME</td>
<td>DOUBLE</td>
</tr>
<tr>
<td></td>
<td>(formatted as TIMEw.d)</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DOUBLE</td>
</tr>
<tr>
<td></td>
<td>(formatted as DATETIMEw.d)</td>
</tr>
</tbody>
</table>

**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 30.
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 Salesforce Database as a Data Source For SAS Cloud Analytic Services

Use the CASLIB statement to establish a connection between your Salesforce source data and a caslib, Salesforcecaslib. All options except the PASSWORD= option in this example are required in the CASLIB statement.

In this example, the Salesforce data is stored at the location that is designated by the SERVER= and DATABASE= options.

```
caslib mycaslib desc='Salesforce Caslib'
   dataSource=(srctype='sforce',
                username='user1',
                password='myPwd',
                database='myDatabase');
```

Example 2: Load Salesforce Data into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="mycaslib"; 1
   load casdata="myDBdata" incaslib="mycaslib" outcaslib="casuser" casout="class_from_mycaslib"; 2
   list files incaslib="casuser"; 3
   contents casdata="class_from_mycaslib" incaslib="casuser"; 4
   quit;
```

1 List the tables in Mycaslib before loading your data.
2 Load the table myDBdata from Mycaslib into caslib Casuser. Call the new table class_from_mycaslib.
3 List the tables in Casuser to see the newly created table, class_from_mycaslib, that you loaded.
4 List information about the newly loaded table, including column names, data types, and so on.
SAP HANA Data Connector

Enables you to load data from SAP HANA into CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Notes: All users with SAS/ACCESS Interface to SAP HANA can use SAS Data Connector to SAP HANA.

Support for this data connector was added in SAS Viya 3.3.

Examples: Load an SAP HANA data source and add a caslib to it.

```sas
  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.

```sas
  proc casutil
    incaslib='hacaslib'
sessref=mysess;
  list files;
  run;
  quit;
```

Syntax

Data Connector Options for SAP HANA

Data connector options are used in the context of different statements that connect your data in SAP HANA with CAS. 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 and action calls, see “Where to Specify Data Connector Options” on page 160.

**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
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 the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1.0</td>
</tr>
<tr>
<td>Range</td>
<td>1.0–5.0</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Example</td>
<td>conopts=&quot;QueryTimeout=0&quot;</td>
</tr>
</tbody>
</table>

**DBCREATETABLEOPTS="text"**

specifies any table options that should be appended to a CREATE TABLE statement that is passed to the data source.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>PROC CASUTIL: SAVE statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

**INSERTBUFF="number-of-rows"**

specifies the number of rows per block of data to save to a table in the data source.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Calculated automatically based on row size</td>
</tr>
</tbody>
</table>
### 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.

### NCHARMULTIPLIER=value

- **Specifies**: The bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.
- **Valid in**: CASLIB statement
  - PROC CASUTIL: LOAD and SAVE statements
- **Default**: 3
- **This option applies** to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

### NUMREADBUFFERS=number

- **Specifies**: The number of buffers that the data connector is to read and convert from the data source to CAS.
- **Valid in**: CASLIB statement
  - PROC CASUTIL: LOAD statement
- **Default**: 4
- **Range**: 1-10

### 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 lets a data connector 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**: CASLIB statement
  - 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 lets a data connector 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**: CASLIB statement
  - PROC CASUTIL: LOAD statement
- **Default**: 1
**Valid in CASLIB statement**

**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, DELETESOURCE, 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, DELETESOURCE, 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

**SLICECOLUMN="column-name"**
specifies the column to use when partitioning data for a multinode data Read. Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

Valid in | CASLIB statement
---|---

Default | none

Interaction | Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

**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)

**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.
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, DELETESOURCE, 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

The SAP HANA data connector supports these data types for loading data from SAP HANA into CAS. This table 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>
<thead>
<tr>
<th>SAP HANA Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character Data Types</td>
<td></td>
</tr>
<tr>
<td>ALPHANUM</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>NCHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>NVARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SHORTTEXT</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>TEXT</td>
<td>VARCHAR</td>
</tr>
</tbody>
</table>
### SAP HANA Data Type

<table>
<thead>
<tr>
<th>SAP HANA Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
</tr>
</tbody>
</table>

### Binary and Other Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>VARBINARY</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>BLOB</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>CLOB</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>NCLOB</td>
<td>VARCHAR</td>
</tr>
</tbody>
</table>

### Numeric Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>BIGINT (INT64)</td>
<td>INT64</td>
</tr>
<tr>
<td>DECIMAL(p, s) or DEC(p, s)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>FLOAT(n)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INT32</td>
</tr>
<tr>
<td>REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SMALLDECIMAL</td>
<td></td>
</tr>
<tr>
<td>SMALLINT (INT32)</td>
<td>INT32</td>
</tr>
<tr>
<td>TINYINT (INT32)</td>
<td></td>
</tr>
</tbody>
</table>

### Date and Time Data Types

<table>
<thead>
<tr>
<th>Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>DOUBLE (formatted as DATE)</td>
</tr>
<tr>
<td>SECONDDATE</td>
<td>DOUBLE (formatted as DATETIME)</td>
</tr>
<tr>
<td>TIME</td>
<td>DOUBLE (formatted as TIME)</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DOUBLE (formatted as DATETIME)</td>
</tr>
</tbody>
</table>

### 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 30.

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.

```sas
CASLIB HACASLIB SESSREF=MYSSESS DESC="hana caslib"
   DATA SOURCE=(SRCTYPE="hana",
                  SERVER="hanaServer",
                  USERNAME="myuser1",
                  PASSWORD="myPwd",
                  TABLETYPE="COLUMN",
                  SCHEMA="<Hana_schema_name>",
                  INSTANCE="00");
```

SAS Data Sets Data Connector

Enables you to load data from SAS data sets into CAS.

**Applies to:** CAS procedure, CASUTIL procedure

**Restrictions:** Although this data connector supports cross-environment data access (CEDA), it does not support DELETE operations with a CEDA data set. Therefore, if CEDA processing is invoked, you must use the DATASETS procedure to delete a SAS data set. Data sets with names that are greater than 32 bytes in UTF8 encoding cannot be accessed.

**Requirement:** Table names must be specified only in lowercase letters.

**Note:** All users with SAS Viya can use SAS Data Connector to SAS Data Sets.

**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.

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

Data connector options are used in the context of different statements that connect your data in SAS data sets with CAS. 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 160.

**ALTER="password"**

specifies the password to access and delete a password-protected data set.

<table>
<thead>
<tr>
<th>Default</th>
<th>none</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>This option is required only if the data set is password-protected.</td>
</tr>
</tbody>
</table>

**CHARMULTIPLIER=value**

specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

<table>
<thead>
<tr>
<th>Valid in</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>CASLIB statement</strong></td>
</tr>
<tr>
<td>PROC CASUTIL: LOAD and SAVE statements</td>
</tr>
<tr>
<td>Default</td>
</tr>
<tr>
<td>1.0</td>
</tr>
<tr>
<td>Range</td>
</tr>
<tr>
<td>1.0–5.0</td>
</tr>
</tbody>
</table>

**COMPRESS="YES" | "NO" | "BINARY" | "CHAR"**

specifies the compression method to use for the data set.

<table>
<thead>
<tr>
<th>Default</th>
<th>NO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interaction</td>
<td>Use this option with the REUSE= option.</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>importOptions option on the loadTable action</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th><strong>ENCODING=</strong>&quot;encoding&quot;</th>
<th>specifies the encoding for the data set.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Default</strong></td>
<td>AUTO</td>
</tr>
<tr>
<td><strong>Restriction</strong></td>
<td>Data is loaded serially if only the controller has access to the SAS data set.</td>
</tr>
<tr>
<td><strong>Requirement</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>Supports</strong></td>
<td>parallel data transfer, serial data transfer</td>
</tr>
</tbody>
</table>

**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.

**NUMREADBUFFERS=**`number`  
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in | CASLIB statement  
---|---
| PROC CASUTIL: LOAD statement
Default | 4
Range | 1-10

**PASSWORD=**`"password"`  
specifies the password for a password-protected SAS data set.

Valid in | PROC CASUTIL: CONTENTS, DELETESOURCE, 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 File Name Matching” on page 94.

Default | none
Requirement | To use the SAS Data Connector to SAS Data Sets, you must specify SRCTYPE=PATH in the DATASOURCE= option on the addCaslib action.
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.

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 reuse space.

*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).

*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= option on the addCaslib action with `srcType= "PATH"` as the specified value. Except for SAS data sets that are protected by metadata-bound library controls, you should be able to 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**

The SAS data sets data connector supports these data types for loading data from SAS data sets into SAS Cloud Analytic Services. This table 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 7.20  Supported SAS Data Sets Data Types

<table>
<thead>
<tr>
<th>SAS Data Sets Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHAR, VARCHAR</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
</tbody>
</table>

Example: Access a SAS Data Set Using the CASLIB Statement and the CAS Procedure

```sas
/* Find an engine-based SAS data set. */
libname abc '/mytest1/wky/test-v9cas/supio/testsio/lax';
proc contents data=abc.customer;run;

/* Create a CAS session. */
cas mysess user=&SYSUSERID;
proc cas;
/* Define a PATH type CASLIB called "casuser" and */
/* point it to where the SAS data set is located. */
addcaslib /
   caslib="casuser"
   datasource={srctype="path"}
   path="/mytest1/wky/test-v9cas/supio/tests001/linus";
run;

/* 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=mysess;
proc print data=mycas.customers;
run;
```

Snowflake Data Connector

Enables you to load data from Snowflake into CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Notes: All users with SAS/ACCESS Interface to Snowflake can use SAS Data Connector to Snowflake.
Support for this data connector was added in SAS Viya 3.4.

Examples: Establish a connection between your Snowflake database and SAS Cloud Analytic Services.
caslib snowcaslib desc='Snowflake Caslib'
    dataSources=
        srctype='snowflake'
        server='SFserver'
        schema='myschema'
        username='user1'
        password='myPwd'
        database="sfdatabase-name"
    );

Overriding the user and password values.

proc casutil;
    load casdata="mycas.pgexamp" casout="mySFdata" incaslib="snowcaslib" outcaslib="casuser"
        dataSourceOptions=(
            username='user5'
            password='myPwd'
        );
    quit;

Syntax

Data Connector Options for Snowflake

Data connector options are used in the context of different statements that connect your data in Snowflake with CAS. 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 160.

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.

AWSCONFIG="path-and-AWS-configuration-file-name"

specifies the location of the Amazon Web Services (AWS) configuration file.

Valid in

| CASLIB statement |
PROC CASUTIL: SAVE statement

Default  
~/.aws/config

Requirement  
To specify this option, you must also set BULKLOAD=TRUE.

Note  
Support for this option was added in SAS Viya 3.4.

BLOPTIONS="option-1<, option-2<, ..., option-N>"  
specifies options to pass to the bulk-load facility. These options affect how data is loaded and processed.  
Any options that you specify are appended to the COPY INTO command that is executed by the Snowflake data connector.

Valid in  
CASLIB statement

Alias  
BULKLOADOPTIONS=

Default  
none

Requirement  
To specify this option, you must also set BULKLOAD=TRUE.

Note  
Support for this option was added in SAS Viya 3.4.

Example  
BLOPTIONS="ROUNDEC"

BUCKET="Amazon-S3-bucket-name"  
specifies the name of the Amazon S3 bucket to use when bulk loading data. The bucket name that you specify must already exist.

Default  
none

Requirement  
To specify this option, you must also set BULKLOAD=TRUE.

Note  
Support for this option was added in SAS Viya 3.4.

BULKLOAD=TRUE | FALSE  
specifies whether to use bulk loading to insert a large number of rows from CAS into an external table.

Valid in  
CASLIB statement

Default  
none

CHARMULTIPLIER=value  
specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

Valid in  
CASLIB statement
PROC CASUTIL: LOAD and SAVE statements

<table>
<thead>
<tr>
<th>Default</th>
<th>1.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Range</td>
<td>1.0–5.0</td>
</tr>
</tbody>
</table>

**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.

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

**COMPRESS=TRUE | FALSE**
specifies whether to compress data using the GZIP format.

Valid in  CASLIB statement

<table>
<thead>
<tr>
<th>Default</th>
<th>FALSE</th>
</tr>
</thead>
</table>

**CREDSFILE=“path-and-filename”**
specifies the path and name of the AWS credentials file to use for bulk loading data.

Valid in  CASLIB statement

<table>
<thead>
<tr>
<th>Default</th>
<th>none</th>
</tr>
</thead>
</table>

**CREDSPROFILE=“profile-name”**
specifies the profile name to use when there is more than one profile in the AWS credentials file.

Valid in  CASLIB statement

<table>
<thead>
<tr>
<th>Default</th>
<th>none</th>
</tr>
</thead>
</table>

**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

**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

**DEFAULTDIR=**"directory-path"

specifies where all temporary files are created during bulk loading.

The value that you specify is prepended onto the file name. Provide the complete directory path, including the directory separator character.

Valid in  CASLIB statement  

PROC CASUTIL: SAVE statement

Alias  DEFAULTDIRECTORY=

Default  temporary-file-directory that is specified by the UTILLOC= system option

Requirement  To specify this option, you must also set BULKLOAD=TRUE.

Note  Support for this option was added in SAS Viya 3.4.

**DELETEDATAFILES=**TRUE | FALSE

specifies whether to delete only the data file or all files that are created during bulk loading. The files are deleted from the local machine and from the S3 bucket or the Snowflake stage area.

Valid in  CASLIB statement  

PROC CASUTIL: SAVE statement
### DELIMITER="character"

specifies an override of the default delimiter character that separates columns of data during bulk loading. You might specify a value for DELIMITER= if your character data contains the default delimiter character.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>bell character (ASCII 0x07)</td>
</tr>
<tr>
<td>Requirement</td>
<td>To specify this option, you must also set BULKLOAD=TRUE.</td>
</tr>
<tr>
<td>Note</td>
<td>Support for this option was added in SAS Viya 3.4.</td>
</tr>
</tbody>
</table>

### ESCAPE=TRUE | FALSE

specifies whether the Snowflake data connector scans character data for problematic character sequences that need to be escaped in the result data file.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>FALSE</td>
</tr>
<tr>
<td>Requirement</td>
<td>To specify this option, you must also set BULKLOAD=TRUE.</td>
</tr>
<tr>
<td>Note</td>
<td>Support for this option was added in SAS Viya 3.4.</td>
</tr>
</tbody>
</table>

### INSERTBUFF="number-of-rows"

specifies the number of rows per block of data to save to a table in the data source.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>Calculated automatically based on row size</td>
</tr>
</tbody>
</table>

### KEY="value"

specifies the Amazon Web Services access key that is used with key-based access control. If you are using temporary token credentials, this is the temporary access key ID.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Requirement</td>
<td>To specify this option, you must also set BULKLOAD=TRUE.</td>
</tr>
<tr>
<td>Note</td>
<td>Support for this option was added in SAS Viya 3.4.</td>
</tr>
</tbody>
</table>
**NCHARMULTIPLIER=value**
specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

Valid in | CASLIB statement
---|---
**PROC CASUTIL: LOAD and SAVE statements**
Default | 3

**NUMREADBUFFERS=number**
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in | CASLIB statement
---|---
**PROC CASUTIL: LOAD statement**
Default | 4
Range | 1-10

**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 lets a data connector 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 | CASLIB statement
---|---
**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 lets a data connector 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 | CASLIB statement
---|---
**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, DELETESOURCE, 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.

**PROFILE=**"profile-name"

specifies the profile to use when there is more than one profile in the AWS configuration file. When more than one profile is used, the profile name precedes the set of name-value pairs in square brackets. Here is an example configuration file with two profiles:

```
[default]
region=us-west-2
output=text

[analyst]
region=us-east-1
output=json
```

Valid in | CASLIB statement
---|---
---| PROC CASUTIL: SAVE statement

Default | default

Requirement | To specify this option, you must also set BULKLOAD=TRUE.

Note | Support for this option was added in SAS Viya 3.4.

**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

**SNOWFLAKE_DSN=**"Snowflake.datasource-name"

specifies the data source name.

With this option, you can use a Snowflake data source that you have previously configured in an odbc.ini file.

Valid in | CASLIB statement [required]
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.

**REGION=**"region"

specifies the AWS region from which S3 data is being loaded. Use the regions values that are available for the Amazon S3 service.

Valid in CASLIB statement

Valid in PROC CASUTIL: SAVE statement

Default none

Requirements To specify this option, you must also set BULKLOAD=TRUE.

You must specify the region when using bulk loading with the Snowflake data connector.

Note Support for this option was added in SAS Viya 3.4.

**S3CONFIG=**"local-path-and-configuration-file-name"

specifies the location of a local configuration file for Amazon S3. For more information about Amazon S3 configuration files, see the Amazon Web Services documentation.

Valid in CASLIB statement

Valid in PROC CASUTIL: SAVE statement

Default none

Requirements To specify this option, you must also set BULKLOAD=TRUE.

Note Support for this option was added in SAS Viya 3.4.

**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

Valid in PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
SECRET="secret-access-key"
specifies the secret access key or temporary secret access key that is used to
bulk load data.

An Amazon Web Services (AWS) secret access key is associated with the key
ID that you specify with the KEY= option.

Valid in  CASLIB statement
           PROC CASUTIL: SAVE statement

Default  none

Requirement To specify this option, you must also set BULKLOAD=TRUE.

Tip     To increase security, you can encode the key value by using PROC
         PWENCODE.

SERVER="server-identifier"
specifies the server identifier for the DBMS server.

Valid in  CASLIB statement [required]

Default  none

SLICECOLUMN="column-name"
specifies the column to use when partitioning data for a multinode data Read.
Ideally, the column that you select should contain integer data with a relatively
small number of distinct values that are fairly evenly distributed. Columns with
continuous numeric values can work as well. SAS uses the MOD function,
applied to the values in the specified column, to assign data to a partition. Each
partition is assigned to a node that performs the Read operation for its slice of
data.

Valid in  CASLIB statement

Default  none

Interaction Multinode data Read is not on by default. Use NUMREADNODES=
          to specify a number.

SRCTYPE="snowflake"
specifies that the data source is a Snowflake database.

Valid in  CASLIB statement [required]

Default  none

TOKEN="temporary-token"
specifies a temporary token that is associated with the temporary credentials that
you specify with the KEY= and SECRET= options.

Valid in  CASLIB statement
### 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

**Default**  
FALSE

### USESSL=TRUE | FALSE
specifies whether to use SSL encryption for connections to Amazon S3.

**Valid in**  
CASLIB statement

**Default**  
TRUE

### 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]

**Default**  
none

### PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements

**Aliases**  
UID=

**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
Details

Supported Snowflake Data Types

The following table shows the data types that can be loaded from Snowflake 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 7.21  Supported Snowflake Data Types

<table>
<thead>
<tr>
<th>Snowflake Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td><strong>Numeric Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>BIGINT</td>
<td>INT64</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DECIMAL(p, s)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INT32</td>
</tr>
<tr>
<td>NUMERIC(p, s)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td><strong>Date and Time Data Types</strong></td>
<td></td>
</tr>
<tr>
<td>DATE</td>
<td>DOUBLE (formatted as DATEw.)</td>
</tr>
<tr>
<td>TIME</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td>DOUBLE (formatted as TIMEw.) (formatted as DATETIMEw.d)</td>
</tr>
</tbody>
</table>
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 30.

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 Snowflake Database as a Data Source for SAS Cloud Analytic Services

Use the CASLIB statement to establish a connection between your Snowflake source data and a caslib, SFcaslib. All of the options supplied in this example are required in the CASLIB statement, except the PASSWORD= option.

In this example, the Snowflake data is stored at the location designated by the SERVER= and DATABASE= options.

```caslib SFcaslib desc='Snowflake Caslib'
dataSource=(srctype='snowflake',
server='SFserver',
username='user1',
password='myPwd',
database='sfdatabase');
```

Example 2: Load Snowflake Data into SAS Cloud Analytic Services Using PROC CASUTIL

```proc casutil;
list files incaslib="SFcaslib"; 1
load casdata="mySFdata" incaslib="SFcaslib" outcaslib="casuser"
casout="class_from_SFcaslib"; 2
list files incaslib="casuser";
contents casdata="%upcase(class_from_SFcaslib)" incaslib="casuser"; 3
```
List the tables in SFcaslib before loading your data.

Load the table mySFdata from SFcaslib into caslib Casuser. Call the new table class_from_SFcaslib.

Note: You must specify Snowflake table names using the capitalization that is used in the database.

List the tables in Casuser to see the newly created table, class_from_SFcaslib, that you loaded.

List information about the newly loaded table, including column names, data types, and so on.

---

**Spark Data Connector**

Enables you to load data into CAS from Spark or Hadoop that resides on a file system, including HDFS.

**Valid in:** CASLIB statement, PROC CASUTIL statements (see options for details)

**Requirement:** To transfer data in parallel requires a separate license to SAS In-Database Technologies for Hadoop and a massively parallel processing (MPP) system.

**Notes:** All users with SAS/ACCESS Interface to Hadoop who run SAS Viya can use the serial SAS Data Connector to Spark.

Support for this data connector was added in SAS Viya 3.4.

**Examples:**

Establish a connection between your Spark data source and SAS Cloud Analytic Services using serial mode.

```plaintext
caslib sparkcaslib desc='Spark Caslib'
dataSource=(srctype='spark',
            dataTransferMode="serial",
            hadoopjarpath="/hadoop/jars:hadoop/spark/jars",
            hadoopconfigdir="/hadoop/conf",
            username="hive",
            server="thriftserver",
            schema="default");
```

Establish a connection between your Spark data source and SAS Cloud Analytic Services using parallel mode.

```plaintext
caslib sparkcaslib desc='Spark Caslib'
dataSource=(srctype='spark',
            dataTransferMode="parallel",
            hadoopjarpath="/hadoop/jars:hadoop/spark/jars",
            hadoopconfigdir="/hadoop/conf",
            username="hive",
            server="thriftserver",
            schema="default");
```

Load a Spark data source using PROC CASUTIL.

```plaintext
proc casutil
    incaslib=sparkcaslib outcaslib="casuser" sessref=mysess;
load incaslib=sparkcaslib casdata="cars"
Syntax

Data Connector Options for Spark

Data connector options are used in the context of different statements that connect your data in Spark or Hadoop with CAS. 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 and action calls, see "Where to Specify Data Connector Options" on page 160.

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [see Requirement]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>AUTHDOMAIN=</td>
</tr>
<tr>
<td>Requirement</td>
<td>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.</td>
</tr>
<tr>
<td>Supports</td>
<td>HDMD, Hive, parallel data transfer, serial data transfer</td>
</tr>
</tbody>
</table>

**BUFFERSIZE=bytes | "nB" | "nK" | "nM" | "nG" | "nT"**

specifies the buffer size that is used to send data to or receive data from SAS embedded processes. If you set this value in PROC CASUTIL, then this value overrides a value that you set in the CASLIB statement. Increasing the size might result in better performance with a trade-off of increased memory usage.

You can specify this value as an integer (bytes) or with a suffix, such as "4M" to indicate four megabytes. Supported suffixes are "B" (bytes), "K" (kilobytes), "M" (megabytes), "G" (gigabytes), and "T" (terabytes). Suffixes are case insensitive.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1048576</td>
</tr>
</tbody>
</table>
Restriction: This option applies only with parallel data transfer for Spark (DATATRANSFERMODE="parallel").

Requirement: You must specify a value that is greater than 0.

Supports: HDMD, Hive, parallel data transfer

Tip: If you experience memory errors during data transfer, reduce the value of BUFFERSIZE=.

**CHARMULTIPLIER=value**

Specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set. Because all Hive tables are UTF-8 tables, this value does not affect Hive.

Valid in: CASLIB statement

PROC CASUTIL: LOAD and SAVE statements

Default: 1.0

Range: 1.0–5.0

Supports: HDMD, parallel data transfer, serial data transfer

**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"

**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 or save the data in parallel using embedded processing. If this fails, a note is written to the SAS log and serial processing is then attempted.

- PARALLEL: Specifies to load or save the data in parallel by using the SAS Data Connect Accelerator to your database or data source.

- SERIAL: Specifies to load or save 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  

Restriction  
The DATATRANSFERMODE= option that is part of a caslib definition is not the same as the DATATRANSFERMODE= file import option. Although they share the same name and values, they are functionally different and therefore are not interchangeable. See “Where to Specify Data Transfer Modes” for details.

Interaction  
If NUMREADNODES= or NUMWRITENODES= is a value other than 1 and DATATRANSFERMODE="AUTO", SAS attempts first to transfer the data in parallel. 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 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 for all Hive string data types. For example, if `dbMaxText=100`, all Hive string data types are loaded as `VARCHAR(100)`.

Valid in | CASLIB statement  
| PROC CASUTIL: LOAD statement  

Alias | DBMAX_TEXT=  

Default | 1024  

Range | 1 (minimum)  

Supports | Hive, parallel data transfer, serial data transfer  

**HADOOPCONFIGDIR="configuration-files-directory"**  
specifies the Spark and Hadoop configuration files directory. These are the configuration files that are obtained by running the Hadoop tracer tool on the target cluster.

Valid in | CASLIB statement  

Default | none
HADOOPJARPATH="Hadoop-jar-files-path"
specifies one or more paths to the Spark and Hadoop JAR files. These are the
JAR files that you can obtain by running the Hadoop tracer tool on the target
cluster. These files are delimited by colons for Linux.

The tracer tool obtains the Spark configuration file and Spark JAR files along
with the Hadoop configuration and JAR files. The Spark JAR files are stored
under a subfolder called spark under the Hadoop JAR folder.

Class path order: On Hortonworks Hadoop, the Spark JAR folder must be the
last one on the JAR path. On Cloudera Hadoop, the Spark JAR folder must be
the first one on the JAR path.

If you are accessing Hive tables that require a customized Hive SerDe, you must
include the SerDe JAR files in the hadoopJarPath.

Valid in CASLIB statement
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
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
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
Default "/TMP"
Supports HDMD, Hive, parallel data transfer, serial data transfer
**MAXLOADTHREADS**="**maximum-number-of-threads**"

specifies the maximum number of threads to use per worker node.

Valid in  
CASLIB statement  
PROC CASUTIL: LOAD statement  

Default  
24  

Supports  
HDMD, Hive, parallel data transfer

**MAXSAVETHREADS**="**maximum-number-of-threads**"

specifies the maximum number of threads to use per worker node.

Valid in  
CASLIB statement  
PROC CASUTIL: SAVE statement  

Default  
24  

Supports  
parallel data transfer

**NCHARMULTIPLIER=** **value**

specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

Valid in  
CASLIB statement  
PROC CASUTIL: LOAD and SAVE statements  

Default  
3  

Supports  
HDMD, Hive, parallel data transfer, serial data transfer

**NUMREADBUFFERS=** **number**

specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in  
CASLIB statement  
PROC CASUTIL: LOAD statement  

Default  
4  

Range  
1-10  

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 lets a data connector 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.
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 lets a data connector 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 CASLIB statement
PROC CASUTIL: LOAD statement

Default 1
Supports HDMD, Hive, 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, DELETESOURCE, 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

PORT=port-value

specifies the Spark Thrift Server port number.

Valid in CASLIB statement
### 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.

**Default**: none  
**Supports**: Hive, parallel data transfer, serial data transfer  
**Valid in**: CASLIB statement

<table>
<thead>
<tr>
<th>Alias</th>
<th>Default</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Calculated automatically based on row size</td>
<td>This option is not valid with parallel data transfer (DATATRANSFERMODE=&quot;parallel&quot;). Specify BUFFERSIZE= when loading data in parallel.</td>
</tr>
</tbody>
</table>

### READBUFF="number-of-rows"

Specifies the number of rows to fetch per block of data retrieved.

**Default**: Calculated automatically based on row size  
**Supports**: Hive, parallel data transfer, serial data transfer  
**Valid in**: CASLIB statement, PROC CASUTIL: LOAD statement

### 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.

**Default**: 'default'  
**Supports**: Hive, parallel data transfer, serial data transfer  
**Valid in**: CASLIB statement, PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements
**SCRATCH_DB="Hive-schema-name"**
specifies the Hive database to use when creating temporary tables and views.

- **Valid in:** CASLIB statement
- **Default:** none
- **Supports:** Hive, parallel data transfer, serial data transfer

**SERVER="server-identifier"**
specifies the name of the node in the Hadoop cluster where the Spark2 Thrift Server is running.

- **Valid in:** CASLIB statement
- **Default:** none
- **Supports:** Hive, parallel data transfer, serial data transfer

**SLICECOLUMN="column-name"**
specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

- **Valid in:** CASLIB statement
- **Default:** none
- **Interaction:** Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.
- **Supports:** serial data transfer

**SRCTYPE="spark"**
specifies that the data source is Spark (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 parallel data transfer (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

**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, DELETESOURCE, 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

Details

Use Options

Data connector options are used in the context of different statements that connect your data in Spark 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 160.

Spark 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 (Spark) 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.

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 Spark 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 Spark 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 Spark or Hadoop LIBNAME engine creates a SASHDMD file when a new table is created in Spark or 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 Spark 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 options that are specific to HDMD that are on a Hive caslib.

Table 7.22  Options for Specifying a Hive or HDMD Caslib for Spark

<table>
<thead>
<tr>
<th>HDFSMETADIR= Option</th>
<th>SERVER= Option</th>
<th>URI= Option</th>
<th>Caslib Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>specified</td>
<td>specified</td>
<td>specified</td>
<td>HDMD</td>
</tr>
<tr>
<td></td>
<td></td>
<td>not specified</td>
<td>HDMD</td>
</tr>
<tr>
<td></td>
<td>not specified</td>
<td>specified</td>
<td>not specified</td>
</tr>
<tr>
<td>not specified</td>
<td>specified</td>
<td>specified</td>
<td>Hive, using URI=</td>
</tr>
<tr>
<td></td>
<td></td>
<td>not specified</td>
<td>Hive, using the</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>configuration that is</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>found in HadoopConfigPath</td>
</tr>
<tr>
<td></td>
<td>not specified</td>
<td>specified</td>
<td>not specified</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Hive server must</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>be high availability</td>
</tr>
</tbody>
</table>

Supported Hive and HDMD Data Types

The Spark data connector supports these data types for loading Hive and HDMD data into SAS Cloud Analytic Services. This table 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 7.23  Supported Hive and HDMD Data Types for Spark

<table>
<thead>
<tr>
<th>Hive Data Type</th>
<th>HDMD Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character Data Types</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>STRING</td>
<td>VARCHAR</td>
<td></td>
</tr>
</tbody>
</table>
### Numeric Data Types

<table>
<thead>
<tr>
<th>Hive Data Type</th>
<th>HDMD Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ARRAY</td>
<td></td>
<td>not supported</td>
</tr>
<tr>
<td>BIGINT</td>
<td>BIGINT</td>
<td>INT64</td>
</tr>
<tr>
<td>BINARY</td>
<td>BINARY</td>
<td>VARCHAR (for serial) not supported (for parallel)</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td></td>
<td>INT32 (for serial) not supported (for parallel)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DECIMAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INT</td>
<td>INT32</td>
<td></td>
</tr>
<tr>
<td>INTEGER</td>
<td>INT32</td>
<td></td>
</tr>
<tr>
<td>INTERVAL</td>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td>MAP</td>
<td></td>
<td>not supported</td>
</tr>
<tr>
<td>REAL</td>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td>INT32</td>
</tr>
<tr>
<td>STRUCT</td>
<td></td>
<td>not supported</td>
</tr>
<tr>
<td>TINYINT</td>
<td>TINYINT</td>
<td>INT32</td>
</tr>
<tr>
<td>UNIONTYPE</td>
<td></td>
<td>not supported</td>
</tr>
</tbody>
</table>

### Date and Time Data Types

<table>
<thead>
<tr>
<th>Hive Data Type</th>
<th>HDMD Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>DATE</td>
<td>DATE</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>TIME</td>
<td>TIME</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>TIMESTAMP</td>
<td></td>
<td>DOUBLE</td>
</tr>
</tbody>
</table>

### 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 30.

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 Spark. 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 sparkcaslib.

```sas
  caslib sparkcaslib desc='spark caslib'
  dataSource=(srctype='spark',
    dataTransferMode='parallel',
    server='thriftserver',
    username='myuser1',
    password='myPwd',
    hadoopJarPath='/hadoop/jars:/hadoop/spark/jars',
    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 Spark. 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 sparkcaslib.

```sas
  caslib hdmdlib desc='hdmd caslib'
  dataSource=(srctype='spark',
    dataTransferMode='serial',
    username='myuser1',
    hadoopJarPath='/hadoop/jars:/hadoop/spark/jars',
    hadoopConfigDir='Hadoop_configuration_directory',
    hdfsDataDir='HDFS data directory',
    schema='HDFS data directory');
```
Example 3: Load a Hive Table into SAS Cloud Analytic Services Using PROC CASUTIL

```sas
proc casutil;
    list files incaslib="sparkcaslib";  \1
    load casdata="mySKdata" incaslib="sparkcaslib" outcaslib="casuser"
        casout="SKdata_from_sparkcaslib";  \2
    list tables incaslib="casuser";  \3
    contents casdata="SKdata_from_sparkcaslib" incaslib="casuser";  \4
quit;
```

1. List the tables in sparkcaslib before loading your data.
2. Load the table mySKdata from Hive into the caslib, casuser. Call the new table SKdata_from_sparkcaslib.
3. List the tables in caslib casuser to see the newly created SKdata_from_sparkcaslib 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

```sas
proc casutil;
    list files incaslib="hdmdlib";  \1
    load casdata="myHDdata" incaslib="hdmdlib" outcaslib="casuser2"
        casout="SKdata_from_hdmdlib";  \2
    list tables incaslib="casuser2";  \3
    contents casdata="SKdata_from_hdmdlib" incaslib="casuser2";  \4
quit;
```

1. List the tables in HDMDcaslib before loading your data.
2. Load the table mySKdata from Hive into the caslib, casuser2. Call the new table SKdata_from_hdmdlib.
3. List the tables in caslib casuser2 to see the newly created SKdata_from_hdmdlib table that you loaded.
4. List information about the newly loaded table, including column names, data types, and so on.

SPD Engine Files Data Connector

Enables you to load data into CAS from SPD Engine files that reside on a file system, including HDFS.

**Valid in:** CASLIB statement, PROC CASUTIL statements (see options for details)

**Restriction:** When you use parallel data transfer mode, files cannot be encrypted or compressed, and tables cannot require a Read password.
Requirement: To transfer data in parallel requires a separate license to SAS In-Database Technologies for Hadoop and a massively parallel processing (MPP) system.

Notes: All users with SAS/ACCESS Interface to SPD Engine Files can use the serial SAS Data Connector to SPD Engine Files.
Support for this data connector was added in SAS Viya 3.3.

Examples: Establish a connection between your SPD Engine files data source and SAS Cloud Analytic Services using serial mode.

```plaintext
caslib spdecaslib desc='SPDE Caslib' dataSource=(srctype='spde', dataTransferMode="serial", mdfpath="/user/MyMDFFilepath");
```

Establish a connection between your SPD Engine Files data source and SAS Cloud Analytic Services using serial mode.

```plaintext
caslib spdecaslib desc='SPDE Caslib' dataSource=(srctype='spde', dataTransferMode="serial", hdfs=true, hadoopjarpath="/hadoop/jars", hadoopconfigdir="/hadoop/conf", mdfpath="/user/spdemdfs");
```

Establish a connection between your SPD Engine Files data source and SAS Cloud Analytic Services using parallel mode.

```plaintext
caslib spdecaslib desc='SPDE Caslib' dataSource=(srctype='spde', dataTransferMode="parallel", hadoopjarpath="/hadoop/jars", hadoopconfigdir="/hadoop/conf", mdfpath="/user/spdemdfs");
```

Load an SPD Engine Files data source using PROC CASUTIL.

```plaintext
proc casutil;
   incaslib="spdecaslib" sessref=mysess;
   load incaslib="spdecaslib" casdata="cars" casout="cars_CAS"
      options=(dataTransferMode="serial");
run;
quit;
```

Syntax

Data Connector Options for SPD Engine Files

Data connector options are used in the context of different statements that connect your data in SPD Engine files with CAS. 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 and action calls, see “Where to Specify Data Connector Options” on page 160.

**ALTERPW="password"**

specifies the password to access and modify or delete a password-protected SPD Engine table.
**Valid in**  
CASLIB statement

**PROC CASUTIL: DELETESOURCE, LOAD, and SAVE statements**

**Default**  
none

**Supports**  
parallel data transfer (SAVE statement for newly created tables), 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 valid only 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 | "nB" | "nK" | "nM" | "nG" | "nT"**

specifies the buffer size that is used to send data to or receive data from SAS embedded processes. If you set this value in PROC CASUTIL, then this value overrides a value that you set in the CASLIB statement. Increasing the size might result in better performance with a trade-off of increased memory usage.

You can specify this value as an integer (bytes) or with a suffix, such as "4M" to indicate four megabytes. Supported suffixes are "B" (bytes), "K" (kilobytes), "M" (megabytes), "G" (gigabytes), and "T" (terabytes). Suffixes are case insensitive.

**Valid in**  
CASLIB statement

**PROC CASUTIL: LOAD and SAVE statements**

**Default**  
1048576

**Requirement**  
You must specify a value that is greater than 0.

**Supports**  
parallel data transfer

**Tip**  
If you experience memory errors during data transfer, reduce the value of BUFFERSIZE=. 
### CHARMULTIPLIER=value

specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: LOAD statement</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>1.0</td>
</tr>
<tr>
<td>Range</td>
<td>1.0–5.0</td>
</tr>
</tbody>
</table>

### COMPRESS="YES" | "NO" | "BINARY" | "CHAR"

specifies the compression method to use for the SPD Engine table.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: SAVE statement</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>NO</td>
</tr>
<tr>
<td>Supports</td>
<td>serial data transfer</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: CONTENTS, LIST, LOAD statements</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Example</td>
<td>conopts=&quot;QueryTimeout=0&quot;</td>
</tr>
</tbody>
</table>

### DATAPATH="DPF-file-path"

specifies the path where to place the DPF files for the SPD Engine table.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: SAVE statement</td>
<td></td>
</tr>
<tr>
<td>Default</td>
<td>MDFPATH=</td>
</tr>
</tbody>
</table>

### DATATRANSFERMODE="AUTO" | "PARALLEL" | "SERIAL"

specifies the mode of data transfer.

Here are the valid values.

AUTO specifies to first try to load or save the data in parallel using embedded processing. If this fails, a note is written to the SAS log and serial processing is then attempted.
### PARALLEL
specifies to load or save the data in parallel by using the SAS Data Connect Accelerator to your database or data source.

### SERIAL
specifies to load or save the data serially by using the SAS Data Connector to your database or data source.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD and SAVE statements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Aliases</th>
<th>DATATRANSFER=</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>DTM=</td>
</tr>
</tbody>
</table>

| Default | SERIAL |

| Restriction | The DATATRANSFERMODE= option that is part of a caslib definition is not the same as the DATATRANSFERMODE= file import option. Although they share the same name and values, they are functionally different and therefore are not interchangeable. See "Where to Specify Data Transfer Modes" for details. |

### Requirement
To use the PARALLEL option, you must have a license for the SAS Data Connect Accelerator for Hadoop, which is part of SAS In-Database Technologies.

### Interaction
If NUMREADNODES= or NUMWRITENODES= is a value other than 1 and DATATRANSFERMODE="AUTO", SAS attempts first to transfer the data in parallel. 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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>

| 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).

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>
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, serial data transfer

**HDFSTEMPDIR=**"temporary-directory"

specifies the Hadoop HDFS directory to use to store temporary data.

Valid in | CASLIB statement
Default | "/tmp"
Supports | parallel data transfer
**LABEL=**"label-name"

specifies a label name for the created SPD Engine table.

- **Valid in**: CASLIB statement
- **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, DELETESOURCE, LOAD, and SAVE statements

- **Default**: none

- **Supports**: parallel data transfer (SAVE statement for newly created tables), serial data transfer (LOAD and SAVE statements)

**MAXLOAD_THREADS=**"maximum-number-of-threads"

specifies the maximum number of threads to use per worker node.

- **Valid in**: 
  
  PROC CASUTIL: LOAD statement

- **Default**: 24

- **Supports**: parallel data transfer

**MAXSAVE_THREADS=**"maximum-number-of-threads"

specifies the maximum number of threads to use per worker node.

- **Valid in**: 
  
  PROC CASUTIL: SAVE statement

- **Default**: 24

- **Supports**: parallel data transfer

**MDFPATH=**"SPDE-MDF-file-directory"

specifies the HDFS or local disk directory that contains the SPD Engine MDF file.

- **Valid in**: 
  
  PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements

- **Default**: none

- **Supports**: parallel data transfer, serial data transfer
**NUMREADBUFFERS=number**
specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in  
- CASLIB statement  
- PROC CASUTIL: LOAD statement  

Default 4  
Range 1-10

**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 lets a data connector 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  
- CASLIB statement  
- 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.
<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [see Requirement]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Aliases</strong> PASS=</td>
<td>PWD=</td>
</tr>
<tr>
<td><strong>Default</strong> none</td>
<td></td>
</tr>
<tr>
<td><strong>Restrictions</strong> This option is valid only when SPD Engine files are stored in Hadoop (HDFS=TRUE). Parallel processing has limited support for table passwords.</td>
<td></td>
</tr>
<tr>
<td><strong>Requirement</strong> 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.</td>
<td></td>
</tr>
<tr>
<td><strong>Supports</strong> parallel data transfer, serial data transfer</td>
<td></td>
</tr>
<tr>
<td><strong>PLATFORM=</strong> &quot;mapred&quot;</td>
<td>&quot;spark&quot;</td>
</tr>
<tr>
<td>specifies the platform on which to run the SAS Embedded Process job as a MapReduce job or on Spark.</td>
<td></td>
</tr>
<tr>
<td><strong>Valid in</strong> CASLIB statement</td>
<td></td>
</tr>
<tr>
<td><strong>PROC CASUTIL: LOAD and SAVE statements</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Default</strong> mapred</td>
<td></td>
</tr>
<tr>
<td><strong>Supports</strong> parallel data transfer</td>
<td></td>
</tr>
<tr>
<td><strong>READPW=</strong> &quot;password&quot;</td>
<td></td>
</tr>
<tr>
<td>specifies the password to access and read a password-protected SPD Engine table.</td>
<td></td>
</tr>
<tr>
<td><strong>Valid in</strong> CASLIB statement</td>
<td></td>
</tr>
<tr>
<td><strong>PROC CASUTIL: CONTENTS, LOAD, and SAVE statements</strong></td>
<td></td>
</tr>
<tr>
<td><strong>Default</strong> none</td>
<td></td>
</tr>
<tr>
<td><strong>Supports</strong> parallel data transfer (SAVE statement for newly created tables), serial data transfer (LOAD and SAVE statements)</td>
<td></td>
</tr>
<tr>
<td><strong>SLICECOLUMN=</strong> &quot;column-name&quot;</td>
<td></td>
</tr>
<tr>
<td>specifies the column to use when partitioning data for a multinode data Read. Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.</td>
<td></td>
</tr>
<tr>
<td><strong>Valid in</strong> CASLIB statement</td>
<td></td>
</tr>
</tbody>
</table>
**INTERACTION**

Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

**Supports**  
serial data transfer

**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 parallel data transfer (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, DELETESOURCE, LIST, LOAD, and SAVE statements**

**Alias**  
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 USER_NAME= and PASSWORD= values or by specifying an AUTHENTICATION_DOMAIN= 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
The SPD Engine Files data connector supports these data types for loading data from SPD Engine files into SAS Cloud Analytic Services. This table 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>
<thead>
<tr>
<th>SPD Engine Files Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
</tr>
</tbody>
</table>

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 30.

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.

```plaintext
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

```plaintext
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";
```
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.

---

**Teradata Data Connector**

Enables you to transfer data between Teradata and CAS.

**Valid in:**

CASLIB statement, PROC CASUTIL statements (see options for details)

**Requirement:**

To transfer data in parallel requires a separate license to SAS In-Database Technologies for Teradata.

**Note:**

All users with SAS/ACCESS Interface to Teradata can use the serial SAS Data Connector to Teradata.

**Examples:**

Create a caslib that connects Teradata and CAS.

```r
caslib tdlib
datasource=(
srctype='teradata'
dataTransferMode='parallel'
server='TDserver'
username='user1'
password='myPwd'
database='TDdatabase');
```

Load Teradata source data using PROC CASUTIL.

```r
proc casutil;
load incaslib='tdlib' casdata='cars'
casout='cars_CAS'
options=(dataTransferMode='parallel');
quit;
```

---

**Syntax**

**Data Connector Options for Teradata**

Data connector options are used in the context of different statements that connect your data in Teradata with CAS. 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 160.
**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" | "nB" | "nK" | "nM" | "nG" | "nT"

specifies the buffer size that is used to send data to or receive data from SAS embedded processes. If you set this value in PROC CASUTIL, then this value overrides a value that you set in the CASLIB statement. Increasing the size might result in better performance with a trade-off of increased memory usage.

You can specify this value as an integer (bytes) or with a suffix, such as "4M" to indicate four megabytes. Supported suffixes are "B" (bytes), "K" (kilobytes), "M" (megabytes), "G" (gigabytes), and "T" (terabytes). Suffixes are case insensitive.

Valid in
- CASLIB statement
- PROC CASUTIL: LOAD and SAVE statements

Default 1048576

Restriction This option applies only with parallel data transfer for Teradata (DATATRANSFERMODE="parallel").

Requirement You must specify a value that is greater than 0.

Tip If you experience memory errors during data transfer, reduce the value of BUFFERSIZE=.

**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

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>UTF-8</td>
</tr>
<tr>
<td>Restriction</td>
<td>This option applies only to serial data transfer (DATATRANSFERMODE= &quot;serial&quot;).</td>
</tr>
</tbody>
</table>

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [required]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>DB=</td>
</tr>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Interaction</td>
<td>If you supply values for SCHEMA= and DATABASE=, the value for SCHEMA= is used to qualify a table name.</td>
</tr>
</tbody>
</table>

**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 or save the data in parallel using embedded processing. If this fails, a note is written to the SAS log and serial processing is then attempted.

**PARALLEL** specifies to load or save the data in parallel by using the SAS Data Connect Accelerator to your database or data source.

**SERIAL** specifies to load or save the data serially by using the SAS Data Connector to your database or data source.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aliases</td>
<td>DATATRANSFER=</td>
</tr>
<tr>
<td>Default</td>
<td>SERIAL</td>
</tr>
<tr>
<td>Restriction</td>
<td>The DATATRANSFERMODE= option that is part of a caslib definition is not the same as the DATATRANSFERMODE= file</td>
</tr>
</tbody>
</table>
import option. Although they share the same name and values, they are functionally different and therefore are not interchangeable. See "Where to Specify Data Transfer Modes" for details.

### Requirement
To use the PARALLEL option, you must have a license for the SAS Data Connect Accelerator for Teradata, which is part of SAS In-Database Technologies.

### Interaction
If NUMREADNODES= or NUMWRITENODES= is a value other than 1 and DATATRANSFERMODE="AUTO", SAS attempts first to transfer the data in parallel. 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.

- **Note**: By default for Teradata, no primary indexes are assigned when you save a table. To specify a primary index, you can use this option. (See the example.)

<table>
<thead>
<tr>
<th>Valid in</th>
<th>PROC CASUTIL: SAVE statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
<tr>
<td>Restriction</td>
<td>The DBCREATETABLEOPTS= option applies to serial data transfer only.</td>
</tr>
<tr>
<td>Example</td>
<td>Assign a primary key to the column Idnum when you save a table: options={dbcreatetableopts=&quot;unique primary index(idnum)&quot;}</td>
</tr>
</tbody>
</table>

**DBMAXTEXT=**maximum-string-column-length
specifies the maximum length for all string data types. For example, if dbMaxText=100, all string data types are loaded as VARCHAR(100).

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: LOAD statement</td>
</tr>
<tr>
<td>Alias</td>
<td>DBMAX_TEXT=</td>
</tr>
<tr>
<td>Default</td>
<td>32000</td>
</tr>
<tr>
<td>Range</td>
<td>1–32000</td>
</tr>
<tr>
<td>Supports</td>
<td>serial data transfer</td>
</tr>
</tbody>
</table>

**INSERTBUFF="number-of-rows"**
specifies the number of rows per block of data to save to a table in the data source.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: SAVE statement</td>
</tr>
<tr>
<td><strong>Option</strong></td>
<td><strong>Description</strong></td>
</tr>
<tr>
<td>------------</td>
<td>-----------------</td>
</tr>
<tr>
<td><strong>MAXLOADTHREADS</strong></td>
<td>Specifies the maximum number of threads to use per worker node.</td>
</tr>
<tr>
<td><strong>MAXSAVETHREADS</strong></td>
<td>Specifies the maximum number of threads to use per worker node.</td>
</tr>
<tr>
<td><strong>NCHARMULTIPLIER</strong></td>
<td>Specifies the number of bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.</td>
</tr>
<tr>
<td><strong>NUMREADBUFFERS</strong></td>
<td>Specifies the number of buffers that the data connector is to read and convert from the data source to CAS.</td>
</tr>
<tr>
<td><strong>NUMREADNODES</strong></td>
<td>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 lets a data connector 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.</td>
</tr>
</tbody>
</table>
Valid in | CASLIB statement  
| PROC CASUTIL: LOAD statement  
| PROC CASUTIL: SAVE statement  

**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 lets a data connector 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 | CASLIB statement  
| PROC CASUTIL: SAVE statement  

**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, DELETESOURCE, LIST, LOAD, and SAVE statements  

**READBUFF=“number-of-rows”**  
specifies the number of rows to fetch per block of data retrieved.

Valid in | CASLIB statement  
| PROC CASUTIL: LOAD statement  

**Restriction**  
The NUMREADNODES= option applies to serial data transfer only.

**Restriction**  
The NUMWRITENODES= option applies to serial data transfer only.

**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.
Default Calculated automatically based on row size

Restriction This option is not valid with parallel data transfer (DATATRANSFERMODE="parallel"). Specify BUFFERSIZE= when loading data in parallel.

**ROLE="name"**

specifies the Teradata role name.

Valid in CASLIB statement

Default none

Restriction This option is not valid with parallel data transfer (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, DELETESOURCE, 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

**SLICECOLUMN="column-name"**

specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

Valid in CASLIB statement

Default none
### Interaction

Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

### Supports

serial data transfer

### 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
- **Default**: 0 (no message)
- **Restriction**: This option applies only to parallel data transfer (DATATRANSFERMODE="PARALLEL").

### 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]
- **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

How Parallel Processing Works
If the SAS Data Connect Accelerator for Teradata is licensed for your system, then you can use it to load data in parallel. The SAS Data Connect Accelerator for Teradata is part of SAS In-Database Technologies. To load data in parallel, specify DATATRANSFERMODE="parallel". When you load data in parallel, SAS In-Database Technologies use the Teradata database 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.

Case-Sensitivity and Filtering Data
When using parallel data transfer, ensure case-insensitive filtering of data by using the UPCASE function. For example, suppose you have this Sample data set:

<table>
<thead>
<tr>
<th>id</th>
<th>col1</th>
<th>col2</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>ABCD1234</td>
<td>bcde2346</td>
</tr>
<tr>
<td>4</td>
<td>aBcD1234</td>
<td>BcDe2345</td>
</tr>
<tr>
<td>5</td>
<td>aacD1234</td>
<td>BcDe2345</td>
</tr>
<tr>
<td>1</td>
<td>abcd1234</td>
<td>BCDE2345</td>
</tr>
<tr>
<td>2</td>
<td>ABCD1234</td>
<td>bcde2345</td>
</tr>
</tbody>
</table>

Use the UPCASE function to ensure that you get a case-insensitive match for the values in Col1. In this example, you are loading data from the TDsample table in the Tddata caslib. The resulting table, ABCDdata, contains all observations where the Col1 value matches ‘ABCD1234’, regardless of the original case of the data.

Note: You are not changing the case of the data in the resulting table.

```
caslib tddata datasource=(srctype='teradata' dataTransferMode='parallel' server='TDserver' username='user1' password='myPwd' database='TDdatabase'));

proc casutil;
   load casdata="TDsample" incaslib=tddata outcaslib="casuser" casout="ABCDdata" where="upcase(col1)='ABCD1234'";
run;
quit;
```

The resulting ABCDdata table contains observations for ID values 1, 2, 3, and 4.
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 provide 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.

```sas
libname x teradata user="janedoe@TD2" password="janedoeword";
```

This example shows how to connect using a TD2 user name:

```sas
caslib tdlib
data source=(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>
<thead>
<tr>
<th>Teradata Data Type</th>
<th>CAS Data Type</th>
<th>Default Format in CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Character</td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>$CHARw.</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
<td>$CHARw.</td>
</tr>
<tr>
<td>Numeric</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DOUBLE</td>
<td>w.d</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>w.d</td>
</tr>
<tr>
<td>NUMBER</td>
<td>DOUBLE</td>
<td>w.d</td>
</tr>
<tr>
<td>INTEGER</td>
<td>INTEGER</td>
<td>w.</td>
</tr>
<tr>
<td>BYTEINT (I1)</td>
<td>INTEGER</td>
<td>w.</td>
</tr>
<tr>
<td>SMALLINT (I2)</td>
<td>INTEGER</td>
<td>w.</td>
</tr>
<tr>
<td>Teradata Data Type</td>
<td>CAS Data Type</td>
<td>Default Format in CAS</td>
</tr>
<tr>
<td>--------------------</td>
<td>---------------</td>
<td>-----------------------</td>
</tr>
<tr>
<td>BIGINT (I8)</td>
<td>BIGINT</td>
<td>w.</td>
</tr>
</tbody>
</table>

**Date and Time**

<table>
<thead>
<tr>
<th>DATE (DA)</th>
<th>DOUBLE</th>
<th>DATEw.</th>
</tr>
</thead>
<tbody>
<tr>
<td>TIME (AT)</td>
<td>DOUBLE</td>
<td>TIMEw.d</td>
</tr>
<tr>
<td>TIMESTAMP (TS)</td>
<td>DOUBLE</td>
<td>DATETIMEw.d</td>
</tr>
</tbody>
</table>

**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.

Because data is stored in CAS with encoding UTF-8, which uses the UNICODE character set, the maximum length for CHAR and VARCHAR data in CAS is 32000 for Teradata. If you attempt to load data that is longer than this, you receive an error in the SAS log.

For more information, see “VARCHAR Support for Implicit and Explicit Data Type Conversion” on page 30.

**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: Load Teradata Data into SAS Cloud Analytic Services Using PROC CASUTIL**

```sas
Caslib TDlib desc='Teradata Caslib'
dataSource=(srctype='Teradata',
dataTransferMode='serial',
server='teradataServer',
username='user1',
password='******',
```
proc casutil;
  list files incaslib="Teradatacaslib";
  load casdata="myTDdata" incaslib="Teradatacaslib" outcaslib="casuser" casout="TDdata_from_Teradatacaslib";
  list tables incaslib="casuser";
  contents casdata="TDdata_from_Teradatacaslib" incaslib="casuser";
quit;

1 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. For this CASLIB statement, the data is read serially into the caslib Teradatacaslib.

2 List the tables in Teradatacaslib before loading your data.

3 Load the table myTDdata from Teradata into caslib Casuser. Call the new table TDdata_from_Teradatacaslib.

4 List the tables in caslib to see the newly created table, TDdata_from_Teradatacaslib, that you loaded.

5 List information about the newly loaded table, including column names, data types, and so on.

Vertica Data Connector

Enables you to load data from Vertica into CAS.

Valid in: CASLIB statement, PROC CASUTIL statements (see options for details)

Notes: All users with SAS/ACCESS Interface to Vertica can use SAS Data Connector to Vertica. Support for this data connector was added in SAS Viya 3.4.

Examples: Establish a connection between your Vertica database and SAS Cloud Analytic Services.

```plaintext
caslib verticacaslib desc='Vertica Caslib'
  dataSource=(
    srctype='vertica'
    server='VTserver'
    username='user1'
    password='myPwd'
    database='VTdatabase-name'
    schema='myschema');
```

Overriding the user and password values.

```plaintext
proc casutil;
  load casdata='vtexamp' casout='myVTdata' incaslib='verticacaslib'
    outcaslib='casuser'
    dataSourceOptions=
      (username='user5'
      password='myPwd');
quit;
```
syntax

Data Connector Options for Vertica

Data connector options are used in the context of different statements that connect your data in Vertica with CAS. 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 160.

**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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [see Requirement]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PROC CASUTIL: CONTENTS, DELETESOURCE, LIST, LOAD, and SAVE statements</td>
</tr>
</tbody>
</table>

**Alternate Name**

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.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
</table>

**Default**

Active caslib

**CHARMULTIPLIER=value**

specifies the number of bytes per character 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 option applies to data that is encoded using a single-byte character set.

This value overrides a value of CHARMULTIPLIER that was set in the CASLIB statement.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
</table>

**Default**

1.0
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.

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

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

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

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
**NCHARMULTIPLIER=value**

specifies the bytes per character of variable-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string.

This option applies to data that is encoded with a multibyte character set, such as UTF-8. All CAS tables are encoded as UTF-8.

Valid in  
CASLIB statement  
PROC CASUTIL: LOAD and SAVE statements  

Default 3

**NUMREADBUFFERS=number**

specifies the number of buffers that the data connector is to read and convert from the data source to CAS.

Valid in  
CASLIB statement  
PROC CASUTIL: LOAD statement  

Default 4  
Range 1-10

**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 lets a data connector 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  
CASLIB statement  
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 lets a data connector 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  
CASLIB statement  
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

**Vertica Data Connector 365**
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, DELETESOURCE, 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 Vertica database.

**Valid in**
- CASLIB statement

**Default**
5433

**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, DELETESOURCE, LIST, LOAD, and SAVE statements

**Default**
none

**SERVER=**"server-identifier"

specifies the server identifier for the DBMS server.

**Valid in**
- CASLIB statement [required]
SLICECOLUMN="column-name"

specifies the column to use when partitioning data for a multinode data Read.

Ideally, the column that you select should contain integer data with a relatively small number of distinct values that are fairly evenly distributed. Columns with continuous numeric values can work as well. SAS uses the MOD function, applied to the values in the specified column, to assign data to a partition. Each partition is assigned to a node that performs the Read operation for its slice of data.

Valid in: CASLIB statement
Default: none

Interaction: Multinode data Read is not on by default. Use NUMREADNODES= to specify a number.

SRCTYPE="vertica"

specifies that the data source is a Vertica 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, DELETESOURCE, LIST, LOAD, and SAVE statements

Aliases: UID=
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 Vertica

All quoted values are passed to the Vertica 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 Vertica database.

Supported Vertica Data Types

The Vertica data connector supports these data types for loading data from Vertica into CAS. This table 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>
<thead>
<tr>
<th>Data Type</th>
<th>HDMD Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Character Data Types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>(binary) STRING</td>
<td></td>
<td>VARCHAR</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td><strong>Numeric Data Types</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>BIGINT (INT64)</td>
<td>BIGINT</td>
<td>INT64</td>
</tr>
<tr>
<td>BINARY</td>
<td>BINARY</td>
<td>ERROR (not supported)</td>
</tr>
<tr>
<td>BOOLEAN</td>
<td></td>
<td>ERROR (not supported)</td>
</tr>
<tr>
<td>DECIMAL</td>
<td>DECIMAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td>FLOAT</td>
<td></td>
<td>DOUBLE</td>
</tr>
<tr>
<td>INTEGER or INT</td>
<td></td>
<td>INT32</td>
</tr>
<tr>
<td>NUMERIC</td>
<td></td>
<td>DOUBLE</td>
</tr>
</tbody>
</table>
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 Vertica Database as a Data Source for SAS Cloud Analytic Services

Use the CASLIB statement to establish a connection between your Vertica source data and a caslib, Verticacaslib. All options that are supplied in this example are required in the CASLIB statement except the PASSWORD= option.

In this example, the Vertica data is stored at the location designated by the SERVER= and DATABASE= options.

```
caslib Verticacaslib desc='Vertica Caslib'
dataSource=(srctype='vertica',
    server='VTserver',
    username='user1',
    password='myPwd',
    database='VTdatabase',
    schema='myschema');
```

Example 2: Load Vertica Data into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
    list files incaslib="Verticacaslib";
    load casdata="myVTdata" incaslib="Verticacaslib" outcaslib="casuser" casout="class_from_Verticacaslib";
    list files incaslib="casuser";
```

---

<table>
<thead>
<tr>
<th>Data Type</th>
<th>HDMD Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>REAL</td>
<td>REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>SMALLINT</td>
<td>INT32</td>
</tr>
<tr>
<td>TINYINT</td>
<td>TINYINT</td>
<td>INT32</td>
</tr>
</tbody>
</table>
List the tables in Verticacaslib before loading your data.

Load the table myVTdata from Verticacaslib into caslib Casuser. Call the new table class_from_Verticacaslib.

Note: You must specify Vertica table names using the capitalization that is used in the database.

List the tables in Casuser to see the newly created table, class_from_Verticacaslib, that you loaded.

List information about the newly loaded table, including column names, data types, and so on.
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.

SAS Cloud Analytic Services does not support the LENGTH statement for numeric data types and does not support DOUBLE columns that use less than 8 bytes. If you use the LENGTH statement to store numeric data with less than 8 bytes and then load that data into CAS, the server stores the data with 8 bytes. If you have several numeric columns that use less than 8 bytes, the in-memory table size can be much larger than what is required for a .sas7bdat file.

It is possible to specify a data type of INT32 when you import data. Be aware that the data uses 4 bytes, but the in-memory data structures align on 8-byte boundaries. As a result, even if INT32 can be specified for the data type, the memory use is 8 bytes.
## Data Types for SAS Cloud Analytic Services Table Columns

**Table 8.1** Data Types for SAS Cloud Analytic Services Table Columns

<table>
<thead>
<tr>
<th>Data Type Definition Keyword</th>
<th>SAS Cloud Analytic Services Table Column Data Type</th>
<th>Data Type Returned</th>
<th>Missing Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR(n)</td>
<td>CHAR(n)</td>
<td>CHAR(n)</td>
<td>all blanks (the same as in SAS)</td>
</tr>
<tr>
<td>VARCHAR(n)</td>
<td>VARCHAR(n)</td>
<td>all blanks or a zero length</td>
<td></td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td></td>
</tr>
</tbody>
</table>


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)
CLIBEXIST(uuid,caslib name)

Arguments

- **session name**: a valid session name.
- **caslib name**: a valid caslib name.
- **uuid**: specifies the uuid that you are trying to find. See "UUID=" on page 424 for more information about uuid.
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:

```sas
caslib _ALL_ list;
```

### Example

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>cas existingSession cashost=myhost casport=12345;</td>
<td></td>
</tr>
<tr>
<td>%put doIExist= %sysfunc(clibexist(mysess,notfound));</td>
<td>doIExist= 0</td>
</tr>
<tr>
<td>%put doIExist= %sysfunc(clibexist(existingSession,casuser));</td>
<td>doIExist=1</td>
</tr>
</tbody>
</table>

```sas
/* by uuid */
%put doIExist= %sysfunc(clibexist
d3177da7-0659-7a43-84ea-8f5797f619b2,casuser));
```

---

**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**

```sas
GETCASURL(<session>)
```

**Optional Argument**

`session`

If zero parameters are specified, then the SAS SESSREF= option value is used.

### Example: GETCASURL Example

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>%put httpaddr= %sysfunc(getcasurl());</td>
<td>httpaddr=<a href="http://host">http://host</a> and port value</td>
</tr>
</tbody>
</table>
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.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>libname mycas cas;</td>
<td></td>
</tr>
<tr>
<td>%put &quot;caslib name=&quot; %sysfunc(getlcaslib(mycas));</td>
<td>caslib name=casuser(userid);</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>caslib mycaslib datasource=(srctype=path) path=&quot;/tmp&quot;;</td>
<td></td>
</tr>
<tr>
<td>libname mycas cas caslib=mycaslib;</td>
<td></td>
</tr>
<tr>
<td>%put &quot;caslib name=&quot; %sysfunc(getlcaslib(mycas));</td>
<td>caslib name=mycaslib;</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cas mysess UUIDMAC=sessuuid;</code></td>
<td></td>
</tr>
<tr>
<td><code>libname mycas cas;</code></td>
<td></td>
</tr>
<tr>
<td><code>%put &quot;session name=&quot; %sysfunc(getl sessref(mycas));</code></td>
<td><code>session name=mysess;</code></td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>cas mysess UUIDMAC=sessuuid;</code></td>
<td></td>
</tr>
<tr>
<td><code>libname mycas cas tag=test.models;</code></td>
<td></td>
</tr>
<tr>
<td><code>%put &quot;tag=&quot; %sysfunc(getltag(mycas));</code></td>
<td><code>tag=test.models;</code></td>
</tr>
</tbody>
</table>
GETSESSOPT Function

Returns the value for a SAS Cloud Analytic Services session option.

Category: CAS

Syntax

GETSESSOPT(session-name, session-option-name)

GETSESSOPT(uuid, session-option-name)

Required Arguments

session-name
CAS session name.

session-option-name
CAS session option name.

For more information, see “Session Options”.

TIP You can list the session option names with this code:

CAS mysess LISTSESSOPTS;

uuid
specifies the uuid that you are trying to find. See “UUID=" on page 424 for more information about uuid.

Examples:

Example 1: Listing the Active CASLIB

This example returns the CASLIB option value using the session-name.

$put caslib=%sysfunc(GETSESSOPT(mysess, caslib));

caslib=CASUSER(userid)

data _null_;  
caslib = GETSESSOPT(mysess, caslib);  
put caslib=;  
run;

caslib=CASUSER(userid)
Example 2: Listing the Active CASLIB

This example returns the CASLIB option value using the $\textit{uuid}$.

```sas
data _null_;  
caslib = GETSESOPT(d3177da7-0659-7a43-84ea-8f5797f618b2, caslib);  
put caslib=;  
run;
```

caslib = CASUSER(userid)

---

SESSBUSY Function

Determines whether a CAS session is busy processing actions.

**Category:** CAS

**Syntax**

- `SESSBUSY(session-name)`
- `SESSBUSY(uuid)`

**Required Arguments**

- **$\textit{session-name}$**
  
  CAS session name.

- **$\textit{uuid}$**
  
  Specifies the $\textit{uuid}$ that you are trying to find. See “$\text{UUID}=$"session-uuid"" on page 424 for more information about $\textit{uuid}$.

**Details**

Actions can be submitted to the same session from multiple clients or submitted using ASYNC support. When a session is busy, submitted actions are queued and processed in the order received.

See “ASYNC=result name” in SAS Cloud Analytic Services: CASL Reference for information about the ASYNC= option.

SESSBUSY returns a 0 if the session is not busy and a 1 if the session is busy processing actions.
Example

Table 9.1  SESSBUSY Function

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>$put amIBusy= %sysfunc(sessbusy(mysess)); run;</td>
<td>amIBusy=0</td>
</tr>
<tr>
<td>$put amIBusy= %sysfunc(sessbusy(myBusySess)); run;</td>
<td>amIBusy=1</td>
</tr>
<tr>
<td>/* by uuid */ $put amIBusy= %sysfunc(sessbusy(d3177da7-0659-7a43-84ea-8f5797f618b2)); run;</td>
<td>amIBusy=0</td>
</tr>
</tbody>
</table>

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)
SESSFOUND (uuid)

Required Arguments

**session-name**
specifies the name of the session that you are trying to find.

**uuid**
specifies the uuid that you are trying to find. See “UUID=“session-uuid”” on page 424 for more information about uuid.

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.

Example: Determining CAS Sessions

This example shows whether the CAS session is found.
### Statements

<table>
<thead>
<tr>
<th>Statement</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>%put doIExist= %sysfunc(sessfound(mysess));</code></td>
<td>doIExist= 0</td>
</tr>
<tr>
<td><code>%put doIExist= %sysfunc(sessfound(existingSession)); run;</code></td>
<td>doIExist= 1</td>
</tr>
</tbody>
</table>
| /* by uuid */
| `%put %doIExist=%sysfunc(sessfound (d3177da7-0659-7a43-84ea-8f5797f618b2)); run;` | doIExist=0 |

---

Chapter 9 / Functions
Macro Variables

Dictionary

_CASHOST_ Macro Variable
Specifies the primary controller host name and the backup controller host name, if available, on the CAS server.

Default: The value of system option CASHOST, if set. Otherwise, not defined.

Range: 1–256 characters

Interactions: When set, this macro variable overrides SAS system option CASHOST. The CAS statement HOST= option overrides this macro variable.

Notes: When system option CASHOST is set, macro variable _CASHOST_ is set to the value that was specified for system option CASHOST. However, setting macro variable _CASHOST_ does not affect the CASHOST system option value.

When a connection is established, macro variable _CASHOST_ and system option CASHOST are updated to specify the current primary controller host name and, if available, the backup controller host name.

If the primary controller fails, the backup controller becomes the primary controller. Afterward, when a connection is established, macro variable _CASHOST_ and system option CASHOST are updated to specify the backup controller host name only. When the primary server is subsequently restored and a new session connection is established, the CASHOST system option and macro variable _CASHOST_ are updated with the current primary and secondary controller host names.

Tips: To display information about the controllers, run the following code:

```sas
proc cas;
    builtins.listNodes result = r;
    print r.nodelist.where(role eq "controller");
run;
quit;
```
If you want to delete macro variable _CASHOST_ and allow SAS system option CASHOST to prevail, use the following statement:

```sas
%symdel _CASHOST_
```

## Syntax

```sas
%let _CASHOST_ = primary-controller-host-name;
%let _CASHOST_ = (primary-controller-host-name, backup-controller-host-name);
```

### Syntax Description

- **primary-controller-host-name**
  - Specifies the primary controller host name.
  - *Note:* The name can be specified with or without quotation marks.

- **backup-controller-host-name**
  - Specifies the backup controller host name. The name can be specified with or without quotation marks.
  - **Requirements:** The primary controller host name and the backup host name must be enclosed in parentheses and must be separated by a comma or a space.
    - If you enclose the host names in quotation marks, each name must be enclosed in its own pair of quotation marks. Do not enclose the entire list in one set of quotation marks.
  - *Note:* The host name can be specified with or without quotation marks.

### Examples:

**Example 1**

Specify the primary controller host name only:

```sas
%let _CASHOST_ = cloud.example.com;
```

**Example 2**

Specify the primary controller host name and the backup controller host name, and enclose the names in quotation marks:

```sas
%let _CASHOST_ = ("cloud.example.com", "cloudbackup.example.com");
```

---

### _CASHOSTCONT_ Macro Variable

Stores the host name of the primary controller on the CAS server.

- **Default:** The primary host name specified in system option CASHOST, if set. Otherwise, not defined.
- **Range:** 1–256 characters
Notes: Macro variable _CASHOSTCONT_ stores the host name of the primary controller only. When system option CASHOST is set, macro variable _CASHOSTCONT_ is set to the primary controller host name that was specified in system option CASHOST. Macro variable _CASHOST_ is also set to the specified primary controller host name and the backup controller host name, if specified. If the primary controller fails, the backup controller becomes the primary controller. Afterward, when a new connection is established, macro variable _CASHOSTCONT_ is updated to specify the backup controller host name as the current primary controller. Macro variable _CASHOST_ and system option CASHOST are also updated.

Tip: To display information about the controllers, run the following code:

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

See: “_CASHOST_ Macro Variable” on page 383
“CASHOST= System Option” on page 469

Example: Display the current primary controller host name:

```sas
%put NOTE: The Primary Controller is: &cashostcont_;
```

---

**_CASPORT_ Macro Variable**

Specifies the CAS server port.

Default: The value of system option CASPORT, if set. Otherwise, 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.

Notes: When set to 0, CAS selects a port number. When system option CASPORT is set, macro variable _CASPORT_ is set to the value specified for system option CASPORT. However, setting macro variable _CASPORT_ does not affect the CASPORT system option value.

Tip: If you want to delete macro variable _CASPORT_ and allow SAS system option CASPORT to prevail, use the following statement:

```sas
%symdel _CASPORT_;
```

Syntax

```sas
%let _CASPORT_ = port;
```

---

**_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.
# Session Options

## Setting Session Options
- About the Session Options ........................................ 388
- Setting Session Options for a New Session ................. 388
- Setting Options for Existing Sessions ................... 388

## Session Options by Category

### Dictionary
- APPTAG= Session Option .................................................. 391
- AZURETENANTID Session Option .............................. 391
- CASLIB= Session Option .............................................. 392
- CMPLIB= Session Option .............................................. 392
- CMPOPT= Session Option .............................................. 393
- COLLATE= Session Option ............................................. 395
- DATASTEPFMTERR= Session Option ......................... 396
- DATASTEPMSGSUMLEVEL= Session Option ............... 397
- DATASTEPREPLACEABLE= Session Option .............. 397
- DEBUGOPT= Session Option ........................................... 398
- DLOCALE= Session Option ........................................... 399
- DSETPLOC= Session Option .......................................... 399
- EVENTDS= Session Option ............................................ 400
- FMTCASLIB= Session Option ......................................... 400
- INTERVALDS= Session Option ....................................... 401
- LOCALE= Session Option .............................................. 401
- LOGFLUSHTIME= Session Option ............................. 402
- MAXTABLEMEM= Session Option ............................. 402
- MESSAGELEVEL= Session Option ............................. 403
- METRICS= Session Option ............................................ 403
- NWORKERS= Session Option .......................................... 405
- TIMEOUT= Session Option ............................................. 405
- TIMEZONE= Session Option ........................................... 406
- USEGPU Session Option ................................................. 407
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 421. To see the setting for a specific property, use the GETSESSOPT function. See “GETSESSOPT Function” on page 379.

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.

- command-line options that are used in the server start-up command.
- SAS Cloud Analytic Services system defaults.

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.

<table>
<thead>
<tr>
<th>Task</th>
<th>Option to use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Override one or more session options for a new session only.</td>
<td>CAS statement SESSOPTS= option</td>
</tr>
</tbody>
</table>
Task | Option to use:  
---|---  
Override the TIMEOUT option value for all subsequently created sessions. | SAS system option \texttt{CASTIMEOUT=1, 3}  
Override the NWORKERS option value for all subsequently created sessions. | SAS system option \texttt{CASNWORKERS=2, 3}  

1. TIMEOUT in the CAS statement \texttt{SESSOPTS=} option overrides this option.  
2. NWORKERS in the CAS statement \texttt{SESSOPTS=} option overrides this option.  
3. 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 \texttt{SESSOPTS} option  
Change one or more session options for the active session. \(^1\) | SAS system option \texttt{CASSESSOPTS=} (alias \texttt{SESSOPTS=}  
Specify a caslib for the active session. \(^1\) | SAS system option \texttt{CASLIB=}  

1. SAS system option \texttt{SESSREF} stores the name of the currently active session.

### Session Options by Category

| Category | Language Elements | Description  
---|---|---  
Action | \texttt{APPTAG=} Session Option (p. 391) | specifies the string to prefix to log messages.  
Authentication | \texttt{AZURETENANTID} Session Option (p. 391) | specifies the tenant ID for connecting to a Microsoft Azure storage system in a session.  
Caslib | \texttt{CASLIB=} Session Option (p. 392) | specifies the caslib name to set as the active caslib.  
| \texttt{MAXTABLEMEM=} Session Option (p. 402) | specifies the maximum amount of memory, in bytes, to allocate for a table.
<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Data Processing</strong></td>
<td>EVENTDS= Session Option (p. 400)</td>
<td>specifies one or more data sets that define events.</td>
</tr>
<tr>
<td></td>
<td>INTERVALDS= Session Option (p. 401)</td>
<td>specifies one or more interval-name=value pairs, where the value is the name of a data set that contains user-defined intervals.</td>
</tr>
<tr>
<td><strong>Data Quality</strong></td>
<td>DQLOCALE= Session Option (p. 399)</td>
<td>specifies the default 5-letter SAS Quality Knowledge Base (QKB) ISO locale code to use as the default locale for data quality (DQ) operations.</td>
</tr>
<tr>
<td></td>
<td>DQSETUPLOC= Session Option (p. 399)</td>
<td>specifies the name of the default SAS Quality Knowledge Base (QKB) to use for data quality operations.</td>
</tr>
<tr>
<td><strong>DATA step</strong></td>
<td>DATASTEPFMTERR= Session Option (p. 396)</td>
<td>specifies how SAS reacts when a specified variable format cannot be found.</td>
</tr>
<tr>
<td></td>
<td>DATASTEPMSGSUMLEVEL= Session Option (p. 397)</td>
<td>specifies the DATA step message summary level.</td>
</tr>
<tr>
<td></td>
<td>DATASTEPREPLACETABLE= Session Option (p. 397)</td>
<td>specifies whether a DATA step can replace an existing table.</td>
</tr>
<tr>
<td><strong>Debugging</strong></td>
<td>DEBUGOPT= Session Option (p. 398)</td>
<td>specifies the host and port for the SAS Code Debugger in a session.</td>
</tr>
<tr>
<td><strong>Files: SAS Files</strong></td>
<td>CMPLIB= Session Option (p. 392)</td>
<td>specifies one or more CMP libraries that contain compiler subroutines to include during program compilation.</td>
</tr>
<tr>
<td><strong>Formats</strong></td>
<td>FMTCASLIB= Session Option (p. 400)</td>
<td>specifies the caslib where persisted format libraries are retained.</td>
</tr>
<tr>
<td><strong>Input Control</strong></td>
<td>EVENTDS= Session Option (p. 400)</td>
<td>specifies one or more data sets that define events.</td>
</tr>
<tr>
<td></td>
<td>INTERVALDS= Session Option (p. 401)</td>
<td>specifies one or more interval-name=value pairs, where the value is the name of a data set that contains user-defined intervals.</td>
</tr>
<tr>
<td><strong>Localization</strong></td>
<td>LOCALE= Session Option (p. 401)</td>
<td>specifies the locale to use for sorting and formatting.</td>
</tr>
<tr>
<td><strong>Log</strong></td>
<td>LOGFLUSHTIME= Session Option (p. 402)</td>
<td>specifies the log flush time, in milliseconds.</td>
</tr>
<tr>
<td></td>
<td>MESSAGELEVEL= Session Option (p. 403)</td>
<td>specifies the log message level.</td>
</tr>
<tr>
<td></td>
<td>METRICS= Session Option (p. 403)</td>
<td>specifies whether to include default detailed performance metrics reports in the SAS log.</td>
</tr>
<tr>
<td><strong>Session</strong></td>
<td>NWORKERS= Session Option (p. 405)</td>
<td>specifies the number of worker nodes for a new session.</td>
</tr>
</tbody>
</table>
### Category | Language Elements | Description
---|---|---
TIMEOUT= Session Option (p. 405) | specifies the SAS Cloud Analytic Services session time-out in seconds for a new or existing session.

TIMEZONE= Session Option (p. 406) | specifies the user local time zone.

USEGPU Session Option (p. 407) | specifies whether work is offloaded to the Graphics Processing Unit (GPU), if it is available.

Sort | COLLATE= Session Option (p. 395) | specifies the collating sequence for sorting.

System Administration: Performance | CMPOPT= Session Option (p. 393) | 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, CAS configuration file, OPTIONS statement CASSESSOPTS option, GETSESSOPT function
- **Category:** Action
- **Default:** No prefix
- **See:** SAS Cloud Analytic Services: Reference in SAS Viya Administration: SAS Cloud Analytic Services for information about the CAS configuration file.

**Syntax**

`APPTAG="tag-string"`

#### AZURETENANTID Session Option

specifies the tenant ID for connecting to a Microsoft Azure storage system in a session.

- **Valid in:** CAS statement SESSOPTS option, CAS configuration file, OPTIONS statement CASSESSOPTS option, GETSESSOPT function
- **Category:** Authentication
- **Note:** This session option might have already been set by your CAS administrator.
CASLIB= Session Option

classifies 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 447

Syntax

CASLIB="caslib-name"

CMPLIB= Session Option

specifies one or more CMP libraries that contain compiler subroutines to include during program compilation.

Valid in: CAS statement SESSOPTS option, OPTIONS statement CASSESSOPTS option, GETSESSOPT function

Category: Files: SAS Files

Syntax

CMPLIB="caslib-name.cas-table-name <caslib-name.cas-table-name <...>>"
Parameter Value

caslib-name.cas-table-name <caslib-name.cas-table-name <...>>
specifies the name of one or more CMP libraries that contain compiler functions and subroutines. In SAS Cloud Analytic Services, CMP function and subroutine libraries are stored in tables. The CMP libraries are searched in the order in which they are specified in this option. In the case of duplicate functions or subroutines, the last instance that is found in the CMP library search is used.

Requirements Each CMP library name must be a two-level name that specifies the name of the caslib in which the CMP library exists and the name of the CMP library table as follows: caslib-name.cas-table-name.

Note When specifying multiple libraries, separate each name with a space.

Examples Specify a single CMP library for session Casauto:
cas casauto sessopts=(cmplib="casuser.mycmpfuncs");

Specify two CMP libraries for the active SAS Cloud Analytic Services session:
options cassessopts=(cmplib="ourcmplibs.cmpfuncs casuser.mycmpfuncs");

In this case, the CMP library search order is ourcmplibs.cmpfuncs, and then casuser.mycmpfuncs. If functions and subroutines are duplicated in both libraries, those in casuser.mycmpfuncs take precedence.

Display the current CMP libraries for session Casauto:
%put Session CASAUTO CMPLIB=%sysfunc(getsessopt(casauto, cmplib));

See Also
- “FCMP Action Set” in SAS Viya: System Programming Guide

CMPOPT= Session Option

specifies the type of code generation optimizations to use in the SAS language compiler.

Valid in: CAS statement SESSOPTS option, CAS configuration file, 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.

- $x * 1$  
- $x * -1$
- $x + 1$  
- $x + -1$
- $x + 0$  
- $x$
- $x - x$  
- $x + x$
- $-x$  
- 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, CAS configuration file, 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, CAS configuration file, OPTIONS statement CASSESSOPTS option, GETSESSOPT function

Category: DATA step
Alias: FMTERR
Default: TRUE

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.
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, CAS configuration file, 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.

### DATASTEPREPLACE= Session Option

specifies whether a DATA step can replace an existing table.
Valid in: CAS statement `SESSOPTS` option, CAS configuration file, OPTIONS statement `CASSESSOPTS` option, `GETSESSOPT` function

Category: DATA step

Default: `TRUE`


Syntax

```
DATASTEPREPLACEABLE=TRUE | FALSE
```

### DEBUGOPT= Session Option

specifies the host and port for the SAS Code Debugger in a session.

Valid in: CAS statement `SESSOPTS` option, CAS configuration file, OPTIONS statement `CASSESSOPTS` option, GETSESSOPT function

Category: Debugging

Notes: This option is not initialized with the value of SAS system option DEBUGOPT. You must set session option DEBUGOPT separately.

This session option might have already been set by your CAS administrator.

Default: ""


Syntax

```
DEBUGOPT="DEBUGHOST=host-name DEBUGPORT=port"
```

### Debug Options

**DEBUGHOST=** `host-name`

specifies the debugger host by name.

**Requirement** Specify `DEBUGHOST=` and `DEBUGPORT=` separated by a space and enclosed in single or double quotation marks.

**DEBUGPORT=** `port`

specifies the debugger port.

Examples:

Example 1

Specify the debugger host name and port:

```
cas casauto sessopts=(debugopt="debughost=cloud.example.com debugport=5590");
```
**Example 2**
Set session option DEBUGOPT in session Casauto to the current value of SAS system option DEBUGOPT:

```
cas casauto sessopts=(debugopt="%sysfunc(getoption(debugopt))");
```

**Example 3**
List the current DEBUGOPT= setting for session Casauto:

```
%put DEBUGOPT=%sysfunc(getsessopt(casauto,debugopt));
```

**Example 4**
Clear the debug options to revert to the default:

```
cas casauto sessopts=(debugopt="");
```

**See Also**
- “DEBUGOPT= System Option” in *SAS System Options: Reference*

---

**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, CAS configuration file, OPTIONS statement
- CASSESSOPTS option, GETSESSOPT function

**Category:**
Data Quality

**See:**

**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, CAS configuration file, 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, CAS configuration file, 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, CAS configuration file, 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, CAS configuration file, OPTIONS statement CASSESSOPTS option, GETSESSOPT function

Category: Localization

Default: en_US

Syntax

**LOCALE="locale"**

See Also


---

**LOGFLUSHTIME= Session Option**

specifies the log flush time, in milliseconds.

Valid in: CAS statement **SESSOPTS** option, CAS configuration file, 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, CAS configuration file, 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.

Tips: 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.
If you are creating a large table of 5G or greater per node, to improve performance, set MAXTABLEMEM in your active session to 256M or higher.


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, CAS configuration file, 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, CAS configuration file, 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 the SAS log when metrics are enabled.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Note Written to the SAS Log</th>
</tr>
</thead>
<tbody>
<tr>
<td>The number of bytes moved.</td>
<td>NOTE: bytes moved number&lt;units&gt;</td>
</tr>
<tr>
<td>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%.</td>
<td>NOTE: cpu time number seconds (number%)</td>
</tr>
<tr>
<td>The data movement time in seconds.</td>
<td>NOTE: data movement time number seconds</td>
</tr>
<tr>
<td>The amount of memory used in bytes and as a percentage of the total available memory.</td>
<td>NOTE: memory number&lt;units&gt; (number%)</td>
</tr>
<tr>
<td>The amount of time it took to run the action start to finish (real time) in seconds.</td>
<td>NOTE: real time number seconds</td>
</tr>
<tr>
<td>The total available memory.</td>
<td>NOTE: total memory number&lt;units&gt;</td>
</tr>
<tr>
<td>The total number of nodes and cores in the cluster.</td>
<td>NOTE: total nodes number (number cores)</td>
</tr>
</tbody>
</table>
Here is an example of the default metrics that are written to the SAS log when the MDSUMMARY procedure is executed.

<table>
<thead>
<tr>
<th>NOTE: real time</th>
<th>0.126269 seconds</th>
</tr>
</thead>
<tbody>
<tr>
<td>NOTE: cpu time</td>
<td>1.036837 seconds (821.13%)</td>
</tr>
<tr>
<td>NOTE: data movement time</td>
<td>0.013819 seconds</td>
</tr>
<tr>
<td>NOTE: total nodes</td>
<td>27 (1296 cores)</td>
</tr>
<tr>
<td>NOTE: total memory</td>
<td>6.65T</td>
</tr>
<tr>
<td>NOTE: memory</td>
<td>57.25M (0.00%)</td>
</tr>
<tr>
<td>NOTE: bytes moved</td>
<td>4.80K</td>
</tr>
</tbody>
</table>

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, CAS configuration file, 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 474


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, CAS configuration file, 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
TIMEOUT= Session Option

specifies the user local time zone.

Valid in: CAS statement SESSOPTS option, OPTIONS statement CASSESSOPTS option, GETSESSOPT function

Category: Session

Syntax

```plaintext
TIMEOUT=number
```

TIMEZONE= Session Option

specifies the user local time zone.

Valid in: CAS statement SESSOPTS option, OPTIONS statement CASSESSOPTS option, GETSESSOPT function

Syntax

```plaintext
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 or 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.
USEGPU Session Option

specifies whether work is offloaded to the Graphics Processing Unit (GPU), if it is available.

Valid in: CAS statement SESSOPTS option, OPTIONS statement CASSESSOPTS option, GETSESSOPT function

Category: Session

Syntax

USEGPU=ALWAYS | NEVER | OPTIMAL

Parameter Values

ALWAYS
specifies that work is always offloaded to the GPU, if it is available.

NEVER
specifies that work is never offloaded to the GPU.

OPTIMAL
specifies that the system determines when work is offloaded to the GPU, if it is available.

See Also
“TIMEZONE= System Option” in SAS System Options: Reference
Dictionary

CAS Statement
Starts and manages your SAS Cloud Analytic Services session.

Syntax
Form 1:  \texttt{CAS session-name <options>;}\hfill  409
Form 2:  \texttt{CAS \_ALL\_ LIST \_ ALL\_ TERMINATE}\hfill  409

Summary of Optional Arguments

Authentication option
\begin{itemize}
\item \texttt{AUTHDOMAIN=authentication-domain \_SASMETA\_}
  obtains credentials from the SAS Metadata Server or the SAS Viya credentials service.
\item \texttt{AUTHINFO=authentication-info-file}
  specifies an authinfo file or netrc file that includes authentication information.
\end{itemize}

CAS server information options
\begin{itemize}
\item \texttt{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.
\item \texttt{LISTABOUT}
  writes information about SAS Cloud Analytic Services to the SAS log.
\item \texttt{LISTSERVERSTARTOPTS}
  lists the SAS Cloud Analytic Services options and their current values.
\end{itemize}
Connection options

**DISCONNECT**

disconnects SAS from the session.

**HOST=** "primary-host-name" | ("primary-host-name", "backup-host-name")

specifies the primary controller host name and the backup controller host name (optional).

**PORT=** *number*

specifies the port on which the controller listens for client connections.

**RECONNECT**

reconnects to a session using a session name.

**TERMINATE**

terminates a single connected session or all of the connected sessions in your SAS client.

**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 single session or all of the sessions 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-options**)**

specifies one or more session option settings to apply during or after session start-up.

User-defined format options

**ADDFMTLIB** TABLE=**table-name** <FMTLIBNAME=**format-library-name**><CASLIB=**caslib**><POSITION=**APPEND** | **INSERT** | **REPLACE** | **NONE**><REPLACEFMTLIB>

**ADDFMTLIB** PATH=**path** <FMTLIBNAME=**format-library-name**><POSITION=**APPEND** | **INSERT** | **REPLACE** | **NONE**><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.

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

**Wild-card option**

_**ALL**_
specifies that the LIST or TERMINATE argument that follows applies to all of the sessions in the SAS client.

**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**. Also, macro variable **_CASHOST_** and SAS system option **CASHOST** are updated to specify the current primary controller host name on the server and, if available, the backup controller host 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 426.

**Optional Arguments**

_**ALL**_
specifies that the LIST or TERMINATE argument that follows applies to all of the sessions in the SAS client.

**Requirement**
This option must be followed by LIST or TERMINATE.

**See**
"LIST" on page 418
"TERMINATE" on page 423
ADDFMTLIB TABLE=`table-name` <FMTLIBNAME=`format-library-name`> <CASLIB=`caslib`> <POSITION=APPEND | INSERT | REPLACE | NONE> <REPLACEFMTLIB>
ADDFMTLIB PATH=`path` <FMTLIBNAME=`format-library-name`> <POSITION=APPEND | INSERT | REPLACE | NONE> <REPLACEFMTLIB>

adds a session format library. By default, the format library is appended to the format library search list.

<table>
<thead>
<tr>
<th>TABLE=<code>table-name</code></th>
<th>specifies the name of the table where the format library was previously saved using SAVEFMTLIB.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>the name specified in the FMTLIBNAME= option</td>
</tr>
<tr>
<td>Notes</td>
<td>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).</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PATH=<code>path</code></th>
<th>specifies the absolute path to the file in which the format information is stored.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Requirement</td>
<td>The specified path must be authorized and accessible from the controller.</td>
</tr>
<tr>
<td>Notes</td>
<td>Do not specify this option with TABLE= or CASLIB=.</td>
</tr>
<tr>
<td></td>
<td>The path can be outside of a caslib.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>FMTLIBNAME=<code>format-library-name</code></th>
<th>specifies the name of the format library to add.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td><code>_FMTLIBn</code>, where <code>n</code> starts at 1 and is incremented for each format library that is added.</td>
</tr>
<tr>
<td>Restriction</td>
<td>The format library name cannot exceed 63 characters in length.</td>
</tr>
<tr>
<td>Note</td>
<td>If the format library already exists in the session, specify REPLACEFMTLIB to replace the existing format library. Otherwise, an error occurs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CASLIB=<code>caslib</code></th>
<th>specifies the name of the caslib where the table is saved.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>the active caslib for the session</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>POSITION=APPEND</th>
<th>INSERT</th>
<th>REPLACE</th>
<th>NONE</th>
<th>specifies the position of this format library in the format-library search list.</th>
</tr>
</thead>
<tbody>
<tr>
<td>APPEND</td>
<td>appends this format library to the end of the format-library search list.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>INSERT</td>
<td>inserts this format library at the beginning of the format-library search list.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

REPLACFMTLIB  replaces the format library if it already exists.

Tip  When you add a user-defined format library, it persists until it is deleted or the session is terminated. Your SAS Cloud Analytic Services system administrator can configure CAS to add user-defined format libraries to all sessions and promote them automatically at session start-up. See Managing User-Defined Formats in SAS Viya in SAS Viya Administration: Data.

See  SAS Cloud Analytic Services: User-Defined Formats

Examples  “Example 11: Add a Format Library That Is Stored in a SASHDAT File” on page 442


AUTHDOMAIN=authentication-domain | _SASMETA_  obtains credentials from the SAS Metadata Server or the SAS Viya credentials service. SAS system option SERVICESBASEURL specifies the root URL for SAS 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. The domain name can be specified with or without quotation marks.

_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_.

_AUTHDOMAIN_
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 SAS Viya credentials service:
- System option SERVICESBASEURL must specify the base URL for the SAS Viya services.
- Environment variable SAS_VIYA_TOKEN must specify a valid CAS OAuth access token for accessing SAS 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 417 and PORT= on page 421 options, or they must be obtained from the SAS Metadata Server by specifying the CASSERVERMD= on page 415 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:

```bash
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:

```bash
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 or file %HOMEDRIVE%\%HOMEPATH\_%authinfo on Windows 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.

Notes The session name is preserved for use with the RECONNECT option for the duration of the SAS session. See “Example 8: Connect to an Existing Session” on page 438.

When you disconnect from the default session, SAS system option SESSREF= continues to reference the disconnected session. To access the server in that case, you must reconnect to the default session, start a new session, or set system option SESSREF= to reference an existing connected session. See “Example 1: Start a Session” on page 432 and “SESSREF= System Option” on page 479.

Tip The session time-out value determines the lifetime in seconds of a disconnected session. The default is 60 seconds. 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. Before you disconnect, if necessary, use the SESSOPTS= on page 423 option to increase the time-out value.

See “Example 7: Disconnect from a Session” on page 437

DROPFMTLIB FMTLIBNAME=format-library-name <FMTSEARCHREMOVE> drops a session-scope or a global-scope format library.
FMTLIBNAME=\texttt{format-library-name}

specifies the name of the format library to drop.

This argument is required.

FMTSEARCHREMOVE

removes the format library from the format search list.

\textbf{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.

\textbf{See}  
\textit{SAS Cloud Analytic Services: User-Defined Formats}

\textbf{Example}  
"Example 18: Drop a Format Library from Session Scope" on page 447

FMTSEARCH=(\texttt{name1} < \texttt{name2} ... \texttt{nameN}) <\texttt{POSITION=APPEND} | \texttt{INSERT} | \texttt{REPLACE}>

modifies the format library search list for the session.

\texttt{(name1} < \texttt{name2} ... \texttt{nameN}>)

specifies a list of one or more format-library names enclosed in parentheses. Each name is separated by a space.

\texttt{POSITION=APPEND} | \texttt{INSERT} | \texttt{REPLACE}

specifies the position of the format libraries in the format-library search list.

\texttt{APPEND}

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

\texttt{INSERT}

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

\texttt{REPLACE}

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

\textbf{Default}  
\texttt{APPEND}

\textbf{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.

\textbf{See}  
\textit{SAS Cloud Analytic Services: User-Defined Formats}

\textbf{Example}  
Add format libraries FMTLIB1 and FMTLIB2 to the beginning of the format library search list for session Casauto:
\begin{verbatim}
cas casauto fmtsearch=(fmtlib1 fmtlib2) position=insert;
\end{verbatim}

\textbf{Examples}  
"Example 14: Add Global Format Libraries to a Session's Format Search List" on page 444

"Example 15: Change the Format Library Search List" on page 445

FMTSEARCH CLEAR

clears the format library search list for the session.

\textbf{See}  
\textit{SAS Cloud Analytic Services: User-Defined Formats}
Clear the format search list for session Casauto:

```
cas casauto fmtsearch clear;
```

**HOST=** 

```
"primary-host-name" | ("primary-host-name", "backup-host-name")
```

specifies the primary controller host name and the backup controller host name (optional). If the primary controller on the server fails, the backup controller becomes the primary controller. When a new connection is established using the CAS statement, a connect message in the SAS log reflects the host name of the current primary controller. System option CASHOST= and macro variable _CASHOST_ are also updated with the current primary controller host name and, if available, the backup controller host name.

**Alias**

CASHOST

**Default**

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

**Restriction**

This option is valid only when starting a new session.

**Requirement**

Each host name must be enclosed in its own pair of quotation marks.

**Interactions**

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

Macro variable _CASHOST_ and SAS system option CASHOST are updated to specify the current primary controller host name and, if available, the backup controller host name.

**Notes**

The primary controller host name and the backup controller 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 and an attempt is made to start a new session, an error occurs. The session is not started, even when a valid backup host name is specified.

**Tip**

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

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

**See**

- “_CASHOST_ Macro Variable” on page 383
- “_CASHOSTCONT_ Macro Variable” on page 384
- "CASHOST= System Option" on page 469

**Examples**

Start a session on primary controller `cloud.example.com` using port 5570:

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

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

```
cas casauto host=("cloud.example.com", "cloudbackup.example.com")
```
LIST writes to the SAS log information about a single session or all of the sessions 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 this option with `session-name` to list information about a single session, or use it with `_ALL_` on page 411 to list information about all of the sessions that are known to 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 434

---

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

<table>
<thead>
<tr>
<th>Section</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>About</td>
<td>CAS</td>
<td>Product name: SAS Cloud Analytic Services.</td>
</tr>
<tr>
<td></td>
<td>Version</td>
<td>Short-form SAS Cloud Analytic Services version.</td>
</tr>
<tr>
<td></td>
<td>Copyright</td>
<td>Copyright information.</td>
</tr>
<tr>
<td>System</td>
<td>Host name</td>
<td>SAS Cloud Analytic Services primary controller host name.</td>
</tr>
<tr>
<td></td>
<td>OS Name</td>
<td>SAS Cloud Analytic Services host information.</td>
</tr>
<tr>
<td></td>
<td>OS Family</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OS Release</td>
<td></td>
</tr>
<tr>
<td></td>
<td>OS Version</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Model Number</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Documentation</td>
<td>SAS Cloud Analytic Services documentation URL.</td>
</tr>
</tbody>
</table>

---
## License

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>siteNum</td>
<td>Site information.</td>
</tr>
<tr>
<td>expires</td>
<td>License expiration date and time.</td>
</tr>
<tr>
<td>gracePeriod</td>
<td>Grace period in days.</td>
</tr>
<tr>
<td>warningPeriod</td>
<td>Expiration warning period in days.</td>
</tr>
</tbody>
</table>

### 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 445

### 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 444

“Example 15: Change the Format Library Search List” on page 445

### 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.
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 443

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 “CAS Language (CASL) and 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_;
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 435

**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**

“Session Options” on page 387 for a list of the session options.

“Example 4: List the Properties for a Session” on page 436

**Example**

List the session option settings for session Casauto:

cas casauto LISTSESSOPTS;

**PORT=number**

specifies the port on which the controller 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 385

“CASPORT= System Option” on page 475

**PROMOTEFMTLIBLE 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.

**Requirement**

`Format-library-name` must be in the session format search list. Use the LISTFMTSEARCH option to see the current format search list. Use the FMTSEARCH on page 416 option to update the search list, if needed.
Tips

When you promote a user-defined format, it persists as a global format library until it is deleted or the CAS server is restarted. Your SAS Cloud Analytic Services system administrator can configure CAS to add user-defined format libraries to all sessions and promote them automatically at session start-up. See Managing User-Defined Formats in SAS Viya in SAS Viya Administration: Data.

Global format libraries are not automatically included in the format search list for new sessions. Use FMTSEARCH on page 416 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 446

RECONNECT

reconnects to a session using a session name.

Note

If the target session has terminated, an error message is written to the SAS log, and the session is automatically deleted from the SAS client. In that case, to connect to the server, you must create a new session.

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 424.

See

“Example 8: Connect to an Existing Session” on page 438

SAVEFMTLIB FMTLIBNAME=format-library-name <TABLE=table-name> <CASLIB=caslib> <REPLACE> <PROMOTE>

SAVEFMTLIB FMTLIBNAME=format-library-name <PATH=path>

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 file name, the format library that it contains cannot be loaded.
CASLIB=\texttt{caslib}

specifies the caslib in which the table is stored.

Default \hspace{1cm} the active caslib for the session

Requirement \hspace{1cm} You must have Write access to the specified caslib.

Note \hspace{1cm} Do not specify this option with PATH=.

Tip \hspace{1cm} 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 \hspace{1cm} REPLACE

Note \hspace{1cm} This option is ignored when PATH= is specified.

PATH=\texttt{path}

specifies the absolute path to the file in which the format library is to be saved.

Requirement \hspace{1cm} The specified path must be authorized and accessible from the controller.

Notes \hspace{1cm} Do not specify this option with TABLE= or CASLIB=.

The path can be outside of a caslib.

PROMOTE

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

Note \hspace{1cm} This option is ignored when PATH= is specified.

See \hspace{1cm} SAS Cloud Analytic Services: User-Defined Formats

Example \hspace{1cm} “Example 10: Create a Format Library and Save It to a SASHDAT File” on page 439

SESSOPTS=(session-options)

specifies one or more session option settings to apply during or after session start-up.

See \hspace{1cm} “Session Options” on page 387 for a list of the options that you can specify for session-options.

“Example 6: Change a Session Property Value” on page 437

“Program: Start a Session with Custom Properties” on page 433

TERMINATE

terminates a single connected session or all of the connected sessions in your SAS client.

Alias \hspace{1cm} 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 432 and “SESSREF= System Option” on page 479.

Tip: Use this option with session-name to terminate a single connected session, or use it with _ALL_ to terminate all of the currently connected sessions in your SAS client.

See: “Example 9: Terminate a Session” on page 439

**USER=**user-ID

Specifies the user ID to use for connecting to SAS Cloud Analytic Services.

<table>
<thead>
<tr>
<th>Alias</th>
<th>CASUSER</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>SAS system option CASUSER=, if set.</td>
</tr>
<tr>
<td>Restriction</td>
<td>This option is valid only when starting a new session.</td>
</tr>
<tr>
<td>Requirement</td>
<td>User-ID must match a user ID in your authinfo file.</td>
</tr>
<tr>
<td>Interaction</td>
<td>This option overrides SAS system option CASUSER=.</td>
</tr>
</tbody>
</table>

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 478

SAS Viya Administration: Authentication

**UUID=**"session-uuid"

Specifies the UUID of an existing session to which you want to connect.

| Restriction | Only one named session can be associated with a given session UUID. |
| 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.

Note: If you attempt to connect to a session that is busy, an error message is written to the SAS log, and the connection is not established. The error message indicates that the request cannot be submitted because the target session is busy running actions.

Tips: You can view the UUID for a session with the LISTSESSIONS option.
If the target session is busy, use the `actionstatus` action in the Session action set to determine when the target session is free.

See

“LIST” on page 418

“Program: Connect to an Existing Session Using the Session Name and UUID” on page 438

<table>
<thead>
<tr>
<th>UUIDMAC=macro-variable-name</th>
</tr>
</thead>
<tbody>
<tr>
<td>specifies a SAS macro variable name into which the UUID of the session is stored.</td>
</tr>
</tbody>
</table>

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 434

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

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. If you are using SAS Studio, a session might have been started for you when you signed on. To see whether a session has already been started, run the following statement:

```
cas _all_ list;
```

If a session has been started, you should see a message similar to the following in the SAS log.

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

The note in the SAS log indicates the host name and port of the controller to which the session is connected and the session UUID. If a backup controller is available, the host name is the name of the primary controller. If the primary controller is currently not available, the host name is the name of the backup controller.
If a session has not been started, no messages are written to the SAS log. In that case, use a CAS statement to start a session. For examples, see “Example 1: Start a Session” on page 432. After your session has been started, you can use the CAS statement to perform management tasks on your session. Some examples are 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 437.

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 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 12.1  Performing Common Tasks for User-Defined Formats Using the CAS Statement

<table>
<thead>
<tr>
<th>Task</th>
<th>What to Use</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Migrate your user-defined formats stored in SAS format catalogs to CAS.</td>
<td>The IFTC2ITM procedure and the CAS statement LOADFORMATS option</td>
<td>“Migrate User-Defined Formats from SAS To CAS” in SAS Cloud Analytic Services: User-Defined Formats</td>
</tr>
<tr>
<td>Add a user-defined format library stored in a SASHDAT file to your session.</td>
<td>The CAS statement ADDFMTLIB on page 412 option</td>
<td>“Example 11: Add a Format Library That Is Stored in a SASHDAT File” (p. 442)</td>
</tr>
<tr>
<td>Promote a session-scope user-defined format library in your session to global scope.</td>
<td>The CAS statement PROMOTEFMTLIB on page 421 option</td>
<td>“Example 17: Promote a Format Library to Global Scope” (p. 446)</td>
</tr>
<tr>
<td>Task</td>
<td>What to Use</td>
<td>Example(s)</td>
</tr>
<tr>
<td>----------------------------------------------------------------------</td>
<td>-----------------------------------------------</td>
<td>---------------------------------------------------------------------------</td>
</tr>
<tr>
<td>List the format libraries and formats that are available in your session.</td>
<td>The CAS statement LISTFORMATS on page 419 option</td>
<td>“Example 13: List the Session-Scope Formats in a Session” (p. 443)</td>
</tr>
<tr>
<td>List the ranges in a user-defined format in your session.</td>
<td>The CAS statement LISTFMTRANGES on page 419 option</td>
<td>“Example 16: List the Ranges in a Format” (p. 445)</td>
</tr>
</tbody>
</table>
| Display the user-defined format library search list for your session.  | The CAS statement LISTFMTSEARCH on page 419 option | “Example 14: Add Global Format Libraries to a Session’s Format Search List” (p. 444)  
“Example 15: Change the Format Library Search List” (p. 445)                  |
| Modify your session’s user-defined format library search list.         | The CAS statement FMTSEARCH on page 416 option | “Example 14: Add Global Format Libraries to a Session’s Format Search List” (p. 444)  
“Example 15: Change the Format Library Search List” (p. 445)                  |
| Save a format library in your session to a SASHDAT file.               | The CAS statement SAVEFMTLIB on page 422 option | “Example 10: Create a Format Library and Save It to a SASHDAT File” (p. 439) |
| Drop a session-scope or global-scope user-defined format library.      | The CAS statement DROPFMTLIB on page 415 option | “Example 18: Drop a Format Library from Session Scope” (p. 447)             |

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
- The Session is Busy Running Actions on page 431
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.

<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 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 438.
- Verify that session `session-name` exists. See “Example 3: List Information about All of Your Sessions” on page 435. If session `session-name` does not exist, specify a different session or start session `session-name`. See “Example 1: Start a Session” on page 432. |

---

**Could Not Find Netrc or Authinfo File**

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

<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Your netrc or authinfo file that is required for authentication could not be found.</td>
<td>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.</td>
</tr>
</tbody>
</table>

---

**Missing Session Information**

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

**ERROR:** Missing session information.

---

**Note:** The warning message might not appear.
<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 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 438.  
- Verify that session session-name exists. See "Example 3: List Information about All of Your Sessions" on page 435. If session session-name does not exist, specify a different session or start session session-name. See "Example 1: Start a Session" on page 432. |

**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.  


**Note:** The authentication error message might not appear.  

<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 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 435. 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 432. |

**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.
<table>
<thead>
<tr>
<th><strong>Cause</strong></th>
<th><strong>Solution</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>The maximum number of concurrent sessions specified by the SAS Cloud Analytic Services MAXSESSIONS configuration parameter has been reached.</td>
<td>Contact your SAS Cloud Analytic Services system administrator. For more information, see “Maximum Number of Sessions on the Server” on page 426.</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th><strong>Cause</strong></th>
<th><strong>Solution</strong></th>
</tr>
</thead>
</table>
| 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 438.  
▪ Verify that session session-name exists. See “Example 3: List Information about All of Your Sessions” on page 435. If session session-name does not exist, specify a different session or start session session-name. See “Example 1: Start a Session” on page 432. |

### 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.
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.
```

<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>A request such as LISTSESSOPTS or LISTFORMATS was made on session <code>session-name</code>, but session <code>session-name</code> is disconnected.</td>
<td>Reconnect to session <code>session-name</code>. See “Example 8: Connect to an Existing Session” on page 438.</td>
</tr>
</tbody>
</table>

The Session Is Busy Running Actions

The following error message appears:

```
ERROR: The request cannot be submitted to session session-name. The session is busy running actions that were submitted with the ASYNC= option. Number of actions: 1.
```
Table 12.2  Error Cause and Solution

<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
</table>
| A CAS statement was submitted for a session, but the session is currently busy running an action asynchronously. One or more additional error messages might appear. | Wait for the asynchronous action to complete, and then run your CAS statement again. For an existing session, use SAS system function **SESSBUSY** to determine when the session is free. Here is an example:  
```put Session Casauto is busy: %sysfunc(sessbusy(casauto));```  
In this example, function SESSBUSY returns 1 if session Casauto is currently busy or 0 if it is free. If you received this error while attempting to connect to another session using the session’s UUID, use the actionstatus action in the Session action set to determine when the target session is free. |

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.


<table>
<thead>
<tr>
<th>Cause</th>
<th>Solution</th>
</tr>
</thead>
</table>
| 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;```
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:

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

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. NOTE: 'CASUSER(sasdemo)' is now the active caslib. NOTE: Action 'setsessopt' used (Total process time):
```

```plain
<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Additional Info</th>
</tr>
</thead>
<tbody>
<tr>
<td>real time</td>
<td>0.181410 seconds</td>
<td></td>
</tr>
<tr>
<td>cpu time</td>
<td>0.446927 seconds</td>
<td>(246.36%)</td>
</tr>
<tr>
<td>total nodes</td>
<td>142 (4556 cores)</td>
<td></td>
</tr>
<tr>
<td>total memory</td>
<td>35.47T</td>
<td></td>
</tr>
<tr>
<td>memory</td>
<td>13.09M (0.00%)</td>
<td></td>
</tr>
</tbody>
</table>
```

NOTE: The CAS statement 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:

```sas
/* 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 447 and "CAS LIBNAME Statement" on page 37.

- 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 timeout period. See TIMEOUT= on page 405.

- 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:

```sas
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. In this example, the sessions are Casauto, Mysess1, and Mysess2:

```
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:

```sas
  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
  Description = specifies the string to prefix to log messages.

NOTE: Name = caslib
  UsageType = Session
  Type = String
  Value = CASUSERHDFS(sasdemo)
  Default Value =
  Group = Caslib
  Description = specifies the caslib name to set as the active caslib.
```

**Additional Information**

For information about the session properties, see “Session Options” on page 387. The GETSESSOPT function enables you to get the value of a single property. See “GETSESSOPT Function” on page 379.

Example 5: Display a Session Property Value

**Program**

Display the current TIMEOUT property value for session Casauto:

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

**SAS Log**

The following is written to the SAS log:

```
TIMEOUT for session CASAUTO: 60 seconds.
```
Example 6: Change a Session Property Value

Program

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

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

Additional Information

For information about the session properties that you can change, see “Session Options” on page 387. To change one or more properties for all of your generated SAS Cloud Analytic Services sessions, specify the property settings in the CASSESSOPTS= option in an OPTIONS statement. See "CASSESSOPTS= System Option" on page 475.

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 438.

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:

```sas
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.
```

Disconnecting from the Default Session

If you disconnect from the default session, you must do one of the following in order to access the server:

- Reconnect to the default session as shown in the following example:
Issue a CAS statement to create a new default session. When you create a new session, system option SESSREF= is automatically set to reference the new session. See "Example 1: Start a Session" on page 432.

If you have one or more additional sessions available on your client that are connected to the server, set system option SESSREF= to one of the other connected sessions as shown in the following example:

```plaintext
options sessref=session-name;
```

See "SESSREF= System Option" on page 479.

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:

```plaintext
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 435. This example uses session name Mysess and the UUID option to connect to session Mysess, which was started in a different SAS session.

```plaintext
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 ca63ddf-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: The default CAS session CASAUTO identified by SAS option SESSREF= was terminated. Use the OPTIONS statement to set the SESSREF= option to an active session.
NOTE: Request to TERMINATE completed for session CASAUTO.

Terminating the Default Session

If you terminate the default session, a note similar to the following is written to the SAS log.

NOTE: The default CAS session CASAUTO identified by SAS option SESSREF= was terminated. Use the OPTIONS statement to set the SESSREF= option to an active session.

In that case, to connect to the server, do one of the following:

- Issue a CAS statement to create a new default session. When you create a new session, system option SESSREF= is automatically set to reference the new session. See "Example 1: Start a Session" on page 432.

- If you have one or more additional sessions available on your client that are connected to the server, set system option SESSREF= to one of the other connected sessions as shown in the following example:

   options sessref=session-name;

   See "SESSREF= System Option" on page 479.

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. All of the formats use the default
locale. For an example of how to create locale-specific formats, see "Add Locale-Specific Format Libraries" in SAS Cloud Analytic Services: User-Defined Formats.

```sas
/* 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 “Migrate User-Defined Formats from SAS To CAS” in SAS Cloud Analytic Services: User-Defined Formats.

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.
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 file name. 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.

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

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         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         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          cas casauto savefmtlib fmtlibname=myFmtLib
27             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          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 439.

/* 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 file name 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: 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 442.)

```
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
   fmtsearch=(companyFormats deptFormats)
   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 444 to CompanyFormats, MyFmtLib, DeptFormats so that MyFmtLib can override formats in DeptFormats.

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)  /* 1 */
position=replace;

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

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 442.

```
cas casauto listfmranges fmtname=response;
cas casauto listfmranges 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 442, 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 = Global
      Fmtsearch = NA
NOTE: Fmtlib = MYFMTLIB
      Scope = Session
      Fmtsearch = YES

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 446, 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.
```

See Also

*SAS Cloud Analytic Services: User-Defined Formats*

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*.

Data connectors are the software that is used with a caslib to read server-based data sources like databases and Hive. For more information about data connectors, see “Data Connectors” on page 155. For platform file types see “Platform File Types” on page 147.
Syntax

Form 1:  **CASLIB caslib-reference-name <SESSREF=session-reference>**
**DATASOURCE=(SRCTYPE="type" <data-source-options> )**
**<PATH="directory-path"> <options>;**

Form 2:  **CASLIB caslib-reference-name LIST <SESSREF=session-reference> ;**

Form 3:  **CASLIB _ALL_ ASSIGN | _ALL_ LIST <SESSREF=session-reference> ;**

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.

A SAS library that is associated with a caslib can reference only library member names that conform to SAS library naming conventions. 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">, <ENCRYPTIONDOMAIN="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.

Examples  "Example 1: Add a Global Caslib" on page 454
            "Example 5: Encrypt Tables in a Caslib" on page 462

See  "Path-Based Data Source Types and Options" on page 452

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 assigns a libref only 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 456

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. For a global-scope caslib, you can set access controls 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 “Caslib 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 456

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 451

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 3, “CASUTIL Procedure,” on page 69.

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 “Caslib 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 change the active caslib with the CASLIB= session option in the CAS statement. For example, the following CAS statement sets the active caslib to CaslibB:

```
cas casauto sessopts=(caslib=caslibB);
```

For information about the CASLIB= session option, see “CASLIB= Session Option” on page 392.

Path-Based Data Source Types and Options

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 “Data Connectors” on page 155.

**Table 12.3  Path-Based Data Source Types and Options**

<table>
<thead>
<tr>
<th>SRCTYPE= Type</th>
<th>Option Syntax</th>
<th>Example</th>
<th>Supported File Types</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADLS (Azure Data Lake Storage, Gen 2)</td>
<td>DATASOURCE=(SRCTYPE=&quot;ADLS&quot; ACCOUNTNAME=&quot;string&quot; FILESYSTEM=&quot;file-system-name&quot; APPLICATIONID=&quot;string&quot; &lt;DNNSSUFFIX=&quot;network-storage-host-name&quot; &gt; &lt;TIMEOUT=seconds&gt;) &lt;caslib-statement-options&gt; SUBDIRS=TRUE</td>
<td>cas casauto sessopts=(caslib=caslibB) {azureTenantId= 'b1c24e5c-3625-4593-a430-9552373a0c2f'}; caslib azds datasource= {srctype=&quot;adls&quot; accountname=&quot;myaccount&quot; filesystem=&quot;dir&quot; applicationId= &quot;b1fc95d5c-e0e2-45b3-a3cc-a1cf54120f&quot; } subdirs;</td>
<td>CSV, ORC 6</td>
</tr>
<tr>
<td>SRCTYPE</td>
<td>Option Syntax</td>
<td>Example</td>
<td>Supported File Types</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
<td>---------</td>
<td>----------------------</td>
</tr>
</tbody>
</table>
| DNFS | DATASOURCE=(SRCTYPE=DNFS, <ENCRYPTIONDOMAIN ="string">) PATH="file-path"; caslib mycsvs datasource=(srctype="dnfs") path="/data/Mycsvfiles/"; | AUDIO 5  
CSV 3  
DOCUMENT 5  
IMAGE 5  
PARQUET 4  
SASHDAT  
VIDEO 5 | |
| HDFS | DATASOURCE=(SRCTYPE=HDFS, <ENCRYPTIONDOMAIN ="string">) PATH="file-path"; caslib mycsvs datasource=(srctype="hdfs", encryptiondomain="string") path="/data/Mycsvfiles/"; | CSV 3  
SASHDAT | |
| PATH | DATASOURCE=(SRCTYPE=PATH, <ENCRYPTIONDOMAIN ="string">) PATH="file-path"; caslib mycsvs datasource=(srctype="path", encryptiondomain="string") path="/data/Myxlsxfiles/"; | AUDIO 5  
CSV 3  
DOCUMENT 5  
DTA 2  
IMAGE 5  
JMP 2, 5  
ORC 6  
PARQUET 4  
SASHDAT  
SAS7BDAT (specify BASESAS for the fileType parameter) 1  
SAV (specify SPSS for the fileType parameter) 2, 5  
VIDEO 5  
XLS 2  
XLSX (specify EXCEL for the fileType parameter) 2 | |
<table>
<thead>
<tr>
<th>SRCTYPE= Type</th>
<th>Option Syntax</th>
<th>Example</th>
<th>Supported File Types</th>
</tr>
</thead>
</table>
| S3            | DATASOURCE=(SRCTYPE="S3" REGION="region" BUCKET="name" <option(s)>); | caslib mys3 datasource=(srctype="s3" accesskeyid="LASKUFLKJE9EXAMPLE" secretaccesskey="kJflk/M8ofa/bPMFLE" region="EU_Ireland" bucket="myBucket" objectpath="/asr/projB" usessl=true); | - AUDIO 5  
- CSV 3  
- DOCUMENT 5  
- IMAGE 5  
- PARQUET 4  
- SASHDAT  
- VIDEO 5 |

1. Requires the SAS Data Connector to SAS Data Sets.
2. Requires the SAS Data Connector to PC Files.
3. The CSV file type is used to identify any delimited file. For example, to read a semicolon delimited file with a .txt suffix, you can specify CSV as the file type and specify the delimiter.
4. Apache Parquet files are supported for single-machine and distributed servers that use Linux or Linux on Power.
5. These file types are read-only.
6. Apache ORC files are supported for single-machine and distributed servers that use Linux.

Troubleshooting

Here is a message that you might encounter. In most cases, 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;*/
```
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.

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.

Additional Information

- For information about specifying Hadoop data source options, see “Hadoop Data Connector” on page 194.
- 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 option.

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.

```sas
caslib Hadooplib list;
```

SAS Log Showing Data Connector Settings

The settings for uid, server, hadoopConfigDir, hadoopJarPath, and schema appear only 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.

```sas
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)
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)
Type = HDFS
Description = Personal HDFS Caslib
Path = /user/casdemo/
Definition =
Subdirs = Yes
Local = No
Active = No
Personal = Yes

NOTE: Session = CASAUTO Name = HADOOPLIB
Type = hadoop
Description = 'Hadoop Caslib'
Path =
Definition = ui
Subdirs = No
Local = No
Active = No
Personal = No

NOTE: Session = CASAUTO Name = VAPUBLIC
Type = HDFS
Description =
Path = /vapublic/
Definition =
Subdirs = No
Local = No
Active = No
Personal = No

NOTE: Session = CASAUTO Name = Formats
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.

“SAS Log Showing Data Connector Settings” on page 456. 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 “Data Connectors” on page 155.

You can also use the CASUTIL procedure to view information about caslibs. For documentation about the CASUTIL procedure, see Chapter 3, “CASUTIL Procedure,” on page 69.

For the Hadoop connect string syntax, see “Hadoop Data Connector” on page 194.

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 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.

```sas
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.

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 3, “CASUTIL Procedure,” on page 69.

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

```plaintext
caslib ldbeta datasource=(srctype="path")
  path="path-to-directory"
  description="imported files";                 /* 1 */
proc casutil incaslib="LDbeta" outcaslib="hps";                    /* 2 */
  contents casdata="cars.csv";                                    /* 3 */
  load casdata="cars.csv" casout="cars";                          /* 4 */
  list tables incaslib="hps";                                     /* 5 */
  save casdata="cars" incaslib="hps";                             /* 6 */
run;
```

1. The 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. HPS is a global caslib that is available to all users.

3. The CONTENTS statement reads the on-disk file, Cars.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 Cars. The table is now available for analytics.

5. The LIST TABLES statement confirms that the in-memory table named Cars 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.
The following results show the files available in the HPS caslib, displayed by the PROC CASUTIL LIST statement.

Output 12.2 Partial Display of Files Available in the HPS Caslib

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 3, “CASUTIL Procedure,” on page 69.
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

```plaintext
caslib Encr datasource=(srctype="path", encryptionPassword="your-password")
                        path="your-file-path"; /*1*/

libname mycas cas; /*2*/

proc casutil;
    load data=sashelp.cars groupby="make" outcaslib="Encr"; /*3*/
    contents casdata="cars"; /*4*/
    list tables incaslib="encr"; /*5*/

proc mdsummary data=mycas.cars; /*6*/
    var msrp invoice;
    output out=mycas.mdsumstatEncr;
    groupby make;
run;

options obs=15;
proc print data=mycas.mdsumstatEncr; /*7*/
    var Make _Column_ _NObs_ _Mean_ _Max_ _Min_ _Std_;
    title "Summary of MSRP and Invoice, Grouped by Make";
run;

proc casutil incaslib="Encr" outcaslib="Encr"; /*8*/
    save casdata="mdsumstatEncr"; /*9*/
    list files incaslib="encr"; /*10*/
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 DATA= 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.
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 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 12.3  ENCR Caslib Information

<table>
<thead>
<tr>
<th>Caslib Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
</tr>
<tr>
<td>Source Type</td>
</tr>
<tr>
<td>Path</td>
</tr>
<tr>
<td>EncryptionPassword</td>
</tr>
<tr>
<td>Session local</td>
</tr>
<tr>
<td>Active</td>
</tr>
<tr>
<td>Personal</td>
</tr>
<tr>
<td>Hidden</td>
</tr>
</tbody>
</table>
Results: CONTENTS Statement

Output 12.4  Column Information for the Cars Data Set

<table>
<thead>
<tr>
<th>Column</th>
<th>Label</th>
<th>Type</th>
<th>Length</th>
<th>Format Name</th>
<th>Formatted Length</th>
<th>Format Width</th>
<th>Format Decimal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make</td>
<td></td>
<td>char</td>
<td>13</td>
<td>13</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td></td>
<td>char</td>
<td>40</td>
<td>40</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td></td>
<td>char</td>
<td>8</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Origin</td>
<td></td>
<td>char</td>
<td>6</td>
<td>6</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>DriveTrain</td>
<td></td>
<td>char</td>
<td>5</td>
<td>5</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MSRP</td>
<td></td>
<td>double</td>
<td>8</td>
<td>DOLLAR</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>Invoice</td>
<td></td>
<td>double</td>
<td>8</td>
<td>DOLLAR</td>
<td>8</td>
<td>8</td>
<td>0</td>
</tr>
<tr>
<td>EngineSize</td>
<td>Engine Size (L)</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Cylinders</td>
<td></td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Horsepower</td>
<td></td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MPG_City</td>
<td>MPG (City)</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>MPG_Highway</td>
<td>MPG (Highway)</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>Weight (LBS)</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Wheelbase</td>
<td>Wheelbase (IN)</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td>Length (IN)</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
</tbody>
</table>

Results: LIST Statement Information for the Data Source

Output 12.5  List of Files in the Data Source

<table>
<thead>
<tr>
<th>Name</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>Encryption Method</th>
<th>File Size</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>midsumstatEncr.sashdat</td>
<td>-rw-r-xr-x</td>
<td>sasdemo</td>
<td>unix</td>
<td>AES</td>
<td>19.6KB</td>
<td>06Apr2016:20:50:26</td>
</tr>
</tbody>
</table>

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 3, “CASUTIL Procedure,” on page 69.

- 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 4, “MDSUMMARY Procedure,” on page 107.
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 file names.

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 409

---

**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=

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 | integerK | integerM | integerG | ALL
```

Syntax Description

- `integer` specifies the total number of bytes to read.
- `integerK` specifies the total number of kilobytes to read.
- `integerM` specifies the total number of megabytes to read.
- `integerG` 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 Statement Option” on page 42
- “DATALIMIT= Data Set Option” on page 55

---

**CASHOST= System Option**

Specifies the CAS primary and backup host names that are associated with a CAS session.
Category: CAS
PROC OPTIONS GROUP= CAS
Notes: This option can be restricted by a site administrator. For more information, see “Restricted Options” in SAS System Options: Reference.
When you set system option CASHOST, macro variable _CASHOST_ is also set to the value that you specify for system option CASHOST.
When a connection is established, system option CASHOST and macro variable _CASHOST_ are updated to specify the host name of the current primary controller and, if available, the host name of the backup controller.
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-controller-host-name"
CASHOST= ("primary-controller-host-name" <<,> "backup-controller-host-name")

Syntax Description

primary-controller-host-name
specifies the CAS primary server host name.
Default: None
Range: 1–256 characters

backup-controller-host-name
specifies the backup CAS server host 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-controller-host-name is returned. backup-controller-host-name cannot be returned at this time.
Requirement: Parentheses are required when you specify both a primary controller host name and a backup controller host name. You can use either a space or a comma between the primary host name and the backup host name.
Details

Determining the Backup Host Name

To determine whether a backup host is available, connect to the primary server host and port, and then run the following code:

```sas
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. When the CASHOST option is set, macro variable _CASHOST_ is also set to the value that is specified for the CASHOST option.

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.

Use the CAS procedure to see the details for the primary and backup hosts as described in “Determining the Backup Host Name” on page 471. To change to a different CAS server primary controller, specify the new primary controller host name and backup controller host name, if available, in CASHOST=. If you specify the primary controller host name only and a backup controller is available, system option CASHOST= is updated to include the current backup host name when you connect to the server. If you specify an incorrect backup host name, system option CASHOST= is updated to include the correct backup host name when you connect. In both cases, macro variable _CASHOST_ is also updated to include the current backup controller host name.

When a Primary Controller Failure Occurs

If primary-controller-host-name fails, backup-controller-host-name automatically becomes the primary CAS server controller. Existing sessions automatically connect to backup-controller-host-name. New sessions connect to backup-controller-host-name. When a new session connects to backup-controller-host-name, the CASHOST system option and macro variable _CASHOST_ are updated to specify the backup controller host name only. When the primary server is subsequently restored and a new session connects to primary-controller-host-name, the CASHOST system option and macro variable _CASHOST_ are updated to specify the primary and secondary server host names.

Controller failure is transparent to users. It is more of a concern for administrators.

Example

```sas
options cashost="cloud.example.com";
options cashost=("cloud.example.com" "cloudbackup.example.com");
```
CASLIB= System Option

Specifies the caslib name for the session that is identified by the SESSREF= option.


Category: CAS

PROC OPTIONS GROUP=

Interaction: The CAS statement session option CASLIB= overrides this option. For more information, see “CASLIB= Session Option” on page 392.

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:

```sql
cas mysess sessopts=(caslib=mycaslib collate=UCA);
```

See “SESSOPTS=(session-options)” on page 423.

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:
CASNCHARMULTIPLIER= System Option

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


Category: CAS

PROC OPTIONS GROUP= CAS

Defaults: For SBCS environments, the shipped default is 1.
For DBCS environments, the shipped default is 1.5.

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 to the CAS server.

Range 0 < n ≤ 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 Statement Option” on page 44
- “NCHARMULTIPLIER= Data Set Option” on page 58

CASNWORKERS= System Option

Specifies the number of worker nodes to use for a CAS session.


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 409 and “NWOKERS= Session Option” on page 405.

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.
CASPORT= System Option

Specifies the port to use when connecting to CAS.


Category: CAS

PROC
OPTIONS
GROUP=

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.


Category: CAS
PROC OPTIONS GROUP=

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:

```sas
cas mysess sessopts=(caslib=mycaslib collate=UCA);
```

See “SESSOPTS=(session-options)” on page 423.

Syntax

```
CASSESSOPTS=(session-options)
SESSOPTS=(session-options)
```

Syntax Description

**session-options**

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:

```sas
%put caslib=%sysfunc(GETSESSOPT(session, option));
```

See “Session Options” on page 387 for a list of the options that you can specify for **session-options**.

Example

For the default session, set the caslib to MYCASLIB and the session connection time-out to 60 minutes:

```sas
options sessopts=(caslib="mycaslib" timeout=3600)
```

See Also

Functions:

- “GETSESSOPT Function” on page 379

Macro Statement and Functions:

- “%PUT Macro Statement” in SAS Macro Language: Reference
- “%SYSFUNC, %QSYSFUNC Macro Functions” in SAS Macro Language: Reference

Statements:
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.


Category: CAS

PROC OPTIONS GROUP=

Default: 60

Interaction: The TIMEOUT= session option overrides this option. For more information, see “TIMEOUT= Session Option” on page 405.

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 | nK | nM | hexX | MAX | MIN

Syntax Description

\( n \mid nK \mid nM \)

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.

\( hexX \)

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.

MAX

sets the timeout value to 31536000.

MIN

sets the timeout 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

```plaintext
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=

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**

```plaintext
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 424.
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

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

"Determining Where the DATA Step Is Running" 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=

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 409.

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 409