# Contents

_About This Book_ ................................................................. v

_What’s New in 9.4 for SAS Cloud Analytic Services_ .............................. vii

**Chapter 1 • CAS Statement** ................................................................. 1
  Dictionary ........................................................................ 1

**Chapter 2 • CAS LIBNAME Engine** ......................................................... 33
  CAS LIBNAME Engine Overview ...................................................... 34
  Use of WHERE with the CAS LIBNAME Engine .................................. 35
  Working with Temporary Columns .................................................... 35
  Getting Started ...................................................................... 36
  Data Types ........................................................................ 37
  Variable Names and Data Set Names in CAS Engine ......................... 45
  Dictionary ........................................................................... 46

**Chapter 3 • CASLIB Statement** ............................................................. 69
  Dictionary ........................................................................... 69

**Chapter 4 • CASUTIL Procedure** ........................................................... 87
  Overview: CASUTIL Procedure ..................................................... 88
  Syntax: CASUTIL Procedure ......................................................... 89
  Enclose Values in Quotation Marks .................................................. 100
  Subdirectories and Filename Matching ............................................. 100
  Limitations and Restrictions .......................................................... 102
  Results: CASUTIL Procedure ....................................................... 103
  Examples: CASUTIL Procedure .................................................... 105

**Chapter 5 • MDSUMMARY Procedure** .................................................. 113
  Overview: MDSUMMARY Procedure .............................................. 113
  Syntax: MDSUMMARY Procedure .................................................. 113
  PROC MDSUMMARY Output Data Sets ............................................ 116
  Results: MDSUMMARY Procedure ................................................ 116
  Examples: MDSUMMARY Procedure ............................................. 117

**Chapter 6 • Platform Data Sources** ....................................................... 131
  Data Redundancy .................................................................. 131
  Dictionary ........................................................................... 132

**Chapter 7 • Platform File Types** ............................................................ 139
  Dictionary ........................................................................... 139

**Chapter 8 • Data Connectors** ................................................................. 145
  Working with SAS Data Connectors ................................................ 145
  Quick Reference for Data Connector Syntax ..................................... 146
  Where to Specify Data Connector Options ....................................... 147
  Using Wildcards for Pattern Matching ............................................ 148
  Loading a Subset of Table Rows .................................................... 149
  Dictionary ........................................................................... 150
Chapter 9 • Data Types .................................................. 199
  SAS Cloud Analytic Services Data Types .................................. 199

Chapter 10 • Functions .................................................. 201
  Dictionary ................................................................. 201

Chapter 11 • Macro Variables .............................................. 207
  Dictionary ................................................................. 207

Chapter 12 • Session Options .............................................. 209
  Setting Session Options ................................................ 209
  Session Options by Category .......................................... 211
  Dictionary ................................................................. 212

Chapter 13 • System Options .............................................. 225
  Dictionary ................................................................. 225
About This Book

Audience

This book supports the use of SAS Cloud Analytic Services by participating SAS solutions. It explains key concepts and provides essential instructions. The emphasis is on server operation, data management, and security configuration.
What’s New in 9.4 for SAS Cloud Analytic Services

Overview

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

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

• New options have been added to the CAS statement to connect to the SAS Metadata Server.
• New functions have been added to provide information about your CAS session.
• New system options have been added to control data transfer between SAS and CAS.
• A new macro has been added to migrate user-defined formats from SAS catalogs to CAS format libraries.

Enhanced CAS Statement

The following options are new in SAS 9.4M5:

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

For more information, see “CAS Statement”.

Note: These options can be set only from SAS 9.4M5 and not from SAS Viya 3.2.

Functions

The following functions are new in SAS 9.4M5:

• CLIBEXIST on page 201 indicates whether a caslib name exists.
• GETCASURL on page 202 returns the CAS Server Monitor URL.

• GETLCASLIB on page 202 returns the caslib for a CAS LIBNAME engine libref that was specified by the CASLIB= LIBNAME option.

• GETLSESSREF returns the session reference for a CAS LIBNAME engine libref.

• GETLTAG returns the tag for a CAS LIBNAME engine libref that was specified by the TAB= LIBNAME option.

• GETSESSOPT returns the value of a CAS server session option.

• SESSFOUND returns a value to indicate whether a named session that you started in your SAS session is found.

System Options

The following system options are new in SAS 9.4M5:

• Data transfer of CAS tables can impede system performance if the CAS table is very large. You can use the CASDATALIMIT= system option to limit the amount of data in a single CAS table that can be transferred from the CAS server to SAS. By default, the amount of data that can be read from a CAS table is 100M.

• When SAS writes data to the CAS server using the CAS engine, the engine estimates the number of bytes that are needed to transcode the data to UTF-8 based on the character set of the SAS session. The character set is specified by the ENCODING= system option. SBCS environments estimate one byte in UTF-8 for every one byte in the local encoding. DBCS environments estimate 1.5 bytes in UTF-8 for every one byte in the local encoding. You can use the CASNCHARMULTIPLIER= system option to replace the estimate with an explicit value of the byte multiplier when you know the number of bytes that are needed to represent the data in UTF-8.

• By default, the DSCAS system option on page 235 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.

Note: These system options can be set only from SAS 9.4M5 and not from SAS Viya 3.2.

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.
Chapter 1

CAS Statement

Dictionary

CAS Statement

Starts and manages your SAS Cloud Analytic Services session.

Syntax

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

Summary of Optional Arguments

Authentication option

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

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

CAS server information options

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

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

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

Connection options
DISCONNECT
disconnects SAS from the session.

HOST="host-name"
specifies the machine name for the control node of the server.

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

RECONNECT
reconnects to a session using a session name.

TERMINATE
terminates the SAS Cloud Analytic Services session.

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

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

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

Session information options

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

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

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

Session property options

LISTSESSOPTS
lists the session options.

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

User-defined format options

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

DROPFMLIB FMLIBNAME=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.

LISTFMRANGES FMLNAME=format-name
lists the ranges and labels for a format.

LISTFMTSEARCH
displays the format library search list for the session.

\[
\text{LISTFORMATS} \ <\text{FMTLIBNAME=}format-library-name> \ <\text{SCOPE=}BOTH \mid SESSION \mid GLOBAL \mid \text{MEMBERS}>
\]
lists the user-defined format libraries that are known to SAS.

\[
\text{PROMOTE} \ <\text{FMTLIBNAME=}format-library-name> \ <\text{REPLACE}>
\]
promotes a session-local format library to a global format library.

\[
\text{SAVE} \ <\text{FMTLIBNAME=}format-library-name> \ <=\ <\text{TABLE=}table-name> \ <\text{CASLIB=}caslib> \ <\text{REPLACE}> \ | \ <\text{PATH=}path> \ <\text{PROMOTE}>
\]
saves a session format library to a CAS table or to a file.

**Required Argument**

\**session-name**

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

**Notes**

Do not enclose `session-name` in quotation marks.

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

The SAS Cloud Analytic Services system administrator can limit the number of concurrent sessions on the CAS server. See “Maximum Number of Sessions on the Server” on page 15.

**Optional Arguments**

\**ADDFMTLIB FMTLIBNAME=format-library-name <= <TABLE=table-name> <CASLIB=caslib> > | <PATH=path> > <POSITION=APPEND \mid INSERT \mid REPLACE \mid NONE> <PROMOTE> <REPLACEFMTLIB>*

adds a session format library. By default, the format library is appended to the format library search list.

\**FMTLIBNAME=format-library-name**

specifies the name of the format library to add.

**Default**

`_FMTLIBn`, where `n` starts at 1 and is incremented for each format library that is added.

**Note**

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

\**TABLE=table-name**

specifies the name of the table where the format library was previously saved using `SAVEFMTLIB`.

**Default**

the name specified in the `FMTLIBNAME=` option

**Notes**

Do not specify this option with `PATH=`.

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

\**CASLIB=caslib**

specifies the name of the caslib where the table is saved.
Default: the active caslib for the session

Note: Do not specify this option with PATH=.

**PATH=path**

specifies the name of the file where the format information is saved.

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

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

**POSITION=APPEND | INSERT | REPLACE | NONE**

specifies the position of this format library in the format-library search list.

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

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

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

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

Default: **APPEND**

**PROMOTE**

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

**REPLACEFMTLIB**

replaces the format library if it already exists.

**Tip**

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

**See**

*SAS Cloud Analytic Services: User-Defined Formats*

**Example**

“Example 11: Add a Format Library That Is Stored in a SASHDAT File” on page 27

**AUTHDOMAIN=authentication-domain | "authentication-domain"**

obtains credentials from the SAS Metadata Server or the SAS Viya credentials service. SAS system option SERVICESBASEURL specifies the root URL for Viya services. If it is set, SAS first requests credentials for authentication-domain from the SAS Viya credentials service. If system option SERVICESBASEURL is not set or if credentials cannot be obtained from the SAS Viya credentials service, SAS requests credentials from the SAS Metadata Server.

If credentials are obtained, a note is written to the SAS log identifying the source of the credentials. Otherwise, notes are written to the SAS log indicating that credentials could not be obtained, and a session is not started.
**authentication-domain**  
specifies the name of an authentication domain object registered on the SAS Viya credentials service or the SAS Metadata Server that associates user credentials with an identity.

**Valid in**  
SAS 9.4M5 and in later releases

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

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

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

To connect to the CAS server using credentials obtained from the SAS Metadata Server, the CAS server host name and port must be specified by the HOST= on page 7 and PORT= on page 11 options, or they must be obtained from the SAS Metadata Server by specifying the CASSERVERMD= on page 6 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

**Example**  
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;
```

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

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

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

**Note**  
An authinfo file is required when running batch jobs.

**See**  
“Create an Authinfo File” in Client Authentication Using an Authinfo File
CASSERVERMD=server-name | "server-name"
specifies the name of a server object registered on the SAS Metadata Server that
associates SAS Cloud Analytic Services connection parameters with a server name.

Valid in SAS 9.4M5 and in later releases

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

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

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

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

Example This example requests a connection to CAS, which is the name of a
CAS server registered on the SAS Metadata Server:
cas casauto casservermd="cas";

DISCONNECT
disconnects SAS from the session.

Note The session name is preserved for use with the RECONNECT option for the
duration of the SAS session. If the SAS session is terminated after the session
is disconnected, use the session’s UUID to reconnect. See “Example 8:
Connect to an Existing Session” on page 24.

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

See “Example 7: Disconnect from a Session” on page 24

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

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

FMTSEARCHREMO
removes the format library from the format search list.

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

See SAS Cloud Analytic Services: User-Defined Formats

Example “Example 18: Drop a Format Library from Session Scope” on page 32

FMTSEARCH=(name1 <name2...nameN>) <POSITION=APPEND | INSERT |
REPLACE>
modifies the format library search list for the session.
(name1 <name2...nameN> )
specifies a list of one or more format-library names enclosed in parentheses. Each name is separated by a space.

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

- **APPEND** appends the format libraries to the end of the format-library search list.
- **INSERT** inserts the format libraries at the beginning of the format-library search list.
- **REPLACE** replaces the current format-library search list with the specified format libraries.

**Default** APPEND

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

**See** [SAS Cloud Analytic Services: User-Defined Formats](#)

**Example** Add format libraries FMTLIB1 and FMTLIB2 to the beginning of the format library search list for session Casauto:
cas casauto fmtsearch=(fmtlib1 fmtlib2) position=insert;

**Examples**
- “Example 14: Add Global Format Libraries to a Session’s Format Search List” on page 29
- “Example 15: Change the Format Library Search List” on page 30

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

**See** [SAS Cloud Analytic Services: User-Defined Formats](#)

**Example** Clear the format search list for session Casauto:
cas casauto fmtsearch clear;

**HOST=’host-name’**
specifies the machine name for the control node of the server.

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

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

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

**See**
- “_CASHOST_ Macro Variable” on page 207
- “CASHOST= System Option” on page 227
**LIST**

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

**Note**  The server is not accessed for the information.

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

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

See  “Example 2: List Information about the Sessions in Your SAS Client” on page 21

**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>Hostname</td>
<td>SAS Cloud Analytic Services 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>

8 Chapter 1 • CAS Statement
<table>
<thead>
<tr>
<th>Section</th>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>License</td>
<td>site</td>
<td>Site information.</td>
</tr>
<tr>
<td></td>
<td>siteNum</td>
<td></td>
</tr>
<tr>
<td></td>
<td>expires</td>
<td>License expiration date and time.</td>
</tr>
<tr>
<td></td>
<td>gracePeriod</td>
<td>Grace period in days.</td>
</tr>
<tr>
<td></td>
<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. This option is required. The format libraries are searched in the format-library search list. 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 31

**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 29

“Example 15: Change the Format Library Search List” on page 30

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

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

**SCOPE=BOTH | SESSION | GLOBAL**
specifies the scope.

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

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

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

Default **BOTH**
MEMBERS
lists the names of the members in each format library.

See SAS Cloud Analytic Services: User-Defined Formats

Example “Example 13: List the Session-Scope Formats in a Session” on page 29

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

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

Range 0–1999999999

Note 0 is equivalent to _ALL_

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

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

Examples
Print the last 10 actions that were executed in the current session:
cas casauto listhistory;

Print the last 5 actions that were executed in the current session:
cas casauto listhistory 5;

Print all of the actions that were executed in the current session:
cas casauto listhistory _all_;

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

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

Alias LISTSSO

Restriction You must have administration privileges to use this option.

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

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

Use session Casauto to list all of your sessions.

```sql
CAS Statement  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.

### PORT=number

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

- **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.

### PROMOTEFMTLIB FMTLIBNAME=format-library-name <REPLACE>

Promotes a session-local format library to a global format library.

- **FMTLIBNAME=format-library-name**: Specifies the name of the format library. This option is required.
- **REPLACE**: Replaces the format library if it is already promoted.
- **Tip**: Global format libraries are not automatically included in the format search list for new sessions. Use FMTSEARCH on page 6 to add any global format libraries that you want to use to your new session’s format search list.

### RECONNECT

Reconnects to a session using a session name.
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 13.

See “Example 8: Connect to an Existing Session” on page 24

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

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

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

Default the name specified in the FMTLIBNAME= option

Notes Do not specify this option with PATH=.

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

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

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

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

Default the active caslib for the session

Requirement You must have Write access to the specified caslib.

Note Do not specify this option with PATH=.

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

REPLACE
replaces the table if it already exists.

Alias REPLACE

Note This option is ignored when PATH= is specified.

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

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

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

See SAS Cloud Analytic Services: User-Defined Formats
Example 10: Create a Format Library and Save It to a SASHDAT File on page 25

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

See Chapter 12, “Session Options,” on page 209 for a list of the options that you can specify for session-option(s).

“Example 5: Change a Property for a Session” on page 23

“Program: Start a Session with Custom Properties” on page 20

TERMINATE
terminates the SAS Cloud Analytic Services session.

Alias CLEAR

Note When you terminate the active session, SAS system option SESSREF= continues to reference the terminated session. In SAS 9.4M5, 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 20 and “SESSREF= System Option” on page 235.

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

See “Example 9: Terminate a Session” on page 25

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

Alias CASUSER

Default SAS system option CASUSER=, if set.

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


Interaction This option overrides SAS system option CASUSER=.

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

See “CASUSER= System Option” on page 234

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

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

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

**Tip**

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

**See**

“LIST” on page 8

“Program: Connect to an Existing Session Using the Session Name and UUID” on page 24

**UUIDMAC=macro-variable-name**

specifies a SAS macro variable name into which the UUID of the session is stored.

**Tip**

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

**See**

“Program: Start a Session and Store the UUID in a Macro Variable” on page 21

---

**Details**

**What Can I Do with the CAS Statement?**

You can do the following tasks with the CAS statement:

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

**Starting Your Initial Session**

After you sign in to SAS Studio, you must start a session in order to connect to SAS Cloud Analytic Services. After you start your session, you can use it to complete your tasks. Code snippet **New Session** in SAS Studio provides the SAS code that is needed to start a session. Alternatively, you can submit your own CAS statement to start a customized session. See “Example 1: Start a Session” on page 20 for examples. Use the CAS statement to perform management tasks on your session, such as listing or changing properties, managing format libraries, and so on.

**CAS Statement Status**

When you execute a CAS statement for the first time in your SAS session, global macro variable CASSTMTERR is created. In SAS 9.4M5, 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 24.

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

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

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

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

Use the GETCASURL function to get the CAS Server Monitor URL. See “GETCASURL Function” on page 202.

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

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

### Table 1.1 Performing Common Tasks for User-Defined Formats Using the CAS Statement

<table>
<thead>
<tr>
<th>Task</th>
<th>Option To Use</th>
<th>Example(s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Add a user-defined format library stored in a SASHDAT file to your session.</td>
<td>ADDFMTLIB (p. 3)</td>
<td>“Example 11: Add a Format Library That Is Stored in a SASHDAT File” (p. 27)</td>
</tr>
<tr>
<td>Promote a session-scope user-defined format library in your session to global scope.</td>
<td>PROMOTEFMTLIB (p. 11)</td>
<td>“Example 17: Promote a Format Library to Global Scope” (p. 31)</td>
</tr>
<tr>
<td>List the format libraries and formats that are available in your session.</td>
<td>LISTFORMATS (p. 9)</td>
<td>“Example 13: List the Session-Scope Formats in a Session” (p. 29)</td>
</tr>
<tr>
<td>Task</td>
<td>Option To Use</td>
<td>Example(s)</td>
</tr>
<tr>
<td>------</td>
<td>---------------</td>
<td>------------</td>
</tr>
<tr>
<td>List the ranges in a user-defined format in your session.</td>
<td>LISTFMTRANGES (p. 9)</td>
<td>“Example 16: List the Ranges in a Format” (p. 31)</td>
</tr>
</tbody>
</table>
| Display the user-defined format library search list for your session. | LISTFMTSEARCH (p. 9) | “Example 14: Add Global Format Libraries to a Session’s Format Search List” (p. 29)  
“Example 15: Change the Format Library Search List” (p. 30) |
| Modify your session’s user-defined format library search list. | FMTSEARCH (p. 6) | “Example 14: Add Global Format Libraries to a Session’s Format Search List” (p. 29)  
“Example 15: Change the Format Library Search List” (p. 30) |
| Save a format library in your session to a SASHDAT file. | SAVEFMTLIB (p. 12) | “Example 10: Create a Format Library and Save It to a SASHDAT File” (p. 25) |
| Drop a session-scope or global-scope user-defined format library. | DROPFMTLIB (p. 6) | “Example 18: Drop a Format Library from Session Scope” (p. 32) |

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

**Troubleshooting Session Errors**

**What Session-Related Errors Does This Section Cover?**
This section covers the following CAS session-related errors:

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

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

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

**Note:** The warning message might not appear.

<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 24.  
• Verify that session session-name exists. See “Example 3: List Information about All of Your Sessions” on page 22. If session session-name does not exist, specify a different session or start session session-name. See “Example 1: Start a Session” on page 20. |

### 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 24.  
• Verify that session session-name exists. See “Example 3: List Information about All of Your Sessions” on page 22. If session session-name does not exist, specify a different session or start session session-name. See “Example 1: Start a Session” on page 20. |
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>
<tbody>
<tr>
<td>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.</td>
<td>• 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 22. 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 20.</td>
</tr>
</tbody>
</table>

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>Cause</th>
<th>Solution</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 15.</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.
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 24.

• Verify that session session-name exists. See “Example 3: List Information about All of Your Sessions” on page 22. If session session-name does not exist, specify a different session or start session session-name. See “Example 1: Start a Session” on page 20.

### 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.

A request was made on session session-name, but session session-name is disconnected, does not exist, or is not known to SAS.

• If a warning message indicates that session-name is disconnected, reconnect to session session-name. See “Example 8: Connect to an Existing Session” on page 24.

• Verify that session session-name exists. See “Example 3: List Information about All of Your Sessions” on page 22. If session session-name does not exist, specify a different session or start session session-name. See “Example 1: Start a Session” on page 20.

### 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.

A request such as LISTSESSOPTS or LISTFORMATS was made on session session-name, but session session-name is disconnected.

Reconnect to session session-name. See “Example 8: Connect to an Existing Session” on page 24.
Unable to Connect to Cloud Analytic Services

One of the following error messages appears:

```
ERROR: Unable to connect to Cloud Analytic Services host-name
on port host-port Verify connection parameters and retry.
ERROR: Connection failed. Server returned: Authentication
failed: Access denied.
```

<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.

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

**SAS Log**

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

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

**Program: Start a Session with Custom Properties**

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

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

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

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

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

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

```sas
/* options cashost="cloud.example.com" casport=5570; */
cas casauto uuidmac=casauto_uuid;
%put Session casauto UUID: &casauto_uuid;
```

SAS Log

The %PUT statements 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 69 and “CAS LIBNAME Statement” on page 46.
- If you need to disconnect from your session, be sure to set the TIMEOUT property
  for the session to an appropriate value before you disconnect. If no other client
  connections exist and no actions are running when you disconnect, your session is
  automatically terminated if you do not reconnect within the time-out period. See
  TIMEOUT= on page 222.
- 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, which are Casauto, Mysess1, and Mysess2, in this example:

```
cas _all_ list;
```

SAS Log

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

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

Example 3: List Information about All of Your Sessions

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

```
cas casauto listsessions;
```

SAS Log

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

```
NOTE: SessionName = CASAUTO:Mon Feb  8 12:49:00 2016
    UUID= session-UUID
    State = Connected
    Authentication = Active Directory
    Userid = sasdemo
NOTE: SessionName = MYSESS1:Mon Feb  8 12:49:03 2016
    UUID= session-UUID
    State = Connected
    Authentication = Active Directory
    Userid = sasdemo
NOTE: SessionName = MYSESS2:Mon Feb  8 12:49:05 2016
    UUID= session-UUID
    State = Connected
    Authentication = Active Directory
    Userid = sasdemo
NOTE: Request to LISTSESSIONS completed for session CASAUTO.
```
**Example 4: List the Properties for a Session Program**

This example lists the properties for session Casauto:

```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
  Min = 0
  Max = 0
  Description = specifies the string to prefix to log messages.

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

**Additional Information**

For information about the session properties, see Chapter 12, “Session Options,” on page 209. The GETSESSOPT function enables you to get the value of a single property. See “GETSESSOPT Function” on page 204.

**Example 5: Change a Property for a Session Program**

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

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

**Additional Information**

For information about the session properties that you can change, see Chapter 12, “Session Options,” on page 209. To change one or more properties for all of your generated SAS Cloud Analytic Services sessions, specify the property settings in an OPTIONS statement. See “CAS Statement” on page 1.

**Example 6: 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)) minutes.;
```
SAS Log

The following is written to the SAS log:

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

**Example 7: Disconnect from a Session**

**Before You Disconnect from Your Session**

If no other client connection exists and no actions are running when you disconnect, you must reconnect to the session before the connection time-out expires (60 seconds by default). Otherwise, the session is automatically terminated. Before you disconnect from a session, be sure to set session option `TIMEOUT=` for that session to an appropriate value. Use the `RECONNECT` option to reconnect to the session. See “Example 8: Connect to an Existing Session” on page 24.

**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.
```

**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:

```sas
   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” on page 23.
about All of Your Sessions” on page 22. This example uses session name Mysess and the UUID option to connect to session Mysess, which was started in a different SAS session.

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

### SAS Log

The following notes are written to the SAS log.

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

---

### Example 9: Terminate a Session

**Program**

Terminate session Casauto:

```sas
cas casauto terminate;
```

### SAS Log

The following notes are written to the SAS log:

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

---

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

**Program**

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

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

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

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

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

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

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 filename. The REPLACE option replaces the table file if it already exists in CASUSER. If the table already exists and you do not specify REPLACE, an error occurs.
Example 11: Add a Format Library That Is Stored in a SASHDAT File

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 25.
If you have not already done so, start your CAS session.

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

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

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

SAS Log

```
1          cas casauto;
  ...        
2          cas casauto addfmtlib fmtlibname=myFmtLib
3            table="myFmtLib.sashdat" caslib=casuser replacefmtlib;
5              NOTE: Request to ADDFMTLIB MYFMTLIB completed for session CASAUTO.
6          cas casauto listformats fmtlibname=myFmtLib;
7              NOTE: Fmtlib = MYFMTLIB
8              Scope = Session
9              Fmtsearch = YES
10             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;
```
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;
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 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 27.)

```
cas casauto listformats;                       /* 1 */
cas casauto                                    /* 2 */
    fmtsearch=(companyFormats deptFormats)    /* 2 */
    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**

```sas
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   cas casauto
3      fmtsearch=(companyFormats deptFormats)
4      position=insert;
5      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 29 to CompanyFormats, MyFmtLib, DeptFormats so that MyFmtLib can override formats in DeptFormats.

```sas
   cas casauto
      fmtsearch=(companyformats, myFmtLib, deptFormats) /* 1 */
      position=replace;
   cas casauto listfmtsearch; /* 2 */
```

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

2 Use LISTFMTSEARCH to verify the new search list.
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 27.

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

Example 17: Promote a Format Library to Global Scope

Promote format library MyFmtLib, which was added in “Example 11: Add a Format Library That Is Stored in a SASHDAT File” on page 27, 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

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

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

**Example 18: Drop a Format Library from Session Scope**

*Program*

Drop format library MyFmtLib, which was promoted in “Example 17: Promote a Format Library to Global Scope” on page 31, from session scope.

```sas
1 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

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

cas casauto listformats;
NOTE: Fmtlib = MYFMTLIB
  Scope = Global
  Fmtsearch = NA
NOTE: Request to LISTFORMAT completed for session CASAUTO.
```
Chapter 2
CAS LIBNAME Engine

CAS LIBNAME Engine Overview .................................................. 34
Use of WHERE with the CAS LIBNAME Engine ................................. 35
Working with Temporary Columns ................................................. 35
Getting Started .............................................................................. 36
Data Types .................................................................................... 37
  Data Type Definitions ................................................................. 37
  Support for Implicit Declaration of Data Types ......................... 38
  VARCHAR Data Type ................................................................. 39
Variable Names and Data Set Names in CAS Engine ......................... 45
Dictionary ..................................................................................... 46
  CAS LIBNAME Statement .......................................................... 46
  CASLIB= LIBNAME Option ......................................................... 48
  COMPRESS= LIBNAME Option .................................................... 49
  DATALIMIT= LIBNAME Option .................................................... 49
  NCHARMULTIPLIER= LIBNAME Option ...................................... 50
  MAXTABLEMEM= LIBNAME Option .............................................. 51
  READTRANSFERSIZE= LIBNAME Option ..................................... 52
  TAG= LIBNAME Option ............................................................ 53
  TRANSCODE_FAIL= LIBNAME Option ........................................ 54
  Writtetransfersize= LIBNAME Option ........................................ 54
  APPEND= Data Set Option ......................................................... 55
  CASLIB= Data Set Option ........................................................ 56
  COMPRESS= Data Set Option ..................................................... 56
  COPIES= Data Set Option ........................................................ 57
  DATALIMIT= Data Set Option ..................................................... 58
  DUPLICATE= Data Set Option ..................................................... 59
  MAXTABLEMEM= Data Set Option ............................................... 59
  NCHARMULTIPLIER= Data Set Option ......................................... 60
  ONDEMAND= Data Set Option ................................................... 61
  ORDERBY= Data Set Option ...................................................... 61
  PARTITION= Data Set Option .................................................... 63
  PROMOTE= Data Set Option ...................................................... 64
  READTRANSFERSIZE= Data Set Option ...................................... 65
  SCRIPT= Data Set Option ......................................................... 66
  TAG= Data Set Option ............................................................ 66
  TEMPNAMES= Data Set Option .................................................. 67
  TRANSCODE_FAIL= Data Set Option .......................................... 67
  Writtetransfersize= Data Set Option ......................................... 68
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 9.4 session to an existing CAS session The libref then becomes your link between SAS and the CAS server. When you assign a CAS engine libref, you are associating the libref with a CAS session and a caslib in order to work with CAS in-memory tables.

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.*/
libname mycas cas;

/* In the following statement, the libref Mycas is bound to the testTables caslib, even if the active caslib changes to a caslib other than testTables. */
libname mycas cas caslib=testTables;

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

   cas casauto terminate;

Note: See Base SAS Procedures Guide for specific information about working with procedures in CAS.
Use of WHERE with the CAS LIBNAME Engine

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

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

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

Working with Temporary Columns

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

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

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

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

filename newcols temp;                                    /*1*/
data _null_;                                              /*2*/
    file newcols;
    put "t1 = round(mpg_highway/mpg_city,0.5);"; /*3*/
    put "t2 = round(cylinders/enginesize,0.2);"; /*4*/
    put "t3 = msrp / invoice;"; /*5*/
```
run;
libname mycas cas sessref=casauto;
proc mdsummary data=mycas.cars(tempnames=(t1 t2 t3) script=newcols);
    output out=mycas.carssum;
    var mpg_city mpg_highway t1 t2 t3;
run;

proc print data=mycas.carssum;
    var _column_ -- _sum_; 
run;
1 Instead of naming the location of the file, the code shows the use of TEMP, which means that the values are stored in a temporary file. A path can be specified instead of using TEMP.
2 A NULL DATA step is used to write three expressions to the temporary file. As an alternative, you can enter the expressions in a text file and specify the path instead of using the TEMP device with the FILENAME statement.

The ONDEMAND= option is optional for working with temporary columns. ONDEMAND= enables users to customize how their data is evaluated: one row at a time or collectively at the outset.

---

### Getting Started

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

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

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

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

```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. SAS DATA step programs require a CAS engine libref on both the input and output table names.

4. Add the Sashelp.cars data set as an in-memory table.

5. Run an in-memory DATA step to create a new table with a subset of the rows and columns and add a new column. The input and output tables must begin with the CAS engine libref Mycas.

6. View five rows from the new data. The engine performs data transfer from the CAS server to SAS.

7. Use the Mycas libref to identify the input table for a SAS Viya procedure.

---

Data Types

Data Type Definitions

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

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Example</th>
<th>Missing Value</th>
<th>Note</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fixed Length Character</td>
<td>Stores a fixed-length character string. <em>n</em> 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 mcas.greets; length greet1 $16; greet1 = &quot;Good day&quot;; run; cas data greets;</td>
<td>all blanks</td>
<td></td>
</tr>
<tr>
<td>CHAR(<em>n</em>)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Varying Length Character</td>
<td>Stores a varying-length character string. <em>n</em> is the maximum number of characters to store. Range 1 - 536,870,911 characters (UTF-8 encoding)</td>
<td>data mcas.info; length lastname varchar(40); lastname=&quot;Adams&quot;; run;</td>
<td>all blanks, or zero length</td>
<td></td>
</tr>
<tr>
<td>VARCHAR(<em>n</em>)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DOUBLE (numeric)</td>
<td>Stores a numeric value, including dates and times, as a floating-point number. Range: 8 bytes for the CAS Engine*</td>
<td>data mcas.info; length count 8; count = 25; run;</td>
<td>Any one of the following forms: . .A-Z .-</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

* If a shorter length is specified, then 8 bytes are used and a note is printed to the log.
** The CAS engine is required for storing VARCHAR variables in the output table of a DATA step. The DATA step can read CAS tables containing VARCHAR variables, but it cannot store them unless a CAS engine libref is specified on the output table.

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

### Support for Implicit Declaration of Data Types

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

Implicit declaration means that you do not have to explicitly declare a variable’s type or length before using it. You can create a new variable and use it for the first time in an assignment statement without having to explicitly declare its type or length. When you create a variable this way, SAS determines the type based on the values that you assign to the variable.

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

*Note:* This is different from the V9 engine, which supports a length range of 1 - 8 bytes for NUMERIC types.

- Implicit type declaration is not supported for VARCHAR variables. VARCHAR variables must be explicitly declared in either a LENGTH statement or an ARRAY statement.

In the following DATA step, the type and length for variables x and y are set implicitly by SAS:
data test;
  x=1;
  y='hello';
run;

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

**VARCHAR Data Type**

**Definition**

VARCHAR(* | length*)

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

**Syntax**

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

**variable-name**

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

**length**

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

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

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

**Range**

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

**See**

“PROC CONTENTS Output for Actual and Defined Length” on page 40

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

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

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

**See**

“PROC CONTENTS Output for Actual and Defined Length” on page 40

**array-name**

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

When using a list of VARCHAR variables with the ARRAY statement, you can use the hyphen ( - ), colon / prefix, and double-dash lists:
array arr1[*] v1-v5;
array arr2[*] v:;
array arr3[*] v1--v5;

You cannot use VARCHAR character lists specified as _CHARACTER_.

N
  describes the number and arrangement of elements in the array
*
  specifies the maximum length allowed, 536,870,911 characters. When
assigning a character constant to a VARCHAR variable, the character
constant is limited to 32767 bytes.

array myArray[*] varchar(*) a1 a2 a3 ('a','b','c');

Requirement requires a CAS engine libref on the output table

Engine CAS engine only

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

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

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

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

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

For example, if a VARCHAR variable, columnName, is defined as a VARCHAR(10)
variable, this means that it can store up to 10 characters. But because the value “hello”
contains only 5 characters and each character uses one byte, only 5 bytes of memory are
allocated for the variable. A fixed-length CHAR column, on the other hand, takes up the
defined number of bytes regardless of the actual size of the column.

PROC CONTENTS Output for Actual and Defined Length
When PROC CONTENTS is run against a CAS table that contains a VARCHAR
variable, additional length information is displayed in the output. This information is
displayed only for CAS tables that contain VARCHAR variables.
The **Max Bytes Used** column displays the *actual length* of the data entered, or, the actual width of the column based on the values that it contains. The *actual length* of a VARCHAR variable is the storage size of the data entered, or, the actual width of the column based on the data that it contains. The column data in each row is adjusted to fit the width of the data that is in that individual row. If a VARCHAR variable is uninitialized, this column displays the default minimum length of 1.

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

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

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

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

**Examples: Creating VARCHAR Variables**

- **Example 1:** Create a VARCHAR Using the LENGTH Statement
  ``` Sas
  libname mycas cas;
data mycas.roman;
  length vc32 varchar(32);
  do i = 1 to 10;
    vc32 = put(i, ROMAN.);
    output;
  end;
run;
  ```

- **Example 2:** Create a VARCHAR Using the ARRAY Statement
  ``` 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** – use when the sizes of the column data vary considerably but you are reasonably certain they will not exceed a certain width.
- **VARCHAR(*)** – use when the sizes of the column data vary considerably and the column width might exceed any limits you might place on it.

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

Range

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

Setting Missing VARCHAR Variables

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

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

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

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

VARCHAR Support for Implicit and Explicit Data Type Conversion

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

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

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

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

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

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

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

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

*Output 2.1 PROC CONTENTS Output Showing Explicit Data Type Conversion*

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
<th>Bytes</th>
<th>Chars</th>
<th>Max Bytes Used</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>num</td>
<td>Num</td>
<td>8</td>
<td>8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>vc</td>
<td>Varchar</td>
<td>40</td>
<td>40</td>
<td>4</td>
<td></td>
</tr>
</tbody>
</table>

**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 length \((n \times 1)\). \(n\) 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.
VARCHAR and 4 is the largest number of bytes required to store any character in the character set.

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

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

**Restrictions for the VARCHAR Data Type in the CAS Engine**

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

**Table 2.1 Restrictions and Notable Behaviors for the VARCHAR Data Type in the CAS Engine**

<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.</td>
</tr>
<tr>
<td></td>
<td>Of note, this includes the INDEX and SUBSTR functions.</td>
</tr>
<tr>
<td></td>
<td>See “Index CHAR and VARCHAR Character Strings” in <em>SAS Cloud Analytic Services: DATA Step Programming</em> for a related example.</td>
</tr>
<tr>
<td>KEY= on SET and MODIFY</td>
<td>VARCHAR variables are not supported by the KEY= option in either the SET or MODIFY statements.</td>
</tr>
<tr>
<td>statements</td>
<td></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>
<tr>
<td>VARCHAR</td>
<td>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.</td>
</tr>
</tbody>
</table>
Variable Lists

Selecting a character variable range for character variable lists (for example, \texttt{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:

\begin{verbatim}
_CHARACTERR\_var1-\_CHARACTERR\_-\_CHARACTERR\_-\_CHARACTERR\_\_\_var\_\_\_
\end{verbatim}

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.

\begin{verbatim}
length vc varchar(*);
vc = \texttt{"a string that is 32000 bytes long" || "another string that is 32000 bytes long"};
\end{verbatim}

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.

\begin{verbatim}
length vc1 vc2 varchar(*);
vc1 = \texttt{"a string that is 32000 bytes long"};
vc2 = vc1 \texttt{||"another string that is 32000 bytes long"};
\end{verbatim}

---

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:

\begin{verbatim}
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;
\end{verbatim}

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

CAS LIBNAME Statement

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

**Category:** Data Access

**Data type:** CHAR, VARCHAR, and DOUBLE

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

### Syntax

**Form 1:**

```
LIBNAME libref CAS <options>;
```

**Form 2:**

```
LIBNAME libref CAS HOST="controller-host-name"
PORT=network-port-number <options>;
```

**Form 3:**

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

### Required Arguments

**CAS**

specifies the CAS LIBNAME engine.

**CASLIB=caslib-reference-name**

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

### Optional Arguments

**HOST="controller-host-name"**

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

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

**libref**

specifies a valid SAS name that serves as a shortcut name to associate with the tables on the CAS server. The name must conform with SAS naming conventions. It can be up to eight bytes long.
PORT=network-port-number
specifies an integer for the port where the CAS controller listens for client connections. Otherwise, the engine checks for a value in the CASPORT= system option and then in the _CASPORT_ macro variable.

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

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

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

Example
This example starts a session with the CAS statement and saves the UUID in the MyUUID macro 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";
```
```
data mycas.female;
  set sashelp.class;
  where sex='F';
run;
```

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

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

In the example below, the DATA step reads data from a table in a Base engine library and writes data to a CAS table using the CAS engine library.
```
/*Start a CAS session.*/
cas casauto host="cloud.example.com" port=5570;

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

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

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

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

See Also

- CASLIB statement,
- COMPRESS= LIBNAME option

**CASLIB= LIBNAME Option**

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

- **Category:** Data Access
- **Default:** the active caslib

**Syntax**

CASLIB= *caslib-reference-name*

**Required Argument**

*caslib-reference-name*

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

**Details**

**The Active Caslib**

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

**Comparisons**

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

**See Also**

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

Requests that the table to be created in SAS Cloud Analytic Services is compressed.

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

Syntax

COMPRESS=YES | NO

Required Arguments

YES
specifies that the CAS table is compressed.

NO
specifies that the CAS table is not compressed.

Comparisons

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

See Also

“COMPRESS= Data Set Option”

DATALIMIT= LIBNAME Option

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

Category: Data Access
Default: 100M
Restriction: This option affects only Read access.
Tip: This option can prevent you from accidentally transferring a large amount of data from the server to the client.

Syntax

DATALIMIT=integer | integerK | integerM | integerG | ALL

Optional Arguments

integer
specifies the maximum number of bytes to read.
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 Option”
- “READTRANSFERSIZE= Data Set Option”
- “WRITETRANSFERSIZE= LIBNAME Option”
- “WRITETRANSFERSIZE= Data Set Option”

**NCHARMULTIPLIER= LIBNAME Option**

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

**Category:** Data Access

**Defaults:**
- 1 for SBCS environments
- 1.5 for DBCS environments

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

**Syntax**

NCHARMULTIPLIER=\( n \)

**Required Argument**

\( n \)

specifies the multiplication factor to use when transcoding

**Range**

\( 0 < n \leq 4 \)
Comparisons

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

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

See Also

• “CASNCHARMULTIPLIER= System Option”
• “NCHARMULTIPLIER= Data Set Option”

MAXTABLEMEM= LIBNAME Option

Specifies the maximum amount of memory in bytes that each thread should allocate for in-memory blocks before converting to a memory-mapped file.

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 MAXTABLEMEM= session option.
READTRANSFERSIZE= LIBNAME Option

Specifies the maximum number of bytes to use for the transfer buffer size when reading data from the CAS server.

- **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 Option” on page 49
- “DATALIMIT= Data Set Option” on page 58
- “READTRANSFERSIZE= Data Set Option” on page 65
- “WRITETRANSFERSIZE= LIBNAME Option” on page 54
- “WRITETRANSFERSIZE= Data Set Option” on page 68

TAG= LIBNAME Option

Specifies a user-defined tag to use when constructing table names in SAS Cloud Analytic Services.

Category: Data Access

Syntax

TAG=tag-name

Details

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

The TAG= 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 user-supplied name (for example, mycas.mytab). Table names in SAS Cloud Analytic Services can have more than two levels (for example, lev1.lev2.mytab).

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 66

TRANSCODE_FAIL= LIBNAME Option
Specifies how the engine handles transcoding errors.

Valid in: DATA and PROC steps
Category: Data Access
Default: Error

Syntax
TRANSCODE_FAIL=ERROR | WARN | SILENT

Required Arguments
ERROR
   Writes an error message to the SAS log. Processing stops.
WARN
   Writes a warning message to the SAS log. Processing continues.
SILENT
   Transcoding errors are ignored. Processing continues.

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

See Also
“TRANSCODE_FAIL= Data Set Option” on page 67

WRITETRANSFERSIZE= LIBNAME Option
Specifies the maximum number of bytes to use for the transfer buffer size when writing data to the CAS server.

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

- “DATALIMIT= LIBNAME Option” on page 49
- “DATALIMIT= Data Set Option” on page 58
- “READTRANSFERSIZE= LIBNAME Option” on page 52
- “READTRANSFERSIZE= Data Set Option” on page 65
- “WRITETRANSFERSIZE= LIBNAME Option” on page 54
- “WRITETRANSFERSIZE= Data Set Option” on page 68

APPEND= Data Set Option

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

Valid in: DATA step and PROC step

Category: Data Access

Default: NO

Restriction: You can set either the PROMOTE= on page 64 data set option or the APPEND= data set option to YES. An error results if you set both to YES at the same time.

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

Syntax

APPEND=YES | NO | FORCE
**Required Arguments**

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

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

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

---

**CASLIB= Data Set Option**

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

- **Valid in:** DATA step and PROC step
- **Category:** Data Access
- **Default:** the active caslib

**Syntax**

```
CASLIB= caslib-reference-name
```

**Required Argument**

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

**Comparisons**

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

**See Also**

- “CASLIB Statement” on page 69
- “CASLIB= LIBNAME Option” on page 48
- “CASLIB= Session Option” on page 213
- “CASLIB= System Option” on page 228

---

**COMPRESS= Data Set Option**

Requests that the table to be created in SAS Cloud Analytic Services is compressed.

- **Valid in:** DATA step and PROC step
- **Category:** Data Access
- **Default:** NO
Restrictions: When using this option with the CAS server, use YES and NO values only. Use with output data sets only.

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

Syntax

COMPRESS=YES | NO

Required Arguments

YES
  specifies that the CAS table is compressed.

NO
  specifies that the CAS table is not compressed.

Comparisons

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

See Also

“COMPRESS= LIBNAME Option”

COPIES= Data Set Option

Specifies the number of copies for a redundant table.

Valid in: DATA step and PROC step

Category: Data Access

Default: 1

Restrictions: This option applies only to distributed servers. The number of copies is limited to one less than the number of worker nodes in the server.

Syntax

COPIES=integer

Required Argument

integer
  specifies the number of redundant copies for each block.

Details

SAS Cloud Analytic Services organizes data from tables in blocks. Redundant copies of blocks are stored in-memory across nodes. In the event of node failure, a surviving node accesses the data from the redundant block.
A large number of redundant block copies results in greater fault tolerance for node failures because redundant tables are distributed in memory on a greater number of nodes. COPIES=0 results in no fault tolerance in the event of a node failure.

**DATALIMIT= Data Set Option**

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

- **Valid in:** DATA step and PROC step
- **Category:** Data Access
- **Default:** 100M
- **Restriction:** This option affects only Read access.
- **Tip:** This option can prevent you from accidentally transferring a large amount of data from the server to the client.

### Syntax

```
DATALIMIT=integer | integerK | integerM | integerG | ALL
```

### Required 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= LIBNAME Option”
- “READTRANSFERSIZE= LIBNAME Option”
- “READTRANSFERSIZE= Data Set Option”
- “WRITETRANSFERSIZE= LIBNAME Option”
- “WRITETRANSFERSIZE= Data Set Option”
DUPLICATE= Data Set Option
Specifies whether the output table in the SAS Cloud Analytic Services is duplicated on all nodes.

Valid in: DATA step and PROC step
Category: Data Access
Default: NO
Restriction: This option applies only to distributed servers.
Interaction: The value for the COPIES= data set option is ignored if you specify the DUPLICATE= data set option.

Syntax
DUPLICATE=YES | NO

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

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

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

MAXTABLEMEM= Data Set Option
Specifies the maximum amount of memory in bytes that each thread should allocate for in-memory blocks before converting to a memory-mapped file.

Valid in: DATA step and PROC step
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 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 trans-coded to the UTF-8 encoding in order to run in the CAS server.

Valid in: DATA step and PROC step
Category: Data Access
Defaults: 1 for SBCS environments
          1.5 for DBCS environments
Example: NCHARMULTIPLIER=1.5 expands an 8-byte field to 12 bytes

Syntax
NCHARMULTIPLIER=n

Required Argument

n
   specifies the multiplication factor to use when trans-coding

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 Option”

ONDEMAND= Data Set Option

Specifies how to evaluate temporary computed columns.

Valid in: DATA step and PROC step
Category: Data Access
Default: NO
Tip: On-demand execution is recommended when you transfer data from SAS Cloud Analytic Services to the SAS session, such as when the in-memory table is the input data of a DATA step or a procedure.

Syntax

ONDEMAND=YES | NO

Required Arguments

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

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

See Also

“Working with Temporary Columns” on page 35

ORDERBY= Data Set Option

Specifies the variables by which to order the data within a partition.

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.

```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

“PARTITION= Data Set Option”

PARTITION= Data Set Option

Specifies the list of partitioning variables for the output table.

Valid in: DATA step and PROC step

Category: Data Access

Tip: partition=() removes any existing partition on the table.

Syntax

PARTITION=(variable-list)

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 61

PROMOTE= Data Set Option
Requests that the table to be created in SAS Cloud Analytic Services is added with global scope.

Valid in: DATA step and PROC step
Category: Data Access
Default: NO
Restrictions: You can set either PROMOTE= or APPEND= to YES. An error results if you set both to YES at the same time.
   An error results if a global-scope table with the same name already exists. Existing global scope tables must be dropped or deleted before creating a new global table.
Requirement: The caslib target must also have global-scope.
Note: Global scope lets other sessions access the table, subject to access controls.

Syntax
PROMOTE= YES | NO

Required Arguments
YES
specifies that the table is added with global scope.

NO
specifies that the table is added with a session scope.
READTRANSFERSIZE= Data Set Option

Specifies the maximum number of bytes to use for the transfer buffer size when reading 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.

<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>TEMPEXPRESS=</td>
</tr>
</tbody>
</table>

**Syntax**

```
SCRIPT=fileref
```

See Also

“Working with Temporary Columns” on page 35

---

**TAG= Data Set Option**

Specifies the tag from which to construct table names in SAS Cloud Analytic Services.

<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>
</tbody>
</table>

**Syntax**

```
TAG=tag-name
```

See Also

“TAG= LIBNAME Option” on page 53
**TEMPNAMES= Data Set Option**

Lists the names of the temporary computed columns that are added 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.

- **Valid in:** DATA step and PROC step
- **Category:** Data Access
- **Default:** Error

**Syntax**

TRANSCODE_FAIL=ERROR | WARN | SILENT

**Required Arguments**

- **ERROR**
  writes an error message to the SAS log. Processing stops.

- **WARN**
  writes a warning message to the SAS log. Processing continues.

- **SILENT**
  transcoding errors are ignored. Processing continues.

**Comparisons**

The TRANSCODE_FAIL= data set option overrides the TRANSCODE_FAIL= LIBNAME statement option.
WRITETRANSFERSIZE= Data Set Option

Specifies the maximum number of bytes to use for the transfer buffer size when writing data to the CAS server.

- **Valid in:** DATA step and PROC step
- **Category:** Data Access
- **Alias:** WTS
- **Default:** 512K
- **Restriction:** This option only affects Write access.

**Syntax**

```plaintext
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**

- “DATALIMIT= LIBNAME Option” on page 49
- “DATALIMIT= Data Set Option” on page 58
- “READTRANSFERSIZE= LIBNAME Option” on page 52
- “READTRANSFERSIZE= Data Set Option” on page 65
- “WRITETRANSFERSIZE= LIBNAME Option” on page 54
Chapter 3
CASLIB Statement

Dictionary

CASLIB Statement

Adds and manages caslibs in a SAS Cloud Analytic Services session.

Restriction: You can add caslibs only if you are authorized to do so. See Adjust Caslib Management Privileges.

See: For more examples of accessing data, see SAS Cloud Analytic Services: Accessing and Manipulating Data.

For conceptual information about caslibs, see “Caslibs” in SAS Cloud Analytic Services: Fundamentals.

Syntax

Form 1: CASLIB caslib-reference-name <SESSREF=session-reference>
DASOURCE=(SRCTYPE="type" <data-source-options>)
<PATH=directory-path><option(s)>
;

Form 2: CASLIB caslib-reference-name LIST <SESSREF=session-reference>
;

Form 3: CASLIB _ALL_ ASSIGN | _ALL_ LIST

Required Arguments

caslib-reference-name
specifies the name of the caslib. Names of session caslibs must be unique within the session. Names of global caslibs must be unique across all sessions within a server.

Range 1 to 256 characters

Tips Caslib names are case insensitive. If a session-scope caslib and a global-scope caslib have the same name, the session caslib is searched first.
_ALL_ is a valid name with the LIST option and the ASSIGN option.

For naming rules see Names in the SAS Language.

**DATASOURCE**= (SRCTYPE=", <data-source-options>, <ENCRYPTIONPASSWORD=",") specifies data source options to use when connecting to a data source. The SRCTYPE="" 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=","**

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=","**

specifies a password for encrypting or decrypting stored data.

**Restrictions** The ENCRYPTIONPASSWORD= option can be specified only with the DATASOURCE= option.

When you add a path-based caslib that supports the encryptionDomain and encryptionPassword parameters, you can specify one or the other. Use the encryptionDomain parameter to retrieve credentials automatically from an encryption domain that is managed with SAS Environment Manager.

The following table shows the syntax, supported file types (if applicable), and an example for path-based data sources. For the data source options syntax for databases such as Hadoop and Oracle, see Chapter 8, “Data Connectors,” on page 145.

### Table 3.1 Path-Based Data Source Types and Options

<table>
<thead>
<tr>
<th><strong>SRCTYPE= Type</strong></th>
<th><strong>Option Syntax</strong></th>
<th><strong>Example</strong></th>
<th><strong>Supported File Types</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>PATH</strong></td>
<td>DATASOURCE=(SRCTYPE=PATH, ENCRYPTIONDOMAIN=&quot;, PATH=&quot;file-path&quot;);</td>
<td>caslib mycsvs path=&quot;/data/Mycsvfiles/&quot; datasource=(srctype=&quot;path&quot;, encryptiondomain=&quot;string&quot;);</td>
<td>SASHDAT, SAS7BDAT, CSV, XLS, XLSX</td>
</tr>
<tr>
<td><strong>DNFS</strong></td>
<td>DATASOURCE=(SRCTYPE=DNFS, ENCRYPTIONDOMAIN=&quot;, PATH=&quot;file-path&quot;);</td>
<td>caslib mycsvs datasource=(srctype=&quot;dnfs&quot;) path=&quot;/data/Mycsvfiles/&quot;;</td>
<td>SASHDAT, CSV</td>
</tr>
<tr>
<td>Type</td>
<td>Option Syntax</td>
<td>Example</td>
<td>Supported File Types</td>
</tr>
<tr>
<td>-------</td>
<td>---------------</td>
<td>---------</td>
<td>----------------------</td>
</tr>
<tr>
<td>HDFS</td>
<td>DATASOURCE=(SRCTYPE=HDFS, &lt;ENCRYPTIONDOMAIN=&quot;string&quot;&gt;) PATH=&quot;file-path&quot;;</td>
<td>caslib mycsvs path=&quot;/data/Mycsvfiles/&quot; datasource=(srctype=&quot;hdfs&quot;, encryptiondomain=&quot;string&quot;);</td>
<td>SASHDAT CSV</td>
</tr>
</tbody>
</table>

See For information about the PATH= option, see “PATH=directory-path” on page 72.

Examples

"Example 1: Add a Global Caslib" on page 74

"Example 5: Encrypt Tables in a Caslib" on page 82

Optional Arguments

ASSIGN

used with the _ALL_ option to assign SAS librefs for existing caslibs so that they are visible in the SAS Studio Libraries tree.

Restriction The CASLIB _ALL_ ASSIGN statement only assigns a libref for caslibs that follow SAS libref naming rules. For example, if the caslib has more than eight characters, then no libref is assigned for that caslib.

Requirement The ASSIGN option must be used with the _ALL_ option.

Tip When caslibs are dropped, the corresponding SAS library is not automatically cleared.

Examples

The following statement assigns SAS librefs for all existing caslibs:

```
caslib _all_ assign;
```

The following statement assigns SAS librefs for all existing caslibs in the session named MySess:

```
caslib _all_ assign sessref=mysess;
```

_ALL_
specifies that the ASSIGN, or LIST argument applies to all currently added caslibs.

Example "Program 2: List All Caslibs” on page 76

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"
```
DESCRIPTION="description"
  specifies a description of the data source.

Alias  DESC=

GLOBAL
  adds the caslib so that it has global scope. A global-scope caslib can have access controls set so that it is accessible from all sessions and can be a way to share data. Other connections to the server that get their own sessions have access to the caslib, subject to access controls. If you do not specify GLOBAL, the caslib is created with session scope. You must also grant access to the caslib in the CAS Server Monitor. You can add caslibs only if you are authorized to do so. See Adjust Caslib Management Privileges

See  For information about caslibs scope, see “Caslibs Scope” in SAS Cloud Analytic Services: Fundamentals.

LIBREF=
  creates a libref and associates it with a caslib. The libref can then be used to access data in SAS Cloud Analytic Services.

Example  The following CASLIB statement creates the caslib Casdata and binds it to the libref Mycas.
  caslib casdata path='/cas/mycasdata/' libref=mycas;

LIST
  displays caslib names and their specifications. To display all caslibs, specify _ALL_ for the caslib-reference-name.

Tip  Specify caslib-reference-name to list the settings for a single caslib. Specify _ALL_ to list all caslibs and their caslib settings. The specifications displayed by the LIST _ALL_ option are a subset. For full specification information, use the LIST option with a specific caslib specified.

Example  “Example 2: List Caslib Settings” on page 75

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 73
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 4, “CASUTIL Procedure,” on page 87.

What Can I Do with the CASLIB Statement
You can do the following tasks with the CASLIB statement:

- add a caslib with access to files from the data source and access to in-memory tables that are read from the data source.
- specify the options to use when connecting to a data source.
- list the caslibs that are available to your session.
- view the caslib settings for one or more caslibs.

For conceptual information about caslibs, see “Caslibs” in SAS Cloud Analytic Services: Fundamentals.

Caslibs Scope
A caslib can have session scope or global scope. Session-scope caslibs make data available to the session that added the caslib. By default, when you add a caslib with the CASLIB statement, the caslib is session scope. You cannot change the scope of a caslib once it has been added.

Global-scope caslibs make data available to all sessions. By default, your personal caslib is global scope, but restricted to the sessions that you start. You can promote tables to global-scope caslibs only. Global-scope caslibs are useful for data sources that all programmers need to access or in cases when you want to share data with other users. An administrator might restrict your ability to add a global-scope caslib. Use the GLOBAL CASLIB statement option to add a global-scope caslib. You can add caslibs only if you are authorized to do so. See Adjust Caslib Management Privileges

Session-scope caslibs are useful for ad hoc data analysis and in cases where you do not want to share data with other users.

For more information about caslib’s scope, see “Caslibs Scope” in SAS Cloud Analytic Services: Fundamentals.
You can also hide caslibs. A hidden caslib is omitted from most lists of caslibs. Tables in a hidden caslib are omitted from most lists of tables. For more information about hidden caslibs, see Reduced Visibility: Hidden Caslibs.

**The Active Caslib**

When you start a session, your personal caslib is added by default. Initially, it is the active caslib if your server is configured with personal caslibs. If not, then the first defined global caslib is the active caslib. When you use the CASLIB statement to add a caslib, that caslib becomes the active caslib. The active caslib is used as the default data source if you do not override it. You can override the active caslib in the CASUTIL procedure or as a data set option for a CAS engine libref. Because the active caslib is used as a default data source, only one caslib can be active at a time. If you use another CASLIB statement to add a caslib, the previous caslib becomes inactive, and the new caslib becomes active. To add a caslib without making it the active caslib, use the NOTACTIVE CASLIB statement option. You can also set the active caslib with the CASLIB= session option. For information about the CASLIB= session option, see “CASLIB= Session Option” on page 213.

**Troubleshooting**

The following list identifies several messages that you might encounter. In most cases, the messages can be experienced from any programming interface. For messages that are produced in SAS Studio only, an indication is provided.

**You Do Not Have Permission to Create a Caslib**

**ERROR:** You do not have permission to create global caslibs or **ERROR:** You do not have permission to create session caslibs.

You can add caslibs only if you are authorized to do so. See Adjust Caslib Management Privileges.

**Examples**

**Example 1: Add a Global Caslib**

**Program**

The following example adds the caslibs Vapublic and Hadooplib. They each contain connection information to the data sources. After running this program, Hadooplib is the active caslib.

```sas
/*If not already done, create session Casauto.*/
/*Specify a host and port that are valid for your site.*/
/*options cashost="cloud.example.com" casport=5570;*/
/*cas casauto;*/

caslib Vapublic path="/vapublic"
   datasource=(srctype="hdfs") global ;  /*1*/

caslib Hadooplib desc="Hadoop Caslib"
   datasource=(srctype="hadoop",                          /*2*/
               dataTransferMode="parallel",
               hadoopjarpath="Hadoop-jar-file-path",
               hadoopconfigdir="Hadoop-config-files-path",
               username="user-id",
               server="Hadoop-server-hostname",
```
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 “SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop” on page 150.
- You can add caslibs only if you are authorized to do so. See Adjust Caslib Management Privileges

Example 2: List Caslib Settings

You can list the caslib properties for one or more caslibs with the LIST statement.

Program 1: List the Settings for a Specific Caslib

The CASLIB statement with the LIST option and a caslib name displays the caslib settings for a specific caslib.

```sas
  caslib CASUSER list;
  caslib Hadooplib list;
```

SAS Log Showing Data Connector Settings

The settings for uid, server, hadoopConfigDir, hadoopJarPath, and schema only appear when listing the caslib explicitly. The settings that are displayed correspond to data connector options and are specific to the data connector.

**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=** is session scope. **Local=** corresponds to the SESSION option in the CASLIB statement.

Active=Yes | No indicates whether the caslib is the active caslib.

Personal=Yes | No indicates whether the caslib is a personal caslib or not.

When you list a specific data connector caslib, additional information about the connection settings is displayed, as shown in “SAS Log Showing Data Connector Settings” on page 76. The settings that are displayed correspond to data connector options and are specific to the data connector. For the data source options syntax for databases such as Hadoop and Oracle, see Chapter 8, “Data Connectors,” on page 145.

- You can also use the CASUTIL procedure to view information about caslibs. For documentation about the CASUTIL procedure, see Chapter 4, “CASUTIL Procedure,” on page 87.

- For the Hadoop connect string syntax, see “SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop” on page 150.

**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.

```sas
  caslib Myvapublic path="/vapublic"
    datasource=(srctype="hdfs"); /* 1 */
  libname mycas cas caslib=myvapublic; /* 2 */
  data mycas.carsWght; /* 3 */
    set sashelp.cars;
    if weight<5500 then delete;
    keep make model type weight MPG_City;
  run;

  proc casutil incaslib="Myvapublic" outcaslib="Myvapublic"; /* 4 */
    list tables incaslib="Myvapublic";
    save casdata="CarsWght";
  run;
```

1 The first CASLIB statement adds a global-scope caslib named Myvapublic. The **DATASOURCE=** option and the **PATH=** option provide connection information to the Myvapublic directory. Myvapublic is now the active caslib.

2 The **LIBNAME** statement assigns the libref Mycas. To run a DATA step in CAS, you must specify the CAS engine **LIBNAME** statement and use the CAS engine libref with both the input and output table names. The **CASLIB=** option binds the libref to the Myvapublic caslib.

3 This DATA step creates an in-memory table named Mycas.CarsWght in the Myvapublic caslib. There is no on-disk representation and it does not persist in the Myvapublic directory unless you save it.

4 The **CASUTIL** procedure saves the table to Myvapublic. The **INCASLIB=** option specifies the caslib that contains the file, and the **OUTCASLIB=** option specifies the
caslib that the file is being made available to. Use the PROC CASUTIL LIST statement to make sure that the table exists in the Myvapublic caslib.

SAS Log

The note in the SAS log shows that the table was saved.

```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.

```
The CASUTIL Procedure

Caslib Information

<table>
<thead>
<tr>
<th>Library</th>
<th>MYVAPUBLIC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Source Type</td>
<td>HDFS</td>
</tr>
<tr>
<td>Path</td>
<td>Myvapublic/</td>
</tr>
<tr>
<td>Session local</td>
<td>Yes</td>
</tr>
<tr>
<td>Active</td>
<td>Yes</td>
</tr>
<tr>
<td>Personal</td>
<td>No</td>
</tr>
<tr>
<td>Hidden</td>
<td>No</td>
</tr>
</tbody>
</table>

The SAS System

The CASUTIL Procedure

Table Information for Caslib MYVAPUBLIC

<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>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARSWGH</td>
<td>7</td>
<td>5</td>
<td>utf-8</td>
<td>31Jan2017:14:31:55</td>
<td>31Jan2017:14:31:55</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>
```

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 4, “CASUTIL Procedure,” on page 87.
• For more examples of using the CASUTIL procedure to access and save data, see “Accessing Data” in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.

**Example 4: Copy Data from One Data Source to Another**

```plaintext
caslib ldbeta datasource=(srctype="path")
    path="path-to-directory"
    description="imported files";

proc casutil incaslib="LDbeta" outcaslib="hps";
    contents casdata="donations.csv";
    load casdata="donations.csv" casout="donations";
    list tables incaslib="hps";
    save casdata="donations" incaslib="hps";
run;
```

1. The first CASLIB statement adds a session-scope caslib named Ldbeta. The DATASOURCE= option provides connection information to a path-based directory.

2. The CASUTIL procedure loads and saves the table to the data source. The INCASLIB= option specifies the caslib that contains the file, and the OUTCASLIB= option specifies the caslib that the file is being made available to.

3. The CONTENTS statement reads the on-disk file, Donations.csv, and displays the table metadata. This enables you to learn if the file has column names in the first row and the data types.

4. The LOAD CASDATA= statement reads the CSV file into memory and explicitly names the table Donations. The table is now available for analytics.

5. The LIST TABLES statement confirms that the in-memory table named Donations is available in the HPS caslib.

6. The SAVE statement saves the table as a SASHDAT file so that it can be loaded from the HPS caslib in the future rather than imported from the CSV file.
Results

The following results show table metadata displayed by the PROC CASUTIL CONTENTS statement.

Output 3.1  Table Metadata

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>donationid</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>projectid</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>donor_acctid</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>donor_city</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>donor_state</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>donor_zip</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>is_teacher_acct</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>donation_timestamp</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>donation_to_project</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>donation_optional_support</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>donation_total</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>dollar_amount</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>donation_included_optional_support</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>payment_method</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>payment_included_acct_credit</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>payment_included_campaign_gift_card</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>payment_included_web_purchased_gift_card</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>payment_was_promo_matched</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>via_giving_page</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>for_honoree</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>donation_message</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
The following results show the files available in the HPS caslib, displayed by the PROC CASUTIL LIST statement.

**Output 3.2  Partial Display of Files Available in the HPS Caslib**

<table>
<thead>
<tr>
<th>Label</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>NL S encoding</th>
<th>Created</th>
<th>Last Modified</th>
<th>Prompted Table</th>
</tr>
</thead>
<tbody>
<tr>
<td>All Data Types</td>
<td>279979</td>
<td>64</td>
<td>utf-8</td>
<td>18Mar2016:14:14:13</td>
<td>18Mar2016:14:14:13</td>
<td>Yes</td>
</tr>
<tr>
<td>Car Sales</td>
<td>1660</td>
<td>33</td>
<td>latin1</td>
<td>18Mar2016:14:16:34</td>
<td>18Mar2016:14:16:34</td>
<td>Yes</td>
</tr>
<tr>
<td>Car Sales data from sashelp library</td>
<td>428</td>
<td>15</td>
<td>latin1</td>
<td>18Mar2016:14:17:01</td>
<td>18Mar2016:14:17:01</td>
<td>Yes</td>
</tr>
<tr>
<td></td>
<td>10302</td>
<td>36</td>
<td>latin1</td>
<td>18Mar2016:14:17:30</td>
<td>18Mar2016:14:17:30</td>
<td>Yes</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 4, “CASUTIL Procedure,” on page 87.
- For more examples of using the CASUTIL procedure to access and save data, see “Accessing Data” in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.

**Example 5: Encrypt Tables in a Caslib**

```sas
   caslib Encr datasource=(srctype="path", encryptionPassword="your-password") path="your-file-path";
   run;

   libname mycas cas;
   proc casutil;
   load data=sashelp.cars groupby="make" outcaslib="Encr";
   contents casdata="cars";
   list files incaslib="encr";
   run;

   proc mdsummary data=mycas.cars;
   var msrp invoice;
   output out=mycas.mdsumstatEncr;
   groupby make;
   run;
```
options obs=15;
proc print data=mycas.mdsumstatEncr;
  var Make _Column_ _NObs_ _Mean_ _Max_ _Min_ _Std_
  title "Summary of MSRP and Invoice, Grouped by Make";
run;

proc casutil incaslib="Encr" outcaslib="Encr";
  save casdata="mdsumstatEncr";
  list files incaslib="encr"
run;

1 The ENCRYPTIONPASSWORD= option in the CASLIB statement specifies a
password for encrypting or decrypting tables.

2 Assign a CAS engine libref with the LIBNAME statement. To run PROC
MDSUMMARY and PROC PRINT in CAS, you must specify the CAS engine
LIBNAME statement and use the CAS engine libref with both the input and output
table names.

3 The LOAD CASDATA= statement reads the file into memory. The table is now
available for analytics.

4 The CONTENTS statement reads the on-disk file, Cars, and displays the table
metadata. This enables you to determine whether the file has column names in the
first row and the data types.

5 The LIST TABLES statement confirms that the in-memory table named Cars is
available in the Encr caslib.

6 The MDSUMMARY procedure computes the descriptive statistics and groups them
by Make.

7 The PRINT procedure prints the output.

8 The CASUTIL procedure saves the table to the data source. The INCASLIB= option
specifies the caslib that contains the file, and the OUTCASLIB= option specifies the
caslib that the file is being made available to.

9 The SAVE statement stored the data as UTF-8 when it created the SASHDAT file.

10 The LIST FILES statement confirms that the in-memory table named
MdsumstatEncr is saved in the data source that the Encr caslib is associated with.
Results: LIST Statement

The following partial results of the PROC CASUTIL LIST statement show that the caslib is encrypted and a password is specified.

Output 3.3  ENCR Caslib Information

<table>
<thead>
<tr>
<th>Caslib Information</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Library</td>
<td>ENCRT</td>
</tr>
<tr>
<td>Source Type</td>
<td>PATH</td>
</tr>
<tr>
<td>Path</td>
<td></td>
</tr>
<tr>
<td>EncryptionPassword</td>
<td>*****</td>
</tr>
<tr>
<td>Session local</td>
<td>Yes</td>
</tr>
<tr>
<td>Active</td>
<td>Yes</td>
</tr>
<tr>
<td>Personal</td>
<td>No</td>
</tr>
<tr>
<td>Hidden</td>
<td>No</td>
</tr>
</tbody>
</table>
Results: CONTENTS Statement

**Output 3.4** Column Information for the Cars Data Set

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

Results: LIST Statement Information for the Data Source

**Output 3.5** List of Files in the Data Source

<table>
<thead>
<tr>
<th>CAS File Information</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>mdsunstatEncr.sashdat</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 4, “CASUTIL Procedure,” on page 87.

• For more examples of using the CASUTIL procedure to access and save data, see “Accessing Data” in SAS Cloud Analytic Services: Accessing and Manipulating Data.

• For documentation about the MDSUMMARY procedure, see Chapter 5, “MDSUMMARY Procedure,” on page 113.
# Chapter 4

## CASUTIL Procedure

<table>
<thead>
<tr>
<th>Overview: CASUTIL Procedure</th>
<th>88</th>
</tr>
</thead>
<tbody>
<tr>
<td>What Does the CASUTIL Procedure Do?</td>
<td>88</td>
</tr>
<tr>
<td>Terminology</td>
<td>88</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Syntax: CASUTIL Procedure</th>
<th>89</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL Statement</td>
<td>89</td>
</tr>
<tr>
<td>CONTENTS Statement</td>
<td>90</td>
</tr>
<tr>
<td>DELETESOURCE Statement</td>
<td>91</td>
</tr>
<tr>
<td>DROPTABLE Statement</td>
<td>91</td>
</tr>
<tr>
<td>LIST Statement</td>
<td>92</td>
</tr>
<tr>
<td>LOAD Statement</td>
<td>93</td>
</tr>
<tr>
<td>PROMOTE Statement</td>
<td>97</td>
</tr>
<tr>
<td>SAVE Statement</td>
<td>98</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enclose Values in Quotation Marks</th>
<th>100</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subdirectories and Filename Matching</td>
<td>100</td>
</tr>
<tr>
<td>Exclude Subdirectories from the File Information Results</td>
<td>100</td>
</tr>
<tr>
<td>List the Files in a Subdirectory</td>
<td>100</td>
</tr>
<tr>
<td>Using Wildcard Characters for Filename Matching</td>
<td>101</td>
</tr>
<tr>
<td>Loading Files from a Subdirectory</td>
<td>102</td>
</tr>
<tr>
<td>Saving Files to a Subdirectory</td>
<td>102</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Limitations and Restrictions</th>
<th>102</th>
</tr>
</thead>
<tbody>
<tr>
<td>Results: CASUTIL Procedure</td>
<td>103</td>
</tr>
<tr>
<td>Procedure Output</td>
<td>103</td>
</tr>
<tr>
<td>ODS Table Names</td>
<td>104</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Examples: CASUTIL Procedure</th>
<th>105</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1: Load a CSV File into CAS</td>
<td>105</td>
</tr>
<tr>
<td>Example 2: Append Rows to an In-Memory Table</td>
<td>106</td>
</tr>
<tr>
<td>Example 3: Promote a Table</td>
<td>108</td>
</tr>
<tr>
<td>Example 4: Saving and Loading Encrypted SASHDAT Files</td>
<td>109</td>
</tr>
</tbody>
</table>
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.
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: Accessing and Manipulating Data.

Syntax

PROC CASUTIL <option(s)>;

Optional Arguments

INCASLIB=caslib
specifies the input caslib for the procedure. This option does not change the active caslib for your session. If you do not specify this option here or in a statement, such as LOAD, then the active caslib is used.

Specifying the caslib to use is a best practice until you develop experience working with the active caslib with this procedure, the CAS LIBNAME engine, and other procedures.

OUTCASLIB=caslib
specifies the output caslib for the procedure. This option does not change the active caslib for your session. If you do not specify this option here or in a statement, such as LOAD, then the active caslib is used.

SESSREF=session-name
specifies the session to use with the procedure. If you omit SESSREF=, then procedure uses the session that specified in the(&_SESSREF_) macro variable.

Alias SESSION=
CONTENTS Statement

The CONTENTS statement displays table metadata such as column names and data types for files or in-memory tables.

Examples:

"Example 3: Promote a Table" on page 108
"Example 4: Saving and Loading Encrypted SASHDAT Files" on page 109

Syntax

```
CONTENTS CASDATA="table-name" <INCASLIB="caslib"> <option(s)>;
```

Required Argument

```
CASDATA="table-name"
```

specifies the name of the file or table.

Optional Arguments

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

specifies the file format and options. Specify this option only if you specify a filename in the CASDATA= option. If you want to display metadata for an in-memory table (the result of a LOAD statement), then do not specify this option.

For information about file-type and file-type-options, see "IMPORTOPTIONS=(FILETYPE="file-type" file-type-options)" on page 95.

```
INCASLIB="caslib"
```

specifies the caslib that is associated with the file or table. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

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

specifies overrides for the DATASOURCE= options for the caslib. For more information, see Chapter 6, “Platform Data Sources,” on page 131 and Chapter 8, “Data Connectors,” on page 145.

Alias OPTIONS=

```
ROWCOUNT
```

specifies to include the number of rows in the results. The data source for the input caslib must be HDFS and you must include the filename suffix in the CASDATA= option.

```
VARS=((casinvardesc-1) < (casinvardesc-2) ...)
```

specifies the variables to include. If you do not include this option, all variables are included.

The value for casinvardesc is described in “VARS=((casinvardesc-1) (casinvardesc-2) ...)” on page 96.

Alias VARLIST=

Example: Viewing Column Names from a CSV File

```
contents casdata="somefile.csv" importoptions=(filetype="csv")
```
DELETESOURCE Statement

The DELETESOURCE statement removes a file from the data source that is associated with a caslib. You do not need to drop an in-memory table with the same name before using this statement.

**Note:** You can delete files from path-based caslibs. These are caslibs with a data source type of DNFS, HDFS, or PATH.

**Syntax**

```
DELETESOURCE CASDATA="file-name" <INCASLIB="caslib"> <QUIET>;
```

**Required Argument**

CASDATA="file-name"

specifies the name of the file to remove.

**Optional Arguments**

INCASLIB="caslib"

specifies the caslib that is associated with the file to remove. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

QUIET

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

DROPTABLE Statement

The DROPTABLE statement removes a table from memory on SAS Cloud Analytic Services.

**Example:**

“Example 4: Saving and Loading Encrypted SASDAT Files” on page 109

**Syntax**

```
DROPTABLE CASDATA="table-name" <INCASLIB="caslib"> <QUIET>;
```

**Required Argument**

CASDATA="table-name"

specifies the name of the table to remove from memory.

**Optional Arguments**

INCASLIB="caslib"

specifies the caslib that is associated with the table to remove. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

QUIET

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

The LIST statement lists files from a caslib’s data source or in-memory tables in a caslib.

Example:  “Example 1: Load a CSV File into CAS” on page 105

Syntax

LIST <FILES | TABLES> <option(s)>;

Required Argument

FILES | TABLES
specifies whether to list the files from a caslib's data source or in-memory tables.

FILES
lists the files that are available in the caslib's data source.

TABLES
lists the in-memory tables in a caslib.

Default TABLES

Optional Arguments

INCASLIB="caslib"
specifies the caslib that is associated with the tables or files to list. If specified, this option overrides the INCASLIB= value in the procedure statement or the active caslib.

NOSUBDIRS
specifies to exclude subdirectories from the results.

Applies to LIST FILES and path-based caslibs

DATASOURCEOPTIONS=(data-source-options)
specifies overrides for the DATASOURCE= options for the caslib. For more information, see Chapter 6, “Platform Data Sources,” on page 131 and Chapter 8, “Data Connectors,” on page 145.

Alias OPTIONS=

Applies to LIST FILES

ROWCOUNT
specifies to include the number of rows in the results. The data source for the input caslib must be HDFS.

Applies to LIST FILES

SUBDIR="path"
specifies to list the files in the specified subdirectory.

Applies to LIST FILES and path-based caslibs
LOAD Statement

The LOAD statement reads data from a file in a caslib’s data source, a libref, or a client-side file and loads it into memory on SAS Cloud Analytic Services.

Syntax

Form 1:  LOAD CASDATA="file-name" <INCASLIB="caslib"> CASOUT="table-name"
  <OUTCASLIB="caslib" >
  <IMPORTOPTIONS=(FILETYPE="file-type" <file-type-options>)>
  <GROUPBY=(group-by-variable-1 <group-by-variable-2 ...)>>
  <ORDERBY=(variable-1 <variable-2 ...)>>
  <LABEL="table-description">
  <DATASOURCEOPTIONS=(data-source-options)>
  <PROMOTE | REPLACE>
  <COPIES=integer>
  <VARS=((casinvardesc-1) <, (casinvardesc-2), ...>)>
  <WHERE="where-expression-1 <logical-operator where-expression-2>" ;

Form 2:  LOAD DATA=SAS-data-set(data-set-options)>
  <CASOUT="table-name">
  <OUTCASLIB="caslib">
  <APPEND | COMPRESS>
  <GROUPBY=(group-by-variable-1 <group-by-variable-2 ...)>>
  <ORDERBY=(variable-1 <variable-2 ...)>>
  <LABEL="table-description">
  <DATASOURCEOPTIONS=(data-source-options)>
  <PROMOTE | REPLACE>
  <REPEAT>
  <COPIES=integer>;

Form 3:  LOAD FILE="SAS-file" CASOUT="table-name"
  <OUTCASLIB="caslib">
  <COMPRESS>
  <IMPORTOPTIONS=(FILETYPE=file-type <file-type-options>)>
  <LABEL="table-description">
  <COPIES=integer>;

Required Arguments

CASDATA="file-name"
  specifies the name of the file to load from the server-side data source that is associated with the INCASLIB= option or the active caslib.

Requirement You must specify CASOUT=.

CASOUT="table-name"
  specifies the name to use for the in-memory table.

Interaction This argument is required when you use the LOAD CASDATA= or LOAD FILE= forms.
Note This argument does not follow or enforce SAS naming rules such as the name literal syntax.

Tip Some data sources support table names that exceed 32 bytes. Use this option to limit table names to 32 bytes so that you can access the table with the CAS LIBNAME engine.

DATA=SAS-data-set
specifies the libref and data set name to use.

FILE="SAS-file"
specifies an external file that is accessible to the SAS client host. Use this option to upload a file to the server and import the data. Do not use this option to import a SAS data set, use the DATA= option.

Requirement You must specify CASOUT=.

**Optional Arguments**

**APPEND**
adds the rows from the SAS data set in the DATA= argument to the end of an in-memory table. This option is supported with the DATA= argument only.

**CASNCHARMULTIPLIER=n**

specifies a number that is used as a multiplication factor to control the byte size for fixed character data.

Range $0 < n \leq 4$

Defaults For SBCS environments, the default is 1.

For DBCS environments, the default is 1.5.

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

**COPIES=integer**
specifies the number of replicate copies of the rows to make for fault tolerance. Larger values use more memory and can result in slower performance, but provide high availability for data in the event of a node failure.

Alias REPLICATION=

Default 1

Interactions This option is ignored if the REPEAT option is also specified.

This option is ignored if the data source is a data connector or data connect accelerator. The SAS Data Connector to Oracle and SAS Data Connector to Hadoop are examples of these products.

**DATASOURCEOPTIONS=(data-source-options)**
specifies overrides for the DATASOURCE= options for the input caslib. For more information, see Chapter 6, “Platform Data Sources,” on page 131 and Chapter 8, “Data Connectors,” on page 145.
Alias OPTIONS=

GROUPBY=(group-by-variable-1 <group-by-variable-2...>)
specifies the names of the variables to use for grouping results.

Alias PARTITIONBY=

IMPORTOPTIONS=(FILETYPE="file-type" <file-type-options>)
specifies the file format and options.

FILETYPE="AUTO" | "BASESAS" | "CSV" | "DTA" | "EXCEL" | "HDAT" | "LASR" | "XLS"
specifies the file format. AUTO attempts to determine the file type based on the
filename suffix, such as .sashdat, .csv, and so on. Files with a .txt suffix are
imported as a CSV file.

Default AUTO

file-type-options
specifies options for importing the data. For more information, see Chapter 7,
“Platform File Types,” on page 139 and Chapter 8, “Data Connectors,” on page
145.

INCASLIB="caslib"
specifies the caslib that is associated with the file to load. If specified, this option
overrides the INCASLIB= value in the procedure statement or the active caslib.

LABEL="string"
specifies a descriptive label for the table. The label can be up to 256 characters. If the
label text contains single quotation marks, enclose the label in double quotation
marks. To remove a label from a table, assign a blank space that is enclosed in
quotation marks.

NCHARMULTIPLIER=n

n
specifies the multiplication factor used when transcoding.

Range 0 < n ≤ 4

Defaults For SBCS environments the default is 1
For DBCS environments, the default is 1.5.

ORDERBY=(variable-1 <variable-2...)
specifies the variables to use for ordering observations within partitions. This
parameter applies to partitioned tables.

OUTCASLIB="caslib"
specifies an alternative caslib to use for the in-memory table. If specified, this option
overrides the OUTCASLIB= value in the procedure statement or the active caslib.

PROMOTE
specifies to load the table with global scope. This makes the table available to all
sessions that use the caslib, subject to access controls. The caslib must also have
global scope.

REPEAT
specifies to duplicate the rows for the table on every machine of a distributed server.
Making duplicate copies of tables can be useful in cases like a dimension table that is
used in a join. This option is supported with the DATA= argument only and cannot be combined with the GROUPBY= option.

**REPLACE**

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

**VARS=((casinvardesc-1) < (casinvardesc-2) ...>)**

specifies the variables to load into the table. If you do not specify this option, then all variables are loaded into the table.

The value can be one or more of the following:

- **FORMAT="string"** specifies the format to apply to the variable.
- **FORMATTEDLENGTH=integer** specifies the format field length plus the format precision length.
- **LABEL="string"** specifies the descriptive label for the variable.
- **NAME="string"** specifies the name for the variable.
- **NFD=integer** specifies the format precision length.
- **NFL=integer** specifies the format field length.

**Alias**

**VARLIST=**

**WHERE="where-expression-1 <logical-operatorwhere-expression-2> "** specifies conditions for selecting observations from the data.

- **where-expression** is an arithmetic or logical expression that consists of a sequence of operators, operands, and SAS functions. An operand is a variable, a SAS function, or a constant. An operator is a symbol that requests a comparison, logical operation, or arithmetic calculation. The expression must be enclosed in single or double quotation marks.

- **logical-operator** can be AND, AND NOT, OR, or OR NOT.

**Details**

**Summary of Options**

*Table 4.1  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>CASNCHARMULTIPLIER</td>
<td>CASNCHARMULTIPLIER=</td>
</tr>
<tr>
<td>COMPRESS</td>
<td>LOAD DATA=</td>
</tr>
<tr>
<td>LOAD Statement Option</td>
<td>LOAD Statement Form</td>
</tr>
<tr>
<td>------------------------</td>
<td>---------------------</td>
</tr>
<tr>
<td>COPIES=</td>
<td>All LOAD statement forms. For more information, see “Data Redundancy” on page 131.</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>NCHARMULTIPLIER=</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>VARS=</td>
<td>LOAD CASDATA=</td>
</tr>
<tr>
<td>WHERE=</td>
<td>LOAD CASDATA=</td>
</tr>
<tr>
<td></td>
<td>Note: You can specify WHERE= as a data set option when you use the LOAD DATA= form.</td>
</tr>
</tbody>
</table>

**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 108

**Syntax**

PROMOTE CASDATA="table-name" <INCASLIB="caslib">

PROMOTE Statement
<CASOUT="table-name">
<OUTCASLIB="caslib">
<DROP | KEEP>;

**Required Argument**

CASDATA="table-name"

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

**Optional Arguments**

CASOUT="table-name"

specifies the name to use for the promoted table.

Note  This argument does not follow or enforce SAS naming rules such as the name literal syntax.

Tip  Some data sources support table names that exceed 32 bytes. Use this option to limit table names to 32 bytes so that you can access the table with the CASLIBNAME engine.

DROP

specifies to drop the session-scope table after promoting it to global scope, which is the default behavior and a best practice.

INCASLIB="caslib"

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

KEEP

specifies to keep the session-scope table after promoting it to global scope. This workflow is uncommon. The table precedence rules are that the session-local table is accessed before a global-scope table is accessed.

OUTCASLIB="caslib"

specifies an alternative caslib to use for the promoted table. If specified, this option overrides the OUTCASLIB= value in the procedure statement or the active caslib.

---

**SAVE Statement**

The SAVE statement creates a permanent copy of an in-memory table. The in-memory table is saved to the data source that is associated with the caslib.

Note:  You can save an in-memory table to a file with path-based caslibs. These are caslibs with a data source type of DNFS, HDFS, or PATH.

Example:  “Example 4: Saving and Loading Encrypted SASHDAT Files” on page 109

**Syntax**

SAVE CASDATA="file-name" <INCASLIB="caslib">
<CASOUT="table-name"> <OUTCASLIB="caslib">
<option(s)>;

---
**Required Argument**

CASDATA="table-name"

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

**Optional Arguments**

CASOUT="file-name"

specifies an alternative name for the file. A file is created in the data source that is associated with the caslib from the OUTCASLIB= option. By default, a .sashdat file suffix is added. If you specify a .csv suffix, a CSV file is saved.

---

**Note**

This argument does not follow or enforce SAS naming rules such as the name literal syntax.

**Tip**

Some data sources support table names that exceed 32 bytes. Use this option to limit table names to 32 bytes so that you can access the table with the CAS LIBNAME engine when you load it again.

INCASLIB="caslib"

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

OUTCASLIB="caslib"

specifies an alternative caslib to use for the saved table. If specified, this option overrides the OUTCASLIB= value in the procedure statement or the active caslib.

**SAVE Statement Options**

COMPRESS

specifies to compress the data in the saved file.

COPIES=integer

specifies the number of replicate copies of the rows to make for fault tolerance. This value is ignored unless the output caslib data source is HDFS and you save to the SASHDAT file format.

Alias   REPLICATION=

Default  1

GROUPBY=(group-by-variable-1 <group-by-variable-2...>)

specifies the names of the variables to use for partitioning the SASHDAT file.

Alias   PARTITIONBY=

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

specifies the input file format and options. Specify this option only if you want to read a file from the input caslib's data source and save it without loading it to memory. For more information, see IMPORTOPTIONS= on page 95 for the LOAD statement.

DATASOURCEOPTIONS=(data-source-options)

specifies overrides for the DATASOURCE= options for the caslib. For more information, see Chapter 6, “Platform Data Sources,” on page 131 and Chapter 8, “Data Connectors,” on page 145.

Alias   OPTIONS=
ORDERBY=(variable-1 <variable-2…>)

specifies the variables to use for ordering observations within partitions. This parameter applies to partitioned tables.

REPLACE

specifies that a new file with a given name replaces an existing file with the same name.

WHERE="where-expression-1 <logical-operatorwhere-expression-2> "

The specification for this option is described in WHERE= on page 96 for the LOAD statement.

Enclose Values in Quotation Marks

When you specify a value for one of the following items, you can enclose the value in quotation marks:

- INCASLIB="caslib"
- OUTCASLIB="caslib"
- CASDATA="table-name"
- CASOUT="table-name"

For caslibs that use a case-sensitive file system or database as a data source, you control the case that is used.

If you do not enclose the INCASLIB= and OUTCASLIB= values in quotation marks, then SAS naming rules apply.

Subdirectories and Filename Matching

Exclude Subdirectories from the File Information Results

If a path-based caslib has many subdirectories, the results of the LIST FILES statement can be long and can obscure the list of files that can be loaded as data. The NOSUBDIRS option is available to exclude the subdirectories from the results.

    cas casauto setsessopt=(caslib='casuser');
    proc casutil;
    list files nosubdirs;
    quit;

List the Files in a Subdirectory

This topic is the opposite of the preceding topic. You might have files in a subdirectory of a caslib's data source root that you want to use as data. For example, if your CASUSER personal caslib is assigned to /home/sasdemo and you have data in /user/sasdemo/mydata, you cannot assign a caslib to the /home/sasdemo/mydata path.
To list the files in `/home/sasdemo/mydata`, you can use the SUBDIR= option in the LIST FILES statement.

```sas
cas casauto setsessopt=(caslib='casuser');
proc casutil;
  list files subdir='mydata';
quit;
```

### The CASUTIL Procedure

<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>iris.csv</td>
<td>-rwXr-Xr-x</td>
<td></td>
<td></td>
<td>3.1KB</td>
<td>08Aug2016:19:38:59</td>
</tr>
<tr>
<td>titanic3.xls</td>
<td>-rwXr-Xr-x</td>
<td></td>
<td></td>
<td>277.5KB</td>
<td>09Aug2016:09:01:48</td>
</tr>
<tr>
<td>jta_free_wifi.csv</td>
<td>-rwXr-Xr-x</td>
<td></td>
<td></td>
<td>19.1MB</td>
<td>09Aug2016:09:03:39</td>
</tr>
</tbody>
</table>

### Using Wildcard Characters for Filename Matching

Just as the NOSUBDIRS option can help limit the results of the LIST FILES statement, you can use wildcard characters to limit results. The following example lists the files with a CSV suffix.

```sas
cas casauto setsessopt=(caslib='casuser');
proc casutil;
  list files subdir='mydata/%.csv';
quit;
```

**Tip** When you specify a wildcard pattern, the results include the matches and any subdirectories (matching or otherwise). To exclude subdirectories, you can specify the NOSUBDIRS option.

- Filename matching is case sensitive. The `%csv` pattern does not match files with an uppercase or mixed case suffix.
- `%` matches any number of characters or numbers.
- `_` matches a single character or number. You can include more than one underscore in a pattern.
- `\` escapes a wildcard character so that it is treated as a literal character instead of a wildcard in a pattern.

The SAS Macro language also uses the `%` character to indicate macro names to resolve during program compilation. When you use the `%` character for pattern matching, include the value in single quotations marks. If you enclose a value in double quotation marks, the SAS Macro parser tries to resolve a macro name. For example, the code `SUBDIR="mydata/%s.csv"` results in an attempt to resolve the %S macro name, with the message `WARNING: Apparent invocation of macro %S not resolved`. Instead, specify the code as `SUBDIR='mydata/%s.csv'`.
**Loading Files from a Subdirectory**

After you list the files in a caslib's subdirectory, you can load the file into the server with the LOAD CASDATA= statement.

```plaintext
cas casauto setsessopt=(caslib="casuser");

proc casutil;
    load casdata="mydata/iris.csv" casout="iris";
quit;
```

**Saving Files to a Subdirectory**

By default, when you use the SAVE statement, a file is created in the directory that is associated with your active caslib, or the OUTCASLIB= option. To save a file in a subdirectory, include the path in the CASOUT= option.

```plaintext
cas casauto setsessopt=(caslib="casuser");

proc casutil;
    load casdata="mydata/iris.csv" casout="iris";
    save casdata="iris"
        where='species eq 'Setosa''
        casout="mydata/setosa.csv"
        replace;

    list files subdir="mydata";
quit;
```

---

### The CASUTIL Procedure

<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>iris.csv</td>
<td>-rwxr-xr-x</td>
<td></td>
<td></td>
<td>3.1KB</td>
<td>08Aug2016 19:33:59</td>
</tr>
<tr>
<td>titanic3.xls</td>
<td>-rwxr-xr-x</td>
<td></td>
<td></td>
<td>277.5KB</td>
<td>09Aug2016 09:01:48</td>
</tr>
<tr>
<td>jta_free_wifi.csv</td>
<td>-rwxr-xr-x</td>
<td></td>
<td></td>
<td>19.1MB</td>
<td>09Aug2016 09:03:39</td>
</tr>
<tr>
<td>setosa.csv</td>
<td>-rwxr-xr-x</td>
<td></td>
<td></td>
<td>0.9KB</td>
<td>09Aug2016 10:00:23</td>
</tr>
</tbody>
</table>

---

### Limitations and Restrictions

When working with a SAS library, the engine for the library is restricted to the Base engine. This enables you to work with SAS data sets, but using any other engine, such as a SAS/ACCESS engine, is not supported.

When working with a caslib, the SAVE statement is restricted to caslibs with a data source that supports saving tables as SASHDAT files. These caslib data source types are as follows:
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.

```plaintext
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 filename and the Source Caslib field indicates the original caslib.

For the table details information, see these definitions:

**Node**
This field always reports ALL. This procedure provides a summary of the table information for all machines in a distributed server.
Number of Blocks
   The server organizes rows in blocks. For distributed servers, this column shows the sum of the active blocks and any copies of blocks that provide redundancy.

Active Blocks
   The server reads rows from active blocks.

Fixed Data size
   This field shows the number of bytes that are used for numeric columns and fixed-width character columns.

Variable Data size
   This field shows the number of bytes that are used for variable-width character columns.

Blocks Mapped
   This field shows the number of blocks are currently mapped into memory.

Memory Mapped
   This field shows the number of bytes for the blocks that are mapped.

Blocks Unmapped
   This field shows the number of blocks that the server can map into memory. The blocks are mapped into memory when the server handles a request for data from the table. For distributed servers, the redundant blocks that enable fault tolerance are included in this value.

Memory Unmapped
   This field shows the number of bytes for the blocks that the server can map into memory.

Blocks Allocated
   This field shows the number of blocks that do not have an on-disk representation. The blocks can become cached under the following conditions:
   • when you promote a session-scope table to global-scope.
   • you set the MAXTABLEMEM= CAS session option to a lower value. If you append rows and cross the value, the server caches the blocks.

Memory Allocated
   This field shows the number of bytes for the blocks.

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 directory is 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 4.2 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 /* 1 */
  path="/data/csv/
  desc="Spreadsheets and CSV source data.";

  proc casutil;
    list files;
    load casdata="County_Population.csv" /* 2 */
      importoptions=(filetype="csv" getnames="true")
      casout="county_population";
    list tables;
  quit;
```

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

2 The IMPORTOPTIONS= option is used to specify the file type and options for reading the data into the server.
Results: LIST FILES Statement for the CSVFILES Caslib

The following display shows the results of the LIST FILES statement. It is a listing of the files that the server can access from the /data/csv directory.

<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>-rw-------</td>
<td>nobody</td>
<td>everybody</td>
<td>663.5KB</td>
<td>08Nov2012:07:21:12</td>
</tr>
<tr>
<td>mailorder.csv</td>
<td>-rw-------</td>
<td>nobody</td>
<td>everybody</td>
<td>117.2KB</td>
<td>29Sep2015:11:40:14</td>
</tr>
<tr>
<td>NST_EST2012_ALLDATA.csv</td>
<td>-rw-------</td>
<td>nobody</td>
<td>everybody</td>
<td>20.0KB</td>
<td>15Jan2013:06:59:28</td>
</tr>
<tr>
<td>sinc.csv</td>
<td>-rw-------</td>
<td>nobody</td>
<td>nobody</td>
<td>309.3KB</td>
<td>15Jan2013:13:18:14</td>
</tr>
<tr>
<td>SC-EST2011-6RACE-ALL.csv</td>
<td>-rw-------</td>
<td>nobody</td>
<td>everybody</td>
<td>138.3KB</td>
<td>15Jan2013:09:02:53</td>
</tr>
<tr>
<td>sinc10k.csv</td>
<td>-rw-------</td>
<td>nobody</td>
<td>nobody</td>
<td>582.9KB</td>
<td>09May2013:07:16:16</td>
</tr>
<tr>
<td>UNdata_Export_20130115_092219549.csv</td>
<td>-rw-------</td>
<td>nobody</td>
<td>everybody</td>
<td>595.4KB</td>
<td>15Jan2013:09:22:20</td>
</tr>
<tr>
<td>2012_nll_pbp_data_reg_season1.csv</td>
<td>-rw-------</td>
<td>nobody</td>
<td>everybody</td>
<td>6.5MB</td>
<td>06Feb2013:11:19:02</td>
</tr>
<tr>
<td>County_Population.csv</td>
<td>-rw-------</td>
<td>nobody</td>
<td>nobody</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>Name</th>
<th>Libary</th>
<th>source Type</th>
<th>Description</th>
<th>Path</th>
<th>Session local</th>
<th>Active</th>
<th>Personal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Country_Population</td>
<td>CSVFILES</td>
<td>DSN</td>
<td>*spreadsheets and CSV source data *</td>
<td><em>State</em></td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
</tbody>
</table>

Example 2: Append Rows to an In-Memory Table

Program

```plaintext
proc casutil;
  load data=sashelp.cars(where=(make="Buick");
  casout="some_cars"
  label="Some makes from the Sashelp.Cars sample data.");
  promote;

  /* add rows for a few more makes */
```
Example 2: Append Rows to an In-Memory Table

load data=sashelp.cars(where=(make in ('Ford', 'Chrysler'))) casout="some_cars" append; /* 2 */
list tables; quit;
libname mycas cas;
proc cardinality data=mycas.some_cars outcard=mycas.cars_cardinality;
   vars enginesize mpg_highway mpg_city; /* 3 */
run;
proc casutil;
   contents casdata="cars_cardinality"; /* 4 */
run;
proc print data=mycas.cars_cardinality;
   var _varname_ _cardinality_ _nobs_ _nmiss_ _min--_kurtosis_;
run;

1 The first LOAD DATA= statement subsets the Sashelp.Cars data set based on the Make variable. The CASOUT= option specifies the name for the output table, Some_cars. The PROMOTE option sets the output table as a global-scope table.

2 The second LOAD statement uses the APPEND option to append more rows from the Sashelp.Cars data set.

3 The CARDINALITY procedure is used to calculate summary statistics for three variables. The OUTCARD= option specifies an in-memory table to use for storing the summary data.

4 The CONTENTS statement is used to display the table information and column information for the Cars_cardinality table. Notice that the libref is not included with the CASDATA= option. The results of the statement include the column names. Some of the column names are specified in the PRINT procedure.

Results: LIST TABLES Statement for the CASUSER Caslib

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Label</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>Compressed</th>
</tr>
</thead>
</table>
Results: CONTENTS Statement for the Cars_Cardinality Table

The CASUTIL Procedure

<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>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARS_CARDINALITY</td>
<td>3</td>
<td>26</td>
<td>utf-8</td>
<td>08Apr2016-13:29:19</td>
<td>08Apr2016-13:29:19</td>
<td>No</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
</tbody>
</table>

Results: Select Columns from the Cars/Cardinality Table

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>VARNAME</em></th>
<th><em>CARDINALITY</em></th>
<th><em>NOBS</em></th>
<th><em>NMISS</em></th>
<th><em>MIN</em></th>
<th><em>MAX</em></th>
<th><em>MEAN</em></th>
<th><em>STDDEV</em></th>
<th><em>SKEWNESS</em></th>
<th><em>KURTOSIS</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>EngineSize</td>
<td>17</td>
<td>47</td>
<td>0</td>
<td>2</td>
<td>6.8</td>
<td>3.3957446609</td>
<td>1.0133795046</td>
<td>0.777671628</td>
<td>1.347538265</td>
</tr>
<tr>
<td>2</td>
<td>MPG_Highway</td>
<td>16</td>
<td>47</td>
<td>0</td>
<td>13</td>
<td>36</td>
<td>26.6389297872</td>
<td>4.3413554</td>
<td>-0.725567797</td>
<td>1.398265519</td>
</tr>
<tr>
<td>3</td>
<td>MPG_City</td>
<td>12</td>
<td>47</td>
<td>0</td>
<td>10</td>
<td>27</td>
<td>19.382978723</td>
<td>3.4550101711</td>
<td>0.192445364</td>
<td>0.0283789567</td>
</tr>
</tbody>
</table>

Example 3: Promote a Table

Program

```r
            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;
            /* 2 */
            * The purpose for outcaslib= is to show how to
            * work with more than one caslib.
            */
            proc casutil outcaslib="hps"; /* 3 */
```
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 97 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.
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.

The CSV file uses the Shift JIS encoding. If a file does not use UTF-8 or 7-bit ASCII, then specify the encoding.

The CONTENTS statement on page 90 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 111.

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, 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 4.1 on page 111.

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>NILS 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>42250</td>
<td>21</td>
<td>utf-8</td>
<td>08Apr2016:13:56:01</td>
<td>08Apr2016:13:56:01</td>
<td>No</td>
<td>No</td>
<td>No</td>
<td>jta_free_wifi.csv</td>
<td>CASUSER(unknown)</td>
<td>No</td>
</tr>
</tbody>
</table>

Results: WiFi Hotspot Categories and Counts

Output 4.1 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（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>
Chapter 5
MDSUMMARY Procedure

Overview: MDSUMMARY Procedure

What Does the MDSUMMARY Procedure Do?

The MDSUMMARY procedure computes basic descriptive statistics for variables across all observations or within groups of observations in parallel for data tables stored in SAS Cloud Analytic Services (CAS). The MDSUMMARY procedure uses CAS tables and capabilities, ensuring full use of parallel processing.

Syntax: MDSUMMARY Procedure

Restriction: You cannot use a FORMAT statement in a PROC MDSUMMARY step. If you need to format the input table, apply the formats to the table when it is created. For examples, see “Example 4: Using Formats with Group-By Variables” on page 125.

PROC MDSUMMARY DATA=libref.table-name <NTHREADS=integer>;
   VAR <variable-list>;
   OUTPUT OUT=table-name;
   GROUPBY variable-list </OUT=table-name>;
RUN;

PROC MDSUMMARY Statement
Calculates multidimensional summaries of numeric variables.

Syntax
PROC MDSUMMARY DATA=libref.table-name< NTHREADS=integer> ;

Optional Arguments
DATA=libref.table-name
   specifies the two-level input table name.

 Requirement  The table name must be a two-level name where Libref is a CAS engine libref.

 Alias  TABLE=

NTHREADS=integer
   specifies the number of threads to use within each compute node.

VAR Statement
Specifies the analysis variables and their order in the output.

Syntax
VAR <variable-list>

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

Optional Argument
<variable(s)>
   identifies one or more analysis variables and specifies their order in the results.
GROUPBY Statement

Creates BY groups in terms of the variable value combinations given the variables in the variable list.

**Requirement:**
You must specify either the OUTPUT statement, or at least one GROUPBY statement with the OUT= option specified. You cannot specify both the OUTPUT= statement and a GROUPBY statement with the OUT= option specified.

**Tips:**
Multiple GROUPBY statements can be specified, each having its own output table. If a variable value is missing, PROC MDSUMMARY includes the observations and rows in the analysis.

**Syntax**

```
GROUPBY <variable(s)> < OUT=table-name>;
```

**Without Arguments**
If no variables are listed, then the statistics are calculated across all input observations

**Optional Arguments**

**OUT=table-name**
specifies the table name.

**Requirement**
The table name must be a two-level name where *Libref* is a CAS engine libref.

**variable(s)**
specifies the analysis variables to group by.

OUTPUT Statement

Creates an output table that contains the results of PROC MDSUMMARY.

**Restriction:**
You can specify one OUTPUT statement only.

**Requirement:**
You must specify either the OUTPUT statement or at least one GROUPBY statement that specifies the OUT= option. You cannot specify the OUTPUT= statement with one or more GROUPBY statements that also specify the OUT= option.

**Syntax**

```
OUTPUT OUT=libref.table name;
```

**Required Argument**

**OUT=libref.table name**
specifies the two-level table name.
Requirement: The table name must be a two-level name where Libref is a CAS engine libref.

PROC MDSUMMARY Output Data Sets

You can create output tables by using one OUTPUT statement or multiple GROUPBY statements with the OUT= option specified. You must specify either the OUTPUT statement, or one or more GROUPBY statements with the OUT= option specified. You cannot specify both an OUTPUT statement and a GROUPBY statement with the OUT= option specified. To produce multiple output tables, use multiple GROUPBY OUT= statements.

PROC MDSUMMARY does not display output. You can use PROC PRINT to display the output table.

Results: MDSUMMARY Procedure

Output Tables

PROC MDSUMMARY produces one or more output tables for each By group, defined by a set of variables listed in a GROUPBY statement.

A PROC MDSUMMARY table contains the following:

• One column for each basic statistic and one row for each combination of group-by level and analysis variable.

• If you are creating By groups, then two columns for each group-by variable are also included in the output table. One column is for the group-by variable itself. The other column is of a character type and has the same name as the group-by variable but with _f appended. The column contains the formatted value of the group-by variable.

• A column named _Column_, containing the name of the analysis variable, is included in the output. The _Column_ column denotes the variable for which statistics have been computed.

Group-by processing collects observations for analysis according to the formatted values of the group-by variables, with each unique combination of formatted group-by variable values forming one group-by level. Groups are not collected or processed in any particular order.

The statistics produced by MDSUMMARY are not configurable but are fixed and include:

Table 5.1 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>Column Name</td>
<td>Statistic</td>
<td>Description</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------</td>
<td>------------------------------------</td>
</tr>
<tr>
<td><em>CV</em></td>
<td>CV</td>
<td>Coefficient of variation</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>
<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>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>

Examples: MDSUMMARY Procedure

Example 1: Compute Descriptive Statistics

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

```plaintext
  caslib MyCasLib datasource=(srctype="path") path='your-file-path';
  libname mycas cas;
  proc casutil;
    load data=sashelp.cars outcaslib="MyCasLib ";
    contents casdata="cars";```
quit;

proc mdsummary data=mycas.cars; /* 
  var MSRP MPG_City;
  output out=mycas.mdsumstat;
run;

proc print data=mycas.mdsumstat; /*
  var _Column_ _NObs_ _Mean_ _Max_ _Min_ _Std_ ;
  title 'Summary of MSRP and City Miles Per Gallon';
run;
proc casutil; /*
  save casdata="mdsumstat" incaslib="MyCasLib" outcaslib="MyCasLib";
  list files incaslib="MyCasLib";
quit;

1 The CASLIB statement adds a session-scope caslib named MyCasLib. The caslib provides access to your data source. The DATASOURCE= option and the PATH= option provide connection information to your data source.

2 The LIBNAME statement creates a CAS engine libref. To run PROC MDSUMMARY and PROC PRINT in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names.

3 The CASUTIL procedure loads the data. The LOAD DATA= statement reads the file into memory. The table is now available for analytics. The CONTENTS statement reads the on-disk file, Cars, and displays the table metadata. This enables you to learn if the file has column names in the first row and the data types.


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

6 The CASUTIL procedure saves the table to the data source. The INCASLIB= option specifies the caslib that contains the file, and the OUTCASLIB= option specifies the caslib that the file is being made available to. The SAVE statement stored the data as UTF-8 when it created the SASHDAT file. The LIST FILES statement confirms that the in-memory table named Mdsumstat is saved in the data source.
Results

Output 5.1 Column Information for the Cars Data Set

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Label</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>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CARS</td>
<td>2004 Car Data</td>
<td>428</td>
<td>15</td>
<td>utf-8</td>
<td>05Apr2016 15:23:33</td>
<td>05Apr2016 15:23:33</td>
<td>No</td>
<td>No</td>
<td>No</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>Blocks Unmapped</th>
<th>Memory Unmapped</th>
<th>Blocks Allocated</th>
<th>Memory Allocated</th>
</tr>
</thead>
<tbody>
<tr>
<td>ALL</td>
<td>1</td>
<td>1</td>
<td>428</td>
<td>60480</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>60480</td>
</tr>
</tbody>
</table>

Column Information for CARS in Caslib MYCASLIB:

- Make: char 13
- Model: char 40
- Type: char 8
- Origin: char 6
- DriveTrain: char 5
- Invoice: double 8 DOLLAR
- EngineSize: Engine Size (L) double 8
- Cylinders: double 8
- Horsepower: double 8
- MPG_City: double 8
- MPG_Highway: double 8
- Weight: Weight (LBS) double 8
- Wheelbase: Wheelbase (IN) double 8
- Length: Length (IN) double 8

Results: MDSUMMARY Procedure

Output 5.2 Summary of MSRP and City Miles per Gallon

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>Column</em></th>
<th><em>NObs</em></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>32774.85514</td>
<td>192465</td>
<td>10280</td>
<td>19431.716674</td>
</tr>
<tr>
<td>2</td>
<td>MPG_City</td>
<td>428</td>
<td>20.060747664</td>
<td>60</td>
<td>10</td>
<td>5.2382176386</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 documentation about the CASUTIL procedure syntax, see Chapter 4, “CASUTIL Procedure,” on page 87.

- For more examples of using the CASUTIL procedure to access and save data, see “Accessing Data” in SAS Cloud Analytic Services: Accessing and Manipulating Data.
Example 2: Computing Descriptive Statistics with Group-By Variables

Program

The following examples shows you how to access your data with SAS Cloud Analytics Services (CAS), compute all statistics for each variable (treating the entire input table as one group), and computes all statistics for every unique combination of the formatted values of variables Make and Type. The results are written to an output table.

libname mycas cas; /*
proc casutil incaslib="casuser" outcaslib="casuser" ; /*
   load data=sashelp.cars;
   contents casdata="cars";
quit;

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

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

proc print data=mycas.carsmaketype;/**
   var make Type _Column_ _Nobs_ _Mean_ _Std_ _Min_ _Max_;
   title 'City Mileage by Make and Type';
run;

1 The LIBNAME statement creates a CAS engine libref. To run PROC MDSUMMARY and PROC PRINT in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names.

2 The CASUTIL procedure loads the data into the default caslib, Casuser. The LOAD DATA= statement reads the file into memory. The table is now available for analytics. The CONTENTS statement reads the on-disk file, Cars, and displays the table metadata. This enables you to learn if the file has column names in the first row and the data types.

3 The MDSUMMARY procedure produces summary statistics for MPG_CITY.

4 The first GROUPBY statement with the OUT= option creates an in-memory table named Mycas.CarsMPGCityAll that calculates summaries based on all qualifying rows of the input table

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

6 The MDSUMMARY procedure does not print output. The PRINT procedure prints the table Mycas.MdsumStat.
Example 2: Computing Descriptive Statistics with Group-By Variables

**Results:** MDSUMMARY Procedure

**Output 5.3  Overall Summary of City Mileage**

<table>
<thead>
<tr>
<th>Obs</th>
<th><em>Column</em></th>
<th><em>N Obs</em></th>
<th><em>Mean</em></th>
<th><em>Std</em></th>
<th><em>Min</em></th>
<th><em>Max</em></th>
</tr>
</thead>
</table>
| 1   | MPG City | 428     | 20.066747664 | 5.2382176386 | 10 | 60 |}

**Output 5.4  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>N Obs</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>Acura</td>
<td>SUV</td>
<td>MPG City</td>
<td>1</td>
<td>17</td>
<td></td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>Acura</td>
<td>Sedan</td>
<td>MPG City</td>
<td>5</td>
<td>20.24</td>
<td></td>
<td>18</td>
<td>24</td>
</tr>
<tr>
<td>3</td>
<td>Acura</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>4</td>
<td>Audi</td>
<td>Sedan</td>
<td>MPG City</td>
<td>13</td>
<td>18.615384615</td>
<td>2.3642638579</td>
<td>14</td>
<td>23</td>
</tr>
<tr>
<td>5</td>
<td>Audi</td>
<td>Sports</td>
<td>MPG City</td>
<td>4</td>
<td>19</td>
<td></td>
<td>15</td>
<td>21</td>
</tr>
<tr>
<td>6</td>
<td>Audi</td>
<td>Wagon</td>
<td>MPG City</td>
<td>2</td>
<td>16.5</td>
<td></td>
<td>15</td>
<td>18</td>
</tr>
<tr>
<td>7</td>
<td>BMW</td>
<td>SUV</td>
<td>MPG City</td>
<td>2</td>
<td>16</td>
<td></td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>8</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>9</td>
<td>BMW</td>
<td>Sports</td>
<td>MPG City</td>
<td>4</td>
<td>18.25</td>
<td></td>
<td>16</td>
<td>21</td>
</tr>
<tr>
<td>10</td>
<td>BMW</td>
<td>Wagon</td>
<td>MPG City</td>
<td>1</td>
<td>19</td>
<td></td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>11</td>
<td>Buick</td>
<td>SUV</td>
<td>MPG City</td>
<td>2</td>
<td>17</td>
<td></td>
<td>15</td>
<td>19</td>
</tr>
<tr>
<td>12</td>
<td>Buick</td>
<td>Sedan</td>
<td>MPG City</td>
<td>7</td>
<td>19.428571429</td>
<td>0.9759000729</td>
<td>18</td>
<td>20</td>
</tr>
<tr>
<td>13</td>
<td>Cadillac</td>
<td>SUV</td>
<td>MPG City</td>
<td>2</td>
<td>15</td>
<td>1.4142135624</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>14</td>
<td>Cadillac</td>
<td>Sedan</td>
<td>MPG City</td>
<td>4</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>15</td>
<td>Cadillac</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>16</td>
<td>Cadillac</td>
<td>Truck</td>
<td>MPG City</td>
<td>1</td>
<td>13</td>
<td></td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>17</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>18</td>
<td>Chevrolet</td>
<td>Sedan</td>
<td>MPG City</td>
<td>15</td>
<td>22.266666667</td>
<td>4.0964560758</td>
<td>14</td>
<td>28</td>
</tr>
<tr>
<td>19</td>
<td>Chevrolet</td>
<td>Sports</td>
<td>MPG City</td>
<td>2</td>
<td>18</td>
<td>0</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>20</td>
<td>Chevrolet</td>
<td>Truck</td>
<td>MPG City</td>
<td>5</td>
<td>15.2</td>
<td>1.9235384062</td>
<td>13</td>
<td>18</td>
</tr>
<tr>
<td>21</td>
<td>Chevrolet</td>
<td>Wagon</td>
<td>MPG City</td>
<td>1</td>
<td>22</td>
<td></td>
<td>22</td>
<td>22</td>
</tr>
<tr>
<td>22</td>
<td>Chrysler</td>
<td>Sedan</td>
<td>MPG City</td>
<td>13</td>
<td>20.307692308</td>
<td>1.6525039276</td>
<td>18</td>
<td>22</td>
</tr>
<tr>
<td>23</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>24</td>
<td>Chrysler</td>
<td>Wagon</td>
<td>MPG City</td>
<td>1</td>
<td>17</td>
<td></td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>25</td>
<td>Dodge</td>
<td>SUV</td>
<td>MPG City</td>
<td>1</td>
<td>15</td>
<td></td>
<td>15</td>
<td>15</td>
</tr>
</tbody>
</table>
Results: PROC CASUTIL CONTENTS Statement

Output 5.5  Metadata for the Cars 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 4, “CASUTIL Procedure,” on page 87.

- For more examples of using the CASUTIL procedure to access and save data, see “Accessing Data” in SAS Cloud Analytic Services: Accessing and Manipulating Data.

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

```sas
proc casutil; /*1*/
    load data=sashelp.cars;
    contents casdata="cars";
quit;
libname mycas cas; /*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.

The LIBNAME statement for the CAS engine creates a CAS libref. To run PROC MDSUMMARY and PROC PRINT in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names.

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.

The PRINT procedure prints the table Mycas.CarsMiles.
Results

Output 5.6  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>Column</th>
<th>NObs</th>
<th>Min</th>
<th>Max</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Audi</td>
<td>A4 3.0 Quattro 4dr manual</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>2</td>
<td>Audi</td>
<td>A41.8T convertible 2dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>3</td>
<td>Audi</td>
<td>A6 2.7 Turbo Quattro 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>4</td>
<td>Audi</td>
<td>A6 3.0 Quattro 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>5</td>
<td>Audi</td>
<td>TT 1.8 convertible 2dr (coupe)</td>
<td>Sports</td>
<td>VPG_City</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>6</td>
<td>Audi</td>
<td>TT 3.2 coupe 2dr (convertible)</td>
<td>Sports</td>
<td>VPG_City</td>
<td>1</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>7</td>
<td>BMW</td>
<td>325i 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>8</td>
<td>BMW</td>
<td>330Ci 2dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>BMW</td>
<td>330xi 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>10</td>
<td>BMW</td>
<td>C20i 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>11</td>
<td>BMW</td>
<td>M3 convertible 2dr</td>
<td>Sports</td>
<td>VPG_City</td>
<td>1</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>12</td>
<td>BMW</td>
<td>M3 coupe 2dr</td>
<td>Sports</td>
<td>VPG_City</td>
<td>1</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>13</td>
<td>BMW</td>
<td>Z4 convertible 2.5i 2dr</td>
<td>Sports</td>
<td>VPG_City</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>14</td>
<td>Buick</td>
<td>LeSabre Custom 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>15</td>
<td>Buick</td>
<td>LeSabre Limited 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>20</td>
<td>20</td>
</tr>
<tr>
<td>16</td>
<td>Buick</td>
<td>Regal GS 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>18</td>
<td>18</td>
</tr>
<tr>
<td>17</td>
<td>Buick</td>
<td>Rendezvous CX</td>
<td>SUV</td>
<td>VPG_City</td>
<td>1</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>18</td>
<td>Cadillac</td>
<td>SRX V6</td>
<td>SUV</td>
<td>VPG_City</td>
<td>1</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>19</td>
<td>Chevrolet</td>
<td>Cavalier 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>20</td>
<td>Chevrolet</td>
<td>Malibu LT 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>23</td>
<td>23</td>
</tr>
<tr>
<td>21</td>
<td>Chevrolet</td>
<td>Silverado SS</td>
<td>Truck</td>
<td>VPG_City</td>
<td>1</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>22</td>
<td>Chevrolet</td>
<td>Tahoe LT</td>
<td>SUV</td>
<td>VPG_City</td>
<td>1</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>23</td>
<td>Chrysler</td>
<td>Concorde LX 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>21</td>
<td>21</td>
</tr>
<tr>
<td>24</td>
<td>Chrysler</td>
<td>Concorde LXI 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>19</td>
<td>19</td>
</tr>
<tr>
<td>25</td>
<td>Chrysler</td>
<td>PT Cruiser 4dr</td>
<td>Sedan</td>
<td>VPG_City</td>
<td>1</td>
<td>22</td>
<td>22</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 4, “CASUTIL Procedure,” on page 87.

- For more examples of using the CASUTIL procedure to access and save data, see “Accessing Data” in SAS Cloud Analytic Services: Accessing and Manipulating Data.
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
libname mycas cas; /*
libname mycas cas; /*

proc format casfmtlib='fmtlib' ; /*
proc format casfmtlib='fmtlib' ; /*
   value $flvrfmt 'Chocolate'='Chocolate'
   value $flvrfmt 'Chocolate'='Chocolate'
   'Vanilla'='Vanilla'
   'Vanilla'='Vanilla'
   'Rum','Spice'='Other Flavor';
   'Rum','Spice'='Other Flavor';
   value agefmt (multilabel)
   value agefmt (multilabel)
      15 - 29='below 30 years'
      15 - 29='below 30 years'
      30 - 50='between 30 and 50'
      30 - 50='between 30 and 50'
      51 - high='over 50 years';
      51 - high='over 50 years';
run;
run;

data mycas.cake; /*
data mycas.cake; /*
   input LastName $ 1-12 Age 13-14 PresentScore 16-17
   input LastName $ 1-12 Age 13-14 PresentScore 16-17
   TasteScore 19-20 Flavor $ 23-32 Layers 34 ;
   TasteScore 19-20 Flavor $ 23-32 Layers 34 ;
   format age agefmt. flavor $flvrfmt.;
   format age agefmt. flavor $flvrfmt.;
datalines;
datalines;
Orlando     27 93 80  Vanilla    1
Orlando     27 93 80  Vanilla    1
Ramey       32 84 72  Rum        2
Ramey       32 84 72  Rum        2
Goldston    46 68 75  Vanilla    1
Goldston    46 68 75  Vanilla    1
Roe         38 79 73  Vanilla    2
Roe         38 79 73  Vanilla    2
Larsen      23 77 84  Vanilla    1
Larsen      23 77 84  Vanilla    1
Davis       51 86 91  Spice      3
Davis       51 86 91  Spice      3
Strickland  19 82 79  Chocolate 1
Strickland  19 82 79  Chocolate 1
Nguyen      57 77 84  Vanilla    .
Nguyen      57 77 84  Vanilla    .
Hildenbrand 33 81 83  Chocolate 1
Hildenbrand 33 81 83  Chocolate 1
Byron       62 72 87  Vanilla    2
Byron       62 72 87  Vanilla    2
Sanders     26 56 79  Chocolate 1
Sanders     26 56 79  Chocolate 1
Jaeger      43 66 74  1
Jaeger      43 66 74  1
Davis       28 69 75  Chocolate 2
Davis       28 69 75  Chocolate 2
Conrad      69 85 94  Vanilla    1
Conrad      69 85 94  Vanilla    1
Walters     55 67 72  Chocolate 2
Walters     55 67 72  Chocolate 2
Rossburger  28 78 81  Spice      2
Rossburger  28 78 81  Spice      2
Matthew     42 81 92  Chocolate 2
Matthew     42 81 92  Chocolate 2
Becker      36 62 83  Spice      2
Becker      36 62 83  Spice      2
Anderson    27 87 85  Chocolate 1
Anderson    27 87 85  Chocolate 1
Merritt     62 73 84  Chocolate 1
Merritt     62 73 84  Chocolate 1
;
;
proc mdsummary data=mycas.cake; /*
proc mdsummary data=mycas.cake; /*
   var TasteScore;
   var TasteScore;
   groupby flavor / out=mycas.flav; /*
   groupby flavor / out=mycas.flav; /*
   groupby flavor age / out=mycas.flag; /*
   groupby flavor age / out=mycas.flag; /*
run;
run;
```
The `LIBNAME` statement for the CAS engine creates a CAS engine libref. To run PROC MDSUMMARY and PROC PRINT in CAS, you must specify the CAS engine `LIBNAME` statement and use the CAS engine libref with both the input and output table names.

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

#### Taste Score for Cake Flavors and Participant’s Age GROUPBY 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>82.1686666667</td>
<td></td>
</tr>
</tbody>
</table>
**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.

```sas
proc casutil; /* 1 */
  load data=sashelp.cars;
  contents casdata="cars";
quit;

libname mycas cas; /* 2 */

proc mdsummary data=mycas.cars; /* 3 */
  var mpg_highway;
  groupby origin type / out=mycas.mpghw_sum;
run;

ods graphics / width=4in;
```

**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 4, “CASUTIL Procedure,” on page 87.

- For more examples of using the CASUTIL procedure to access and save data, see “Accessing Data” in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.

---

**Output 5.8**  PROC PRINT Output: Cake Flavors and Participant’s Age Grouped by Flavor

<table>
<thead>
<tr>
<th>Obs</th>
<th>Flavor</th>
<th>Age</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>Chocolate</td>
<td>below 30 years</td>
<td>TasteScore</td>
<td>5</td>
<td>75</td>
<td>85</td>
<td>80.4</td>
</tr>
<tr>
<td>2</td>
<td>Chocolate</td>
<td>between 30 and 50</td>
<td>TasteScore</td>
<td>2</td>
<td>83</td>
<td>92</td>
<td>87.5</td>
</tr>
<tr>
<td>3</td>
<td>Other Flavor</td>
<td>below 30 years</td>
<td>TasteScore</td>
<td>1</td>
<td>81</td>
<td>81</td>
<td>81</td>
</tr>
<tr>
<td>4</td>
<td>Other Flavor</td>
<td>over 50 years</td>
<td>TasteScore</td>
<td>1</td>
<td>91</td>
<td>91</td>
<td>91</td>
</tr>
<tr>
<td>5</td>
<td>Vanilla</td>
<td>below 30 years</td>
<td>TasteScore</td>
<td>1</td>
<td>80</td>
<td>80</td>
<td>80</td>
</tr>
<tr>
<td>6</td>
<td>Chocolate</td>
<td>over 50 years</td>
<td>TasteScore</td>
<td>2</td>
<td>72</td>
<td>84</td>
<td>78</td>
</tr>
<tr>
<td>7</td>
<td>Vanilla</td>
<td>between 30 and 50</td>
<td>TasteScore</td>
<td>2</td>
<td>73</td>
<td>75</td>
<td>74</td>
</tr>
<tr>
<td>8</td>
<td>Vanilla</td>
<td>over 50 years</td>
<td>TasteScore</td>
<td>3</td>
<td>84</td>
<td>94</td>
<td>88.3333333333</td>
</tr>
<tr>
<td>9</td>
<td>between 30 and 50</td>
<td>TasteScore</td>
<td>1</td>
<td>74</td>
<td>74</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>Other Flavor</td>
<td>between 30 and 50</td>
<td>TasteScore</td>
<td>2</td>
<td>72</td>
<td>83</td>
<td>77.5</td>
</tr>
</tbody>
</table>
The **LIBNAME** statement for the CAS engine creates a CAS engine libref. To run PROC MDSUMMARY, PROC SGPANEL, and PROC PRINT in CAS, you must specify the CAS engine LIBNAME statement and use the CAS engine libref with both the input and output table names.

2. The CASUTIL procedure loads the data into the default caslib, Casuser. The LOAD DATA= statement reads the file into memory. The table is now available for analytics. The CONTENTS statement reads the on-disk file, Cars, and displays the table metadata. This enables you to learn if the file has column names in the first row and the data types.

3. The MDSUMMARY procedure produces summary statistics for highway miles-per-gallon. The OUT= option in the GROUPBY statement creates a table in CAS. The table includes a row of summary statistics for each unique combination of origin and type.

4. The SGPANEL procedure plots the summarized results from the MDSUMMARY procedure. The procedure creates a parameterized vertical bar chart that shows the mean statistic for highway miles-per-gallon. The procedure subsets the data, comparing only the cars made in Asia and the U.S.A. The graph is paneled by country of origin.

### Results

The SGPANEL procedure generates the following graph:

**Output 5.9  Graph Output**
Additional Information

- Caslibs provide a way to organize in-memory tables and an associated data source. They also provide a way to apply access controls to data. In this example, the default caslib CASUSER is being used, so no CASLIB statement is needed. The in-memory table Mycas.Mpghw_Sum is temporary, and is dropped when the session is ended. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.

- For documentation about the CASUTIL procedure syntax, see Chapter 4, “CASUTIL Procedure,” on page 87.

- For more examples of using the CASUTIL procedure to access and save data, see “Accessing Data” in SAS Cloud Analytic Services: Accessing and Manipulating Data.

- 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 6.1 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>Based on COPIES= when the file is loaded.</td>
</tr>
</tbody>
</table>
## Dictionary

### HDFS Data Source

Specifies a Hadoop Distributed File System directory for loading and saving files that the SAS Cloud Analytic Services controller can access. SAS Cloud Analytic Services must be co-located with the Hadoop cluster to use this data source type.

**Applies to:** Distributed servers only, **CASLIB statement**

**Example:** Add a caslib to access files from the `/vapublic` directory in HDFS.

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

```
proc casutil incaslib="public" outcaslib="public";
list files;
quit;
```
Data Source Arguments

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

**PATH="/directory-path"**
- specifies the fully qualified path to the directory to use as a data source. Notice that the PATH= argument is specified outside of the parenthesis for the DATASOURCE= argument.

**SRCTYPE="HDFS"**
- specifies that the data source is an HDFS directory that is co-located with SAS Cloud Analytic Services.

**Requirement**  
The SRCTYPE= argument is required.

---

DNFS Data Source

Specifies a server-side directory for loading and saving files that the SAS Cloud Analytic Services controller can access. The directory must be mounted by every machine that is used by the server.

**Applies to:** Distributed servers only, CASLIB statement

**Example:** Add a caslib to access files from the /data01 directory on the controller.

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

```
proc casutil incaslib="dnfsds" outcaslib="dnfsds";
   list files;
quit;
```

Syntax

**Data Source Arguments**

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

**PATH="/directory-path"**
- specifies the fully qualified path to the directory to use as a data source. Notice that the PATH= argument is specified outside of the parenthesis for the DATASOURCE= argument.

**SRCTYPE="DNFS"**
- specifies that the data source is a directory that is mounted by every machine that is used for SAS Cloud Analytic Services.

**Requirement**  
The SRCTYPE= argument is required.

Details

DNFS is an abbreviation for distributed network file system. This data source type provides support for distributed data access to NFS directories. Several systems such as MapR-FS, EMC Isilon, and others provide high-availability, replicated, high-performance, stand-alone storage clusters with an NFS interface. These systems offer
popular alternatives to Hadoop. DNFS provides a good alternative for deployments where the server is not co-located with Hadoop and yet must provide similar capabilities.

DNFS can also be used to access NFS-mounted directories from standard UNIX or Linux file systems.

The design principle is that NFS-mounted directories are accessed concurrently by each controller node and worker node in a distributed server. This is why the directory path for a DNFS caslib must be mounted on every machine. DNFS performs parallel read and write for SASHDAT and CSV files that are stored in the directory path specified for the caslib.

---

**SAS LASR Analytic Server**

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

**Applies to:** CASUTIL procedure, CASLIB statement

**Examples:**

Add a caslib to access data from a SAS LASR Analytic Server.

```plaintext
caslib publiclasr datasource=(
  srctype="lasr"
  server="gridhost.example.com"
  port=10050
  signer="https://webserver.example.com/SASLASRAuthorization"
  username="sasdemo"
  password="secret"
); 
```

Load a table from a SAS LASR Analytic Server into SAS Cloud Analytic Services.

```plaintext
proc casutil incaslib="publiclasr";
  load casdata="epa_cars" casout="epa_cars"
    importoptions=(filetype="lasr" varchars="true");
run;
```

---

**Syntax**

**Data Source Arguments**

- **METALIB="metadata-libref"**
  - specifies the libref name for the SAS LASR Analytic Server engine library.

- **PASSWORD="string"**
  - specifies the password for the identity in the USERNAME= option.

- **PORT=integer**
  - specifies the network port that the SAS LASR Analytic Server listens on.

- **SERVER="host-name"**
  - specifies the host name or IP address of the SAS LASR Analytic Server.

- **SIGNER="authorization-web-service-uri"**
  - specifies the URI for the SAS LASR Authorization web service. This is specified in the form SIGNER="https://server.example.com/SASLASRAuthorization".

- **SRCTYPE="LASR"**
  - specifies that the data source is a SAS LASR Analytic Server.
Requirement  The SRCTYPE= argument is required.

TAG=server-tag
specifies a tag that is used to qualify the names of SAS LASR Analytic Server tables that are accessed using this caslib.

USERNAME="user-ID"
specifies an identity that is authorized to access data in the SAS LASR Analytic Server.

File Type Arguments

COMPPGM="string"
specifies an expression for each variable that you included in the COMPVARS option. End the expression for each variable with a semicolon.

COMPVARS=("computed-variable-1" <, "computed-variable-2", ...>)
specifies the names of the computed variables to create. Specify an expression for each parameter in the COMPPGM option.

FILETYPE="LASR"
specifies the file type.

Requirement  The FILETYPE= argument is required.

PARALLELMODE="FALLBACK" | "FORCE" | "NONE"
specifies how the table is transferred from SAS LASR Analytic Server to SAS Cloud Analytic Services when both servers are distributed servers. If either server is running as a single-machine server, then parallel data transfer is not possible and that is equivalent to NONE.

FALLBACK
specifies that the worker nodes try to establish communication with each other. If the worker nodes cannot connect, the operation falls back to a serial data transfer between the SAS LASR Analytic Server root node and the SAS Cloud Analytic Services controller node. Serial data transfer is slower than parallel data transfer.

FORCE
specifies that the worker nodes try to establish communication with each other. If the worker nodes cannot connect to perform a parallel data transfer, then the load request fails.

NONE
specifies to perform a serial data transfer between the SAS LASR Analytic Server root node and the SAS Cloud Analytic Services controller node.

Default  FALLBACK

PRESERVEORDER=TRUE | FALSE
when set to True, the rows are inserted into the new table in the same order as they are received from the SAS LASR Analytic Server. Creating the table is less efficient when this parameter is used.

Default  FALSE

VARCHARS=TRUE | FALSE
when set to True, variable-length strings are used for character variables.

Default  FALSE
VARS="string-1" <, "string-2", ...>
    specifies the variables to use in the action.

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

Details

Arguments Summary

Table 6.2 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></td>
<td>Required</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>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>
</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.
Serial data transfer is slower, but is not subject to the memory monitoring during a data transfer.

If the SAS LASR Analytic Server and CAS are on separate clusters, you can set the threshold to 100% to transfer tables to CAS. The threshold can be set when the SAS LASR Analytic Server is started by specifying a value for the EXTERNALMEM= option. Or, you can modify the threshold for a running server by specifying a new value in the SERVERPARM statement of the IMSTAT procedure.

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

**Example**

This example shows how to load a table from a SAS LASR Analytic Server and use the VARS= options. The sample Epa_cars data is available from the SAS Visual Analytics 7.3 documentation page at [http://support.sas.com/documentation/onlinedoc/va/index.html](http://support.sas.com/documentation/onlinedoc/va/index.html).

```sas
   caslib valasr datasource=(
      srctype="lasr"  
      /* ...data source options... */
   );
   cas casauto sessopts=(caslib="valasr");  /* I */
   proc casutil;
```
list files; /* 2 */

load casdata="epa_cars" casout="epa_cars"
  importoptions=(
    filetype="lasr"
    varchar="true"
    vars=("model_year" "vehicle_manufacturer_name") /* 3 */
  );

  contents casdata="epa_cars";
quit;

1 The CAS statement is used to set the active caslib for the Casauto session to Valasr explicitly.
2 The LIST FILES statement shows the tables that are in SAS LASR Analytic Server.
3 The VARS= option is used to subset the columns that are read from SAS LASR Analytic Server and loaded into SAS Cloud Analytic Services.

Path Data Source

Specifies a server-side directory for loading and saving files that the SAS Cloud Analytic Services controller can access.

**Applies to:** CASLIB statement

**Example:** Add a caslib to access files from the /data01 directory on the controller.

```bash
  caslib pathds datasource=(srctype="path") path="/data01";

  proc casutil incaslib="pathds" outcaslib="pathds";
    list files;
  quit;
```

**Syntax**

**Data Source Arguments**

**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=**"PATH"

specifies that the data source is a directory that is accessible to the SAS Cloud Analytic Services controller.

**Requirement** The SRCTYPE= argument is required.
Chapter 7
Platform File Types

Dictionary

Delimited Files (CSV)
Specifies the file type options for loading data from delimited files.

Applies to: CASUTIL procedure

Example: Load a Latin1 encoded CSV file into SAS Cloud Analytic Services.

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

Syntax

**File Type Arguments**

**DELIMITER="string"**

specifies the character to use as the field delimiter.

Default ","

**ENCODING="string"**

specifies the text encoding of the file. If the file is not encoded in UTF-8 or 7-bit ASCII, then specify the encoding.

Default utf-8

**FILETYPE="CSV"**

specifies the file type.
Requirement  The FILETYPE= argument is required.

GETNAMES=TRUE | FALSE
when set to True, the values in the first line of the file are used as variable names.

Default  TRUE

GUESSROWS=integer
specifies the number of rows to scan in order to determine data types for variables.

Default  20

LOCALE="string"
specifies the locale for interpreting data in the file.

STRIPBLANKS=TRUE | FALSE
removes leading and trailing blanks from character variables.

Default  FALSE

VARCHARS=TRUE | FALSE
when set to True, variable-length strings are used for character variables.

Default  TRUE

VARS=((casvardesc-1), (casvardesc-2, ...))
specifies the names, types, formats, and other metadata for variables.

FORMAT="string"
specifies the format to apply to the variable.

FORMATTEDLENGTH=integer
specifies the format field length plus the format precision length.

LABEL="string"
specifies the descriptive label for the variable.

LENGTH=integer
specifies the unformatted length of the variable. This parameter applies to fixed-length character variables (type="CHAR") only.

Default  8

NAME="string"
specifies the name for the variable.

NFD=integer
specifies the format precision length.

NFL=integer
specifies the format field length.

TYPE="CHAR" | "DOUBLE" | "VARCHAR"
specifies the data type for the variable.

Details
Delimited files can be read from caslibs with the following data source types:

•  DNFS
•  HDFS
• PATH

Example

By default, SAS Cloud Analytic Services expects to find column names in the first line of the file. If you have a file that does not include names, you must specify GETNAMES=FALSE. You might also prefer to specify column names when you load the data.

File 7.1 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 7.1 Load a CSV File and Specify Column Names

```sas
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 7.1  Column Information from the CONTENTS Statement

Column Information for CLASS_FR in Caslib CASUSER(......)

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>sexe</td>
<td>varchar</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>nom</td>
<td>varchar</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>âge</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>la taille</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>poids</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

Output 7.2  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>

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.

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

Syntax

File Type Arguments

FILETYPE="HDAT"

specifies the file type.

Requirement  The FILETYPE= argument is required.

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.
Details

SASHDAT files can be read from caslibs with the following data source types:

- DNFS
- HDFS
- PATH

Instead of specifying the ENCRYPTIONPASSWORD= option on a file-by-file basis, you can use it as a data source option when you add a caslib. For an example, see Encrypt Tables in a Caslib on page 82.
Chapter 8
Data Connectors

Working with SAS Data Connectors .............................................. 145
Quick Reference for Data Connector Syntax ................................. 146
Where to Specify Data Connector Options .................................. 147
Using Wildcards for Pattern Matching ....................................... 148
  Overview of Wildcards ....................................................... 148
  Using Wildcards with SAS Macros ....................................... 149
Loading a Subset of Table Rows .............................................. 149

Dictionary .............................................................................. 150
  SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop 150
  SAS Data Connector to Impala .............................................. 161
  SAS Data Connector to ODBC ............................................. 168
  SAS Data Connector to Oracle ............................................. 174
  SAS Data Connector to PC Files .......................................... 178
  SAS Data Connector to PostgreSQL .................................... 180
  SAS Data Connector to SAS Data Sets .................................. 186
  SAS Data Connector to Teradata and SAS Data Connect Accelerator for Teradata .............................................. 190

Working with SAS Data Connectors

To access and load data, the administrator must create and configure data connectors for SAS Cloud Analytic Services (CAS). Data connectors contain connection information and data-source specifics to connect with data sources, such as Oracle or SAS data sets. For instructions on how to configure data connectors, see the SAS® Viya™ 3.2: Deployment Guide.

There are two methods to load data into CAS. You can load data serially with a data connector, or you can load data in parallel with a data connect accelerator. Data connect accelerators work with the SAS Embedded Process and must be licensed separately.

You can associate data connectors that require logins with a domain on the CAS server. When users connect to the data source through a data source name (DSN), the domain name is used to retrieve user credentials that are associated with that data connector. The credentials are then passed to the third-party data source. User credentials are stored on your system and are accessible by the CAS server.

Data connectors can also contain optional information to control CAS data source behavior. Data connectors form the foundation for connectivity to a third-party database
or data source. You can assign privileges that control user access to the data. However, relational databases provide authorization that limits the operations that can be performed on the data. CAS respects authorizations that are defined and enforced by a third-party database or data source. Authorizations that are defined on a third-party database overrule permissions and privileges that are set in CAS.

Be aware that, if you save the contents of a SAS data set to a table in a database management system (DBMS), the resulting database table might contain different data types than the original SAS data set. In addition, any formats on the SAS data set that cannot be mapped to an equivalent DBMS type are not preserved. Verify that your data is what you expect after copying from SAS to a DBMS. For more information about what SAS data types and functions can be preserved, refer to SAS/ACCESS for Relational Databases: Reference for the target database.

---

**Quick Reference for Data Connector Syntax**

This table shows the syntax, supported file types (if applicable), and an example for each data source. For path-based data source syntax, see the DATASOURCE= parameter for the CASLIB statement on page 69.

**Table 8.1 Data Source Types and Options**

<table>
<thead>
<tr>
<th>Data Connector</th>
<th>srcType= Type</th>
<th>Option Syntax</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hadoop</td>
<td>hadoop</td>
<td>“SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop” on page 150</td>
<td>caslib hvlib desc=&quot;Hadoop Caslib* datasource=(srctype=&quot;hadoop&quot;, dataTransferMode=&quot;parallel&quot;, hadoopjarpath=&quot;/data/cdh54/sdm/lib&quot;, hadoopconfigdir=&quot;/data/cdh54/sdm/conf&quot;, username=&quot;hadoopuser&quot;, server=&quot;hive01.example.com&quot;, schema=&quot;default&quot;);</td>
</tr>
<tr>
<td>Impala</td>
<td>impala</td>
<td>“SAS Data Connector to Impala” on page 161</td>
<td>caslib imlib desc=&quot;Impala Caslib* datasource=(srctype=&quot;impala&quot;, username=&quot;impalauser&quot;, server=&quot;impala01.example.com&quot;);</td>
</tr>
<tr>
<td>ODBC</td>
<td>odbc</td>
<td>“SAS Data Connector to ODBC” on page 168</td>
<td>caslib odbccaslib desc=&quot;ODBC caslib* datasource=(srctype=&quot;odbc&quot;, username=&quot;user1&quot; password=&quot;password1&quot; databases=&quot;dbodbc&quot; catalog=&quot;*&quot;));</td>
</tr>
<tr>
<td>Oracle</td>
<td>oracle</td>
<td>“SAS Data Connector to Oracle” on page 174</td>
<td>caslib oraclecaslib desc=&quot;Oracle Caslib* datasource=(srctype=&quot;oracle&quot;, username=&quot;user1&quot;, password=&quot;password1&quot;, path=&quot;/machine.lnx.com:5570/exadat&quot;);</td>
</tr>
<tr>
<td>Data Connector</td>
<td>srcType= Type</td>
<td>Option Syntax</td>
<td>Example</td>
</tr>
<tr>
<td>----------------</td>
<td>-------------</td>
<td>---------------</td>
<td>---------</td>
</tr>
</tbody>
</table>
| PC Files       | path        | “SAS Data Connector to PC Files” on page 178 | caslib pcfiles caslib desc='PCFiles Caslib'
  datasource=(srctype='path'
    filetype='xls'
    path="/path/mytest/customers.xls"
    getnames='true'
    range="A10-D14"); |
| PostgreSQL     | postgres    | “SAS Data Connector to PostgreSQL” on page 180 | caslib postgres caslib desc='PostgreSQL Caslib'
  datasource=(srctype='postgres'
    server='PGserver'
    username='user1'
    password='myPwd'
    database="PGdatabase-name"); |
| SAS Data Sets  | path        | “SAS Data Connector to SAS Data Sets” on page 186 | caslib sasdataset caslib desc='SASDataSets Caslib'
  datasource=(srctype='path'
    path="/mytest/customer"
    password='password1'
    filetype='basesas' dtm="parallel"); |
| Teradata       | teradata    | “SAS Data Connector to Teradata and SAS Data Connect Accelerator for Teradata” on page 190 | caslib TDcaslib desc='Teradata Caslib'
  datasource=(srctype='teradata'
    username='user1'
    password='password1'
    server='teradataServer"); |

**Where to Specify Data Connector Options**

The options that you specify for your data connector are specific to your data source. In general, you specify options for data connectors in the following places in your SAS code.

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, then the caslib can be added as follows:

```sas
/* 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, or LOAD statement.

```sas
proc casutil;
  list files incaslib="tdlib" dataSourceOptions=(schema="tdSchema");
```
### 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 "aBCe". To do this, you might specify this pattern to match:

```
path='aBc_%'
```

You can use wildcards in the `path=` parameter for the `fileInfo` action. Case sensitivity is based on the behavior of your data source. You can use multiple wildcard characters within a pattern.

Here are the available wildcard characters:

- `%` matches any number of characters.
- `_` matches a single character. You can include more than one `_` in a pattern.
- `\` escapes a wildcard character so that it is treated as a literal character instead of a wildcard in a pattern.

If you have filenames that contain underscores, escape the `_` character with the `\`. For example, this pattern searches for all files that begin with ‘aBc_’:

```
path='aBc_&'
```
Using Wildcards with SAS Macros

The SAS Macro language also uses the ‘%’ character to indicate variables to resolve during program compilation. When you use the ‘%’ character for pattern matching, include the value in single quotations marks. If you enclose a value in double quotation marks, the SAS Macro parser tries to resolve a macro variable name. For example, the code `path="testmisc/prdsa%e.csv"` results in an attempt to resolve a macro variable, E, and typically results in an error. Instead, specify the code as `path='testmisc/prdsa%e.csv'`.

Loading a Subset of Table Rows

If you want to load a subset of the data in a table, based on a condition that you specify, you use a WHERE clause. The way that you indicate the WHERE clause depends on the data source that you are accessing.

If you are loading data from a SAS data set, then you can use the WHERE= option for the LOAD statement in PROC CASUTIL.

```bash
proc casutil;
  load data=sashelp.cars(where=(msrp>20000)) casout="some_cars"
    vars=(name="make"), (name="model"));
quit;
```

If you want to load a subset of data from an external database or data source table, such as PostgreSQL or Teradata, then use the DBMSWHERE= data connector option. With the DBMSWHERE= option, the WHERE clause that you specify is passed directly to the external database or data source for use while loading the data.

You can use the DBMSWHERE= option only on tables that do not have restrictions on access. For example, you could not use the DBMSWHERE= option on a table with restricted access to one or more columns. If you try to use the DBMSWHERE= option for a table with restricted access, you receive an error in the log.

```bash
proc casutil;
  load casdata='cars' incaslib='tdlib'
    casout='cars_CAS'
    options=(dbmsWhere='cylinders=8'
              dataTransferMode='parallel');
quit;
```

Note: Using the WHERE= option in a LOAD CASDATA= statement results in an error for an external database or data source.

For more information, see the LOAD Statement for PROC CASUTIL.
SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop

SAS Data Connector to Hadoop lets you load data serially from Hive into SAS Cloud Analytic Services. All users can use SAS Data Connector to Hadoop. SAS Data Connect Accelerator for Hadoop is an additional product that lets you load data in parallel using the SAS Embedded Process.

Valid in:  
- CASLIB statement  
- CAS table actions  
- 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.

Requirement:  
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.

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

See:  
Chapter 9, "Data Types,"

Examples:  
Load a Hadoop data source and add a caslib to Hadoop.

```bash
caslib hvlib sessref=mysess
dataSource=(
    srctype='hadoop',
    dataTransferMode='parallel',
    server='HiveServer',
    username='myuser1'
    hadoopJarPath="<Hadoop jar path directory>",
    hadoopConfigDir="<Hadoop configuration directory>",
    schema="<Hive schema name>");
```

Load a Hadoop data source using PROC CASUTIL.

```bash
proc casutil;
    incaslib="hvlib"
    sessref=mysess;
    load incaslib="hvlib" casdata="cars"
        casout="cars_CAS"
            options=(dbmsWhere="cylinders=8",
            dataTransferMode="parallel");
run;
quit;
```

Syntax
Data Connector Options for Hadoop

For each described option, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements and action calls, see “Where to Specify Data Connector Options” on page 147.

**AUTHENTICATIONDOMAIN="domain"**

specifies the name of the authentication domain that contains credentials (USERNAME= and PASSWORD=) to use 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, then these credentials override any that were set when the caslib was added.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alias</td>
<td>AUTHDOMAIN=</td>
</tr>
<tr>
<td>Requirement</td>
<td>If your Hive database or data source requires authentication, then 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</td>
</tr>
</tbody>
</table>

**BUFFERSIZE=bytes**

specifies the buffer size length in bytes of the buffer that is used to receive data from SAS embedded processes. This value overrides the BUFFERSIZE= value that is set in the CASLIB statement. Increasing the size might result in better performance with a trade-off of increased memory usage.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1048576</td>
</tr>
<tr>
<td>Restriction</td>
<td>This option applies only when DATATRANSFERMODE=&quot;parallel&quot; for SAS Data Connect Accelerator for Hadoop.</td>
</tr>
<tr>
<td>Supports</td>
<td>HDMD, Hive, parallel</td>
</tr>
</tbody>
</table>

**CATALOG="catalog name"**

specifies a logical catalog name for data sources that do not natively support catalogs. 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>Default</td>
<td>active caslib</td>
</tr>
</tbody>
</table>

**DATATRANSFERMODE="AUTO | PARALLEL | SERIAL."**

specifies the mode of data transfer. This value overrides a value of DATATRANSFERMODE= that was set in the CASLIB statement.
Here are the valid values.

**AUTO** specifies to first try to load the data in parallel (using embedded processing). If this fails, a note-level message is issued and serial processing is then attempted.

**PARALLEL** specifies to load the data in parallel by using the SAS Data Connect Accelerator to your database or data source.

**SERIAL** specifies to load the data serially by using the SAS Data Connector to your database or data source.

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

**Aliases**
- DATATRANSFER=
- DTM=

**Default** SERIAL

**Requirement** To use the PARALLEL option, you must have a licensed copy of the SAS Data Connect Accelerator for Hadoop.

**Supports** HDMD, Hive, parallel, serial

**Tip** Specify PARALLEL if you know that all nodes have access to the data set. Specify SERIAL if you know that only the controller has access to the data set. If you are not sure what type of access the nodes have to the data set, specify AUTO or do not specify a value for this option. In this case PARALLEL is used if all nodes can access the data set. Otherwise, SERIAL is used.

**DBMSWHERE=**"WHERE-clause"

specifies a database- or data source-specific SQL WHERE clause to submit to the database or data source. The WHERE clause that you specify is passed to the database or data source exactly as you enter it. This option is used to filter the rows that are read into a CAS table.

**Valid in** PROC CASUTIL: LOAD statement

**Default** "" (zero-length string)

**Restriction** This option is not allowed when CAS column security is defined for a table. Also, you can use this option only on tables that do not have restrictions on column access. For more information, see “Loading a Subset of Table Rows”.

**HADOOPCONFIGDIR=**"configuration files directory"

specifies the Hadoop configuration files directory, which is the one that is obtained by running Ansible or the Hadoop extract script on the target Hadoop cluster to set the configuration path.

**Valid in** CASLIB statement [required]

**Default** "" (zero-length string)
**HADOOPJARPATH=*/"Hadoop jar files path"*

specifies one or more paths to the Hadoop JAR files. These are the JAR files that are obtained by running Ansible or the Hadoop extract script on the target Hadoop cluster to set the jar path. These files are delimited by colons for Linux.

- **Valid in**: CASLIB statement [required]
- **Default**: "" (zero-length string)
- **Supports**: HDMD, Hive, parallel, serial

**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 [required]
- **Default**: "" (zero-length string)
- **Supports**: HDMD, parallel, serial

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

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

- **Valid in**: CASLIB statement
- **Default**: none
- **Supports**: HDMD, parallel, serial

**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, serial

**NAME=*/"hadoop"*

specifies Hadoop as the data source.

- **Default**: none

    **Example**: s:loadDatasource{name='hadoop'}

**PASSWORD=*/"password"*

specifies the Hadoop 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, then these credentials override any credentials that were specified when the 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>
</tbody>
</table>

**Aliases**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>PASS=</td>
<td></td>
</tr>
<tr>
<td>PWD=</td>
<td></td>
</tr>
</tbody>
</table>

**Default**

"" (zero-length string)

**Requirement**

If your Hive 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, serial

**PATH=**"Hive table path and name"

specifies the name of a Hive source table.

**Default**

none

**Example**

```properties=
*hive.exec.dynamic.partition.mode=nonstrict;hive.exec.dynamic.partition=true
```

**PORT=**port value

specifies the Hive JDBC port number.

**Valid in**

CASLIB statement

**Default**

10000

**Range**

1–65535

**Supports**

Hive, parallel, serial

**PROPERTIES=**"Hive JDBC properties value"

specifies a free-form value for Hive JDBC properties. The value is appended to the JDBC connection URI. You can use it to override default Hive behaviors.

**Valid in**

CASLIB statement

**Default**

none

**Supports**

Hive, parallel, serial

**READBUFF=**"integer"

specifies the number of rows to fetch per block of data when fetching in serial data transfer mode.

**Valid in**

CASLIB statement

**PROC CASUTIL: LOAD statement**

**Aliases**

<table>
<thead>
<tr>
<th>Alias</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>RAS=</td>
<td></td>
</tr>
</tbody>
</table>
### row_array_size=

**Default**
Calculated automatically based on row size

**Restriction**
This option is not valid when DATATRANSFERMODE="parallel" for the SAS Data Connect Accelerator. Specify BUFFERSIZE= when loading data in parallel.

**Supports**
HDMD, Hive, serial

### SCHEMA="schema-name"

Specifies the Hive database name.

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, then the table mySchema.Studydata is accessed. If you specify a value for SCHEMA= in a statement other than the CASLIB statement, then this value overrides any value of SCHEMA= that was set when the caslib was added.

**Valid in**
- CASLIB statement
- PROC CASUTIL: CONTENTS, LIST, LOAD statements

**Default**
"default"

**Supports**
Hive, parallel, serial

### SERVER="Hive server identifier"

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

**Valid in**
- CASLIB statement

**Default**
"" (zero-length string)

**Supports**
Hive, parallel, serial

### SRCTYPE="hadoop"

Specifies that the data source is Hadoop (Hive data source).

**Valid in**
- CASLIB statement [required]

**Default**
"" (zero-length string)

**Supports**
HDMD, Hive, parallel, serial

### STATUSINTERVAL=number

Specifies whether to print a message to the client when a node adds \( n \) buffers to the table, where \( n \) is the value of this option. This value overrides the STATUSINTERVAL= value that was set in the CASLIB statement.

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

**Default**
0 (no message)
Restriction
This option applies only when DATATRANSFERMODE="parallel" for SAS Data Connect Accelerator for Hadoop.

Supports
HDMD, Hive, parallel

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, serial

USERNAME=":user-name"
specifies the username to connect to Hadoop. 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, then these credentials override any credentials that were specified when the caslib was added.

Valid in
CASLIB statement

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Aliases
UID=

USER=

Default
(none) If you do not specify this option, the current user context is used, most often to use the current Kerberos context.

Requirement
If it is required by Hive database or data source, then you must specify valid credentials to access data. You can provide these credentials by specifying eitherUSERNAME= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

Supports
HDMD, Hive, parallel, serial

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 147.
**Hadoop Naming**
The data connector and data connect accelerator can load Hive tables with names up to 128 characters or with column names that are up to 128 characters.

**Authentication to a Hive Data Source**
Depending on the user setup, credentials might be required to access the data in a Hive data source. Your system administrator defines an authentication domain that is associated with a pair of user ID and password values. The authentication domain and associated credentials are then available to you and other users who need to access data.

Typically, you supply credentials when you add a caslib. To do this, use the CASLIB statement and specify a value for either AUTHENTICATIONDOMAIN= or both USERNAME= and PASSWORD=. These credentials are then used for any statement that accesses the data using that caslib.

You can specify different credentials when you issue a statement that accesses the data. If you originally specified an authentication domain when you added a caslib, then you must specify values for both USERNAME= and PASSWORD= to override the original credentials. You can also override authentication domain credentials with a different AUTHENTICATIONDOMAIN= value.

While 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 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 the SAS Data Connect Accelerator for Hadoop is installed on your system, 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 SASHDMD.

For more information about creating metadata files, see *SAS In-Database Products: User's Guide*. 
Define a Hive or HDMD Caslib

When you define a Hadoop caslib, you must specify whether to use it to access Hive tables or HDMD tables. Using the HDFSMETADIR= option determines whether the caslib is a Hive caslib or an HDMD caslib.

Once you define the caslib as either Hive or HDMD, you cannot change the caslib type at the action level. Data source options that are specific to Hive that are on an HDMD caslib are ignored, as are data source parameters that are specific to HDMD that are on a Hive caslib.

Table 8.2  Options for Specifying a Hive or HDMD Caslib

<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></td>
<td></td>
</tr>
<tr>
<td>not specified</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>not specified</td>
<td>specified</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></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></td>
<td></td>
</tr>
<tr>
<td>not specified</td>
<td>specified</td>
<td></td>
<td>Hive</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>not specified</td>
<td>specified</td>
<td></td>
<td>Hive server must be High Availability (HA)</td>
</tr>
</tbody>
</table>

Supported Hive and HDMD Data Types

This table shows the data types that can be read into SAS Cloud Analytic Services from a Hive data source along with the resulting data type and format when data is read into SAS Cloud Analytic Services. The length of the data format in SAS Cloud Analytic Services is based on the length of the source data.

Table 8.3  Supported Hive Data Types

<table>
<thead>
<tr>
<th>Hive Data Type</th>
<th>SAS Cloud Analytic Services Data Type</th>
<th>Default Format in SAS Cloud Analytic Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>w.</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
<td>w.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>w.d</td>
</tr>
</tbody>
</table>

Through conversion, other data types are supported:

- STRING to VARCHAR
• BIGINT, DECIMAL, INTEGER, SMALLINT, TINYINT, and FLOAT to DOUBLE
• DATE, TIME, TIMESTAMP to DOUBLE with FORMAT

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.

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.

```sas
caslib hvlib desc="hadoop caslib"
   dataSource=(srctype="hadoop",
               dataTransferMode="parallel",
               server="hiveServer",
               username="myuser1",
               password="myPwd",
               hadoopJarPath="<Hadoop jar path directory>",
               hadoopConfigDir="<Hadoop configuration directory>",
               schema="<Hive schema name>");
```

Example 2: Specify HDMD as a Data Source for a Caslib in SAS Cloud Analytic Services
Use the CASLIB statement to initialize the data source and add the caslib for Hadoop. No connection is made to the data source until a statement that accesses the data is called. The data is read in serially into the caslib hvcaslib.

```sas
caslib hdmdlib desc="hdmd caslib"
   dataSource=(srctype="hadoop",
               dataTransferMode="serial",
               username="<Hadoop jar path directory>",
               hadoopJarPath="<Hadoop jar path directory>",
               hadoopConfigDir="<Hadoop configuration directory>",
               hadoopDataDir="<HDFS data directory>",
               hadoopMetaDir="<HDFS SASHDMD directory>");
```

Example 3: Load a Hive Table into SAS Cloud Analytic Services Using PROC CASUTIL

```sas
proc casutil;
   list files incaslib="Hadoopcaslib";       \1
   load casdata="myHDdata" incaslib="Hadoopcaslib" outcaslib="casuser"
      casout="HData_from_Hadoopcaslib";       \2
   list tables incaslib="casuser";
   contents casdata="HData_from_Hadoopcaslib" incaslib="casuser";    \3
quit;                                           \4
```

1 List the tables in Hadoopcaslib before loading your data.
2 Load the table myHDdata from Hive into the caslib, casuser. Call the new table HDdata_from_Hadoopcaslib.

3 List the tables in caslib casuser to see the newly created HDdata_from_Hadoopcaslib table that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

**Example 4: Load an HDMD Table into SAS Cloud Analytic Services Using PROC CASUTIL**

```sas
proc casutil;
  list files incaslib="HDMDcaslib";
  load casdata="myHDdata" incaslib="HDMDcaslib" outcaslib="casuser2" casout="HDdata_from_HDMDcaslib";
  list tables incaslib="casuser2";
  contents casdata="HDdata_from_HDMDcaslib" incaslib="casuser2";
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 casuser2 to 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.

**Example 5: Using Lua to Connect to Hive, Examine the Data, and Load a Table**

```lua
-- Add the Caslib for Hadoop
r = s:addCaslib{lib="hvlib",
  dataSource={srcType="hadoop",
    server="<Hive server name>",
    dataTransferMode="parallel",
    schema="<Hive schema name>",
    hadoopJarPath="<Hadoop jar path directory>",
    hadoopConfigDir="<Hadoop configuration directory>",
    username="<Hadoop username>";}}

print("Calling fileInfo")
r = s:fileInfo{caslib='hvlib'}
print(r)

print("Calling columnInfo")
r = s:columnInfo{table={caslib="hvlib", name="Hive table name"}}
print(r)

print("Calling loadTable")
r = s:loadTable{caslib="hvlib", path="Hive table name"}
print(r)
```
Example 6: Using Lua to Connect to HDMD, Examine the Data, and Load a Table

```lua
-- Add the Caslib for Hadoop
r = s:addCaslib{lib="hdmdlib",
dataSource={srcType="hadoop",
hdfsMetaDir="<HDFS SASHDMD directory>",
hdfsDataDir="<HDFS data directory>",
dataTransferMode="serial",
hadoopJarPath="<Hadoop jar path directory>",
hadoopConfigDir="<Hadoop configuration directory>",
username="<Hadoop username>";}}

print("Calling fileInfo")
r = s:fileInfo{caslib='hdmdlib'}
print(r)

print("Calling columnInfo")
r = s:columnInfo{table={caslib="hdmdlib", name="HDMD table name"}}
print(r)

print("Calling loadTable")
r = s:loadTable{caslib="hdmdlib", path="HDMD table name"}
print(r)
```

SAS Data Connector to Impala

SAS Data Connector to Impala lets you load data serially from Impala into SAS Cloud Analytic Services. All users can use SAS Data Connector to Impala.

**Valid in:**
- CASLIB statement
- CAS table actions
- PROC CASUTIL statements (see parameters for details)

**See:**
- Chapter 9, “Data Types,”

**Examples:**
Load an Impala data source and add a caslib to Impala.

```sas
caslib imlib sessref=mysess
dataSource=(srctype='impala',
server='impala01.example.com',
username='myuser1'
password='myPwd'
path="/machine.lnx.com:1521/exadat");
```

Load an Impala data source using PROC CASUTIL.

```sas
proc casutil;
  incaslib="imlib"
sessref=mysess;
  load incaslib="imlib" casdata="cars" casout="cars_CAS"
options=(dbmsWhere="cylinders=8";
run;
quit;
```
## Syntax

### Data Connector Options for Impala

For each described option, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see “Where to Specify Data Connector Options” on page 147.

**AUTHENTICATIONDOMAIN="domain"**

specifies the name of the authentication domain that contains credentials (user name 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, then 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, then 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>

**CATALOG="catalog name"**

specifies a logical catalog name for data sources that do not natively support catalogs. 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>Default</td>
<td>active caslib</td>
</tr>
<tr>
<td>CAS action</td>
<td>addCaslib</td>
</tr>
<tr>
<td>Default</td>
<td>active caslib</td>
</tr>
<tr>
<td>PROC CASUTIL: CONTENTS, LIST, LOAD OPTIONS= statements</td>
<td></td>
</tr>
</tbody>
</table>

**CHARMULTIPLIER=value**

specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. For double-byte character sets, set CHARMULTIPLIER=2.0. 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>
<tr>
<td>Variable</td>
<td>Description</td>
</tr>
<tr>
<td>------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>CLIENT ENCODING= &quot;encoding&quot;</td>
<td>specifies the DBMS encoding type.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>DBMSWHERE= &quot;WHERE-clause&quot;</td>
<td>specifies a database- or data source-specific SQL WHERE clause to submit to the database or data source. The WHERE clause that you specify is passed to the database or data source exactly as you enter it. This option is used to filter the rows that are read into a CAS table.</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>DM_UNICODE= &quot;Unicode setting&quot;</td>
<td>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.</td>
</tr>
<tr>
<td>DRIVER_VENDOR= &quot;driver vendor name&quot;</td>
<td>specifies the name of the specific third-party vendor that supports the Impala driver.</td>
</tr>
<tr>
<td>IMPALA_DSN= &quot;Impala data source name&quot;</td>
<td>specifies the Impala data source name. With this option, you can configure an Impala data source in an odbc.ini file.</td>
</tr>
<tr>
<td>NAME= &quot;impala&quot;</td>
<td>specifies Impala as the data source.</td>
</tr>
</tbody>
</table>

Example: `s:loadDatasource(name='impala')`
**PASSWORD= "password"**
specifies the DBMS password for a user. You typically specify USER inmates 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 USER inmates and PASSWORD= value in a statement other than the CASLIB statement, then these credentials override any credentials that were specified when the caslib was added.

*Valid in*

CASMIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD statements

*Aliases*

PASS=

PWD=

*Default*

"" (zero-length string)

*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 USER inmates= and PASSWORD= values or by specifying an AUTHENTICATIONDOMAIN= value.

**PATH= "Impala table path and name"**
specifies the name of an Impala source table.

*Default*

none

**PORT= "port value"**
specifies the port to use to connect to the Impala database.

*Valid in*

CASMIB statement

*Default*

21050

*Range*

1–65535

**READBUFF= "integer"**
specifies the number of rows to fetch per block of data.

*Valid in*

CASMIB 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, then the table mySchema.Studydata is accessed. If you specify a value for SCHEMA= in a statement other than the CASLIB statement, then this value overrides any value of SCHEMA= that was set when the caslib was added.
Valid in  **CASLIB statement**

**PROC CASUTIL: CONTENTS, LIST, LOAD statements**

Default  "default"

**SERVER="Impala server identifier"**
specifies the server identifier for the Impala server.

Valid in  **CASLIB statement [required]**

Default  "" (zero-length string)

**SRCTYPE="impala"**
specifies that the data source is an Impala database.

Valid in  **CASLIB statement [required]**

Default  "" (zero-length string)

**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, then these credentials override any credentials that were specified when the caslib was added.

Valid in  **CASLIB statement [see Requirement]**

**PROC CASUTIL: CONTENTS, LIST, LOAD statements**

**Aliases**

UID=

USER=

Default  none

**Requirement**

If it is required by your database or data source, then 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 147.
**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 USERNAMNAME= 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 USERNUMNAME= 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 USERNAMNAME= and PASSWORD= in separate statements. For example, you might supply a USERNAMNAME= value in the CASLIB statement and then supply a PASSWORD= value when you specify a LOAD statement that accesses the data. In this situation, you must supply the PASSWORD= value for each statement that accesses the data.

**Load a Connection to an Impala Data Source**

When you load a connection to a data source, you specify the type of data source to which SAS Cloud Analytic Services is connecting. However, a connection is not made until you load data from the data source. To load the connection to the data source and to define a caslib that accesses that data source, use the CASLIB statement.

**Supported Impala Data Types**

This table shows the data types that can be read into SAS Cloud Analytic Services from an Impala database, along with the resulting data type and format when data is read into SAS Cloud Analytic Services. The length of the data format in SAS Cloud Analytic Services is based on the length of the source data.

<table>
<thead>
<tr>
<th>Impala Data Type</th>
<th>SAS Cloud Analytic Services Data Type</th>
<th>Default Format in SAS Cloud Analytic Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>w.</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
<td>w.</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>w.d</td>
</tr>
</tbody>
</table>

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.

Examples

**Example 1: Specify an Impala Database as a Data Source for a Caslib in SAS Cloud Analytic Services**

```plaintext
caslib imlib desc='Impala Caslib'
    dataSource=(srctype='Impala',
               server='impalaServer',
               username='myuser1',
               password='myPwd',);
```

**Example 2: Load an Impala Table into SAS Cloud Analytic Services Using PROC CASUTIL**

```plaintext
proc casutil;
    list files incaslib="Impalacaslib";
1  load casdata="myIMdata" incaslib="Impalacaslib" outcaslib="casuser" casout="IMdata_from_Impalacaslib";
2  list tables incaslib="casuser";
3  contents casdata="IMdata_from_Impalacaslib" incaslib="casuser";
4  quit;
```

1. List the tables in Impalacaslib before loading your data.
2. Load the table myIMdata from Hive 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.

**Example 3: Using Lua to Connect to Impala, Examine the Data, and Load a Table**

```lua
-- Load the Impala data source
r=s:loadDatasource{name='impala'}

-- Add the Caslib for Impala
r = s:addCaslib{lib='imlib',
               dataSource={srcType='impala',
                           server='Impala server name',
                           username='myuser1',
                           password='myPwd',
                           schema='Impala schema name'}
}

print("Calling fileInfo")
r = s:fileInfo{caslib='imlib'}
print(r)

print("Calling columnInfo")
r = s:columnInfo{table={caslib='imlib',
                       name='Impala table name'}}
```
print(r)

print("Calling loadTable")
r = s:loadTable{caslib='imlib', path='Impala table name'}
print(r)

---

**SAS Data Connector to ODBC**

SAS Data Connector to ODBC specifies the options to use when loading data from an ODBC data source into SAS Cloud Analytic Services. Data connector options are used in the context of different statements that connect your data in ODBC with CAS.

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

**Example:** Establish a connection to ODBC data. (Note: The GLOBAL option is restricted to administrators.)

```sas
caslib odbccaslib desc="SQLviaODBCtoCaslib"
  datasource=(srctype="odbc"
    username="user1"
    password="password1"
    odbc_dsn="dbodbc") global;
```

---

**Syntax**

**Data Connector Options for ODBC**

For each option that is described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see “Where to Specify Data Connector Options” on page 147.

**AUTHENTICATIONDOMAIN="domain"**

specifies the name of the authentication domain that contains credentials (user name 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, then these credentials override any that were set when the caslib was added.

**Valid in**
- CASLIB statement [see Requirement]
- PROC CASUTIL: CONTENTS, LIST, LOAD statements

**Alias**

AUTHDOMAIN=

**Requirement**

If your database or data source requires authentication, then 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 for data sources that do not natively support catalogs. The logical name can be any user-defined name. This name is displayed in
the Catalog column for all tables in the results from the CONTENTS statement in PROC CASUTIL.

For data sources that natively support multiple catalogs, such as Microsoft SQL Server or Netezza, specify CATALOG="*". Specifying CATALOG="*" enables all catalogs to be referenced.

Valid in
- CASLIB statement
- PROC CASUTIL: LOAD, LIST, and CONTENTS statements

Default
active caslib

CHARMULTIPLIER=value
specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in
- CASLIB statement
- PROC CASUTIL: LOAD statement

Default
1.0

Range
1.0–5.0

CLIENT_ENCODING="<encoding-value>
specifies the DBMS encoding of the source data. This value is independent of the NLS_LANG environment variable. For encodings other than UTF-8, the data is transcoded into UTF-8 when data is loaded into CAS.

Valid in
- CASLIB statement
- PROC CASUTIL: LOAD statement

Default
none

CONOPTS="<connection-options>"
specifies optional connection options that you can pass to the underlying DBMS.

Valid in
- CASLIB statement
- PROC CASUTIL: LOAD, LIST, and CONTENTS statements

Default
none

Example
To set the WSID option to MyComputerName for a Microsoft SQL Server driver, you might specify this code:

conopts="WSID=MyComputerName"

DBMSWHERE="WHERE-clause"
specifies a database- or data source-specific SQL WHERE clause to submit to the database or data source. The WHERE clause that you specify is passed to the database or data source exactly as you enter it. This option is used to filter the rows that are read into a CAS table.

Valid in
- PROC CASUTIL: LOAD statement
### 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

### NAME=

"odbc"

Specifies the name of a DBMS type.

**Default**

none

### ODBC_DSN="DSN-name"

Specifies the data source name for your DBMS.

When you specify a value for ODBC_DSN=, the table name is qualified with that name. For example, if you set ODBC_DSN="myDB" and you want to access table Studydata, then the table myDB.Studydata is accessed.

**Valid in**

- CASLIB statement [Required]

**Aliases**

- DATABASE=
- DB=

**Interaction**

If you supply values for SCHEMA= and ODBC_DSN=, then the value for SCHEMA= is used to qualify a table name.

### 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, then these credentials override any credentials that were specified when the caslib was added.

**Valid in**

- CASLIB statement [see Requirement]

**Aliases**

- PASS=
- PWD=

**Default**

"" (zero-length string)
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  
auto-calculated based on row size

**SCHEMA=** "schema-name"

specifies the schema name to use for the connection to the database.

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, then the table `mySchema.Studydata` is accessed.

If you supply a value for `SCHEMA= in a statement other than the CASLIB statement, then this value overrides any value of `SCHEMA=` that was set when the caslib was added.

Valid in  
- CASLIB statement
- PROC CASUTIL: CONTENTS, LIST, LOAD statements

Default  
"" (zero-length string)

**SRCTYPE=** "ODBC"

specifies that the data source for the caslib is accessed through ODBC.

Valid in  
- CASLIB statement [Required]

Default  
none

**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, then these credentials override any credentials that were specified when the caslib was added.

Valid in  
- CASLIB statement [see Requirement]
- PROC CASUTIL: CONTENTS, LIST, LOAD statements

Aliases  
UID=
USER=

Default  
none
Requirement

If it is required by your database or data source, then 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

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>
<thead>
<tr>
<th>ODBC Data Type</th>
<th>CAS Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL_CHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>SQL_VARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_LONGVARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_WCHAR</td>
<td>CHAR</td>
</tr>
<tr>
<td>SQL_WVARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_WLONGVARCHAR</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_BINARY</td>
<td>CHAR</td>
</tr>
<tr>
<td>SQL_VARBINARY</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_LONGVARBINARY</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>SQL_NUMERIC</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_DECIMAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_INTEGER</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_SMALLINT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_FLOAT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_REAL</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_DOUBLE</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_BIGINT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_TINYINT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_BIT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SQL_TYPE_DATE</td>
<td>DOUBLE (formatted as DATE)</td>
</tr>
<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>

**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 69.

```plaintext
  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**

```plaintext
  proc casutil;
    list files incaslib="odbccaslib";      /* 1 */
    load casdata="myDBdata" incaslib="odbccaslib" outcaslib="casuser" casout="class_from_odbcaslib";    /* 2 */
    list tables incaslib="casuser";        /* 3 */
    contents casdata="class_from_odbcaslib" 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_odbcaslib.
3 List the tables in casuser to see the newly created table, class_from_odbcaslib, that you loaded.
4 List information about the newly loaded table, including column names, data types, and so on.
SAS Data Connector to Oracle

SAS Data Connector to Oracle specifies the options to use when loading data from Oracle into SAS Cloud Analytic Services. Data connector options are used in the context of different statements that connect your data in Oracle with CAS.

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

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

caslib oraclecaslib desc='Oracle Caslib'
  datasource=(srctype='oracle'
    username='myusr1'
    password='myPwd'
    path='//machine.lnx.com:1521/exadat');

Override USER= and PASSWORD= values.

proc casutil;
  load casdata="%upcase(mycas.orexamp)"
  dataSourceOptions=(username='user5' password='pwd5');

Syntax

Data Connector Options for Oracle

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see “Where to Specify Data Connector Options” on page 147.

AUTHENTICATIONDOMAIN="domain"

specifies the name of the authentication domain that contains credentials (user name 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, then these credentials override any that were set when the caslib was added.

Valid in  CASLIB statement [see Requirement]
PROC CASUTIL: CONTENTS, LIST, LOAD statements

Alias  AUTHDOMAIN=

Requirement  If your database or data source requires authentication, then you must specify valid credentials to access data. You can provide these credentials by specifying either an AUTHENTICATIONDOMAIN= value or by specifying USERNAME= and PASSWORD= values.
CHARMULTIPLIER=value

specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in  
CASLIB statement

<table>
<thead>
<tr>
<th></th>
<th>PROC CASUTIL: LOAD 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>

DBMSWHERE="WHERE-clause"

specifies a database- or data source-specific SQL WHERE clause to submit to the database or data source. The WHERE clause that you specify is passed to the database or data source exactly as you enter it. This option is used to filter the rows that are read into a CAS table.

Valid in  
PROC CASUTIL: LOAD statement

<table>
<thead>
<tr>
<th></th>
<th>PROC CASUTIL: LOAD statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>&quot;&quot; (zero-length string)</td>
</tr>
<tr>
<td>Restriction</td>
<td>This option is not allowed when CAS column security is defined for a table. Also, you can use this option only on tables that do not have restrictions on column access. For more information, see “Loading a Subset of Table Rows”.</td>
</tr>
</tbody>
</table>

ORA_ENCODING="<encoding-name>"

specifies the encoding of the data in the Oracle database. This value is independent of the NLS_LANG environment variable setting. The data is transcoded from the database encoding into UTF-8 when data is loaded into CAS.

Valid values include LATIN1, WLATIN1, and UNICODE. Set this value to UNICODE when you are loading non-Latin1 data.

Tip If the source data is encoded in UTF-8, then data does not need to be transcoded into UTF-8 when loading it into CAS. This reduces the time it takes to load the data.

Valid in  
CASLIB statement

<table>
<thead>
<tr>
<th></th>
<th>PROC CASUTIL: LOAD statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>UNICODE</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, then these credentials override any credentials that were specified when the caslib was added.

Valid in  
CASLIB statement [see Requirement]

<table>
<thead>
<tr>
<th></th>
<th>PROC CASUTIL: CONTENTS, LIST, LOAD statements</th>
</tr>
</thead>
<tbody>
<tr>
<td>PROC CASUTIL: CONTENTS, LIST, LOAD statements</td>
<td></td>
</tr>
</tbody>
</table>
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=**"Oracle-database-name"

specifies the Oracle driver, node, and database. Aliases are required if you are using SQL*Net Version 2.0 or higher.

Valid in: **CASLIB statement**

PROC CASUTIL: CONTENTS, LIST, LOAD [Required] statements

Default: none

Example:
```
caslib oraclecaslib desc='Oracle Caslib'
  datasource=(srcType='oracle'
    username='myuser1'
    password='myPwd'
    path="/machine.lnx.com:1521/exadat");
```

**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, then the table `mySchema.Studydata` is accessed. If you specify a value for `SCHEMA=` in a statement other than the CASLIB statement, then this value overrides any value of `SCHEMA=` that was set when the caslib was added.

Valid in: **CASLIB statement**

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Default: "default"

**SRCTYPE=**"ORACLE"

specifies that the data source is an Oracle database.

Valid in: **CASLIB statement [Required]**

Default: none

**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, then these credentials override any credentials that were specified when the caslib was added.
Valid in | CASLIB statement [see Requirement]
---|---

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Aliases | UID=
---|---

USER=

Default | none

Requirement | If it is required by your database or data source, then 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

### Restrictions on Manipulating Oracle Data

When you load data from Oracle into SAS Cloud Analytic Services, these Oracle data types are not supported:

- array
- Boolean
- large object types (BLOB, CLOB)
- map
- struct
- union

### 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, Oraclelib. 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.

```sas
caslib oraclelib
datasource=(srctype='oracle',
  username='myuser1',
  password='myPwd',
  path="myORAdata");
```
Example 2: Load Oracle Data into SAS Cloud Analytic Services Using PROC CASUTIL

```sas
proc casutil;
  list tables incaslib="casuser";  \[1\]
  load casdata="MYORADATA" incaslib="oraclecaslib" outcaslib="casuser" casout="ORAdata_from_oraclecaslib";  \[2\]
  list tables incaslib="casuser";
  contents casdata="%upcase(class_from_oraclecaslib)" incaslib="casuser";  \[3\]
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_oraclecaslib.
   
   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_oraclecaslib, that you loaded.
4. List information about the newly loaded table, including column names, data types, and so on.

SAS Data Connector to PC Files

Specifies the options to use when loading data from files of various file types into SAS Cloud Analytic Services.

Applies to: **CASUTIL procedure**

See: Chapter 9, “Data Types.”

Example: Load the Titanic table into SAS Cloud Analytic Services. The 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;
```

Syntax

**Data Connector Options for PC Files**

*FILETYPE="DTA" | "EXCEL" | "JMP" | "SPSS" | "XLS"*

specifies the file type. Specify FILETYPE="excel" to work with XLSX files (.xlsx filename suffix).

<table>
<thead>
<tr>
<th>Default</th>
<th>Restriction</th>
</tr>
</thead>
<tbody>
<tr>
<td>XLS</td>
<td>Use XLSX for national language characters. These do not display correctly for XLS files.</td>
</tr>
</tbody>
</table>
**Requirement**
You do not need to specify a value to work with XLS files. However, you must specify a valid value before you can work with any other file type.

**GETNAMES=** **TRUE** | **FALSE**
when set to true, the values in the first line of the file are used as variable names.

**Default**  
TRUE

**PATH=** "table path and name"
specifies the name of a PC Files source table.

**Default**  
none

**RANGE=** "cell range"
specifies the range of data (a subset of cells) within the worksheet to import.

**Default**  
none (This imports the entire worksheet.)

**Requirement**
To use this option, you must specify a range that represents a rectangle in the worksheet, such as A17–D51. For XLS files, the range "Sheet1$A1:B5" is the range address for a rectangular block of 10 cells, where the top left cell is A1 and the bottom right cell is B5. For XLSX files (FILETYPE="excel"), do not include the worksheet name in the range.

**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.

**Default**  
none (This imports the entire worksheet.)

**Details**

**Caslib Type**
Working with Microsoft Excel files does not require any data source options, but you must use a caslib with a data source type of PATH.

**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.

Example: Use RANGE= with the CASUTIL Procedure to Specify Data to Import

This example shows how to use the RANGE= option to specify a block of cells to import. The file is available from http://catalog.data.gov/dataset/2010-federal-stem-education-inventory-data-set. One characteristic of the file is that the first two rows are used to provide column descriptions and the columns exceed SAS naming rules. The RANGE= option skips the first two rows. GETNAMES="false" specifies to read data in Row 3 as data values rather than column names.

```
proc casutil;
   load file="2010 Federal STEM Education Inventory Data Set.xls"
      casout="stem2010"
      importoptions=(filetype="xls" getnames="false" range="Sheet1$A3:IV254");
   /* Be sure that the data type is listed as DOUBLE
    * for Columns G, H, and I. */
   contents casdata="stem2010";
run;
```

```
libname mycas cas;
proc print data=mycas.stem2010(caslib="casuser" obs=2);
    var a -- j;
run;
```

Disclaimer: SAS might reference other websites, content, or resources for use at the Customer's sole discretion. SAS has no control over any websites or resources that companies or persons other than SAS provide. The Customer acknowledges and agrees that SAS is not responsible for the availability or use of any such external sites or resources, and does not endorse any advertising, products, or other materials on or available from such websites or resources. The Customer acknowledges and agrees that SAS is not liable for any loss or damage that the Customer or its end users might incur as a result of the availability or use of those external sites or resources, or as a result of any reliance that the Customer or its end users places on the completeness, accuracy, or existence of any advertising, products, or other materials on or available from such websites or resources.

---

SAS Data Connector to PostgreSQL

SAS Data Connector to PostgreSQL enables you to load data from PostgreSQL into SAS Cloud Analytic Services. Data connector options are used in the context of different statements that connect your data in PostgreSQL with CAS.

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

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

```
caslib postgrescaslib desc='PostgreSQL Caslib'
   dataSource=(srctype='postgres'
                 server='PGserver'
                 username='user1')
```
Overriding the user and password values.

```sas
proc casutil;
  load casdata="mycas.pgexamp" casout="myPGdata" casuser dataSourceOptions=(
    username='user5'
    password='myPwd');
quit;
```

**Syntax**

**Data Connector Options for PostgreSQL**

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see “Where to Specify Data Connector Options” on page 147.

**AUTHENTICATIONDOMAIN="domain"**

specifies the name of the authentication domain that contains credentials (user name 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, then these credentials override any that were set when the caslib was added.

Valid in CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Alias AUTHDOMAIN=

Requirement If your database or data source requires authentication, then 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 the name of the catalog to use for a connection.

Valid in CASLIB statement

Default caslib name

**CHARMULTIPLIER=value**

specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. For double-byte character sets, set CHARMULTIPLIER=2.0. This value overrides a value of CHARMULTIPLIER= that was set in the CASLIB statement.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement
CLIENT_ENCODING="encoding"
specifies the DBMS encoding in the PostgreSQL database. This value is independent of the NLS_LANG environment variable. For encodings other than UTF-8, the data is transcoded into UTF-8 when data is loaded into CAS.

Valid in  CASLIB statement
          PROC CASUTIL: LOAD statement

CONOPTS="connection-options"
specifies DBMS-specific connection options.

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

Default  "" (empty string)

DATABASE="PostgreSQL-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, then the table myDB.Studydata is accessed.

Valid in  CASLIB statement [Required]

Default  "" (empty string)

Interaction If you supply values for SCHEMA= and DATABASE=, then the value for SCHEMA= is used to qualify a table name.

DBMSWHERE="WHERE-clause"
specifies a database- or data source-specific SQL WHERE clause to submit to the database or data source. The WHERE clause that you specify is passed to the database or data source exactly as you enter it. This option is used to filter the rows that are read into a CAS table.

Valid in  PROC CASUTIL: LOAD statement

Default  "" (zero-length string)

Restriction This option is not allowed when CAS column security is defined for a table. Also, you can use this option only on tables that do not have restrictions on column access. For more information, see “Loading a Subset of Table Rows”.

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, then 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, LIST, LOAD statements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Aliases</th>
<th>PASS=</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>PWD=</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default</th>
<th>&quot;&quot; (zero-length string)</th>
</tr>
</thead>
</table>

| 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. |

<table>
<thead>
<tr>
<th>PTG_DSN=&quot;PostgreSQL-datasource-name&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>specifies the PostgreSQL data source name. This option enables you to use a configured PostgreSQL data source in an odbc.ini file.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Default</th>
<th>&quot;&quot; (empty string)</th>
</tr>
</thead>
</table>

| 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, then set the ODBCINI environment variable to the full path of the directory that contains the configured odbc.ini file. |

<table>
<thead>
<tr>
<th>READBUFF=&quot;/number-of-rows&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>specifies the number of rows to fetch per block of data.</td>
</tr>
</tbody>
</table>

<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>

<table>
<thead>
<tr>
<th>Default</th>
<th>calculated automatically based on row size</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>SCHEMA=&quot;/schema-name&quot;</th>
</tr>
</thead>
<tbody>
<tr>
<td>specifies the schema name to use for the connection to the database or data source.</td>
</tr>
</tbody>
</table>

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, then the table mySchema.Studydata is accessed. If you specify a value for SCHEMA= in a statement other than the CASLIB statement, then 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, LIST, LOAD statements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Default</th>
<th>&quot;default&quot;</th>
</tr>
</thead>
</table>

| Interaction | If you specify values for SCHEMA= and DATABASE=, then the value for SCHEMA= is used to qualify the table name. |

|          |          |          |          |          |          |          |          |          |          |          |          |          |          |          |
SERVER="PostgreSQL-server-identifier"
    specifies the server identifier for the PostgreSQL server.

Valid in  CASLIB statement [Required]

SRCTYPE="postgres"
    specifies that the data source is a PostgreSQL database.

Valid in  CASLIB statement [Required]

Default none

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, then these credentials override any credentials that were specified when
the caslib was added.

Valid in  CASLIB statement [see Requirement]  

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Aliases  UID=

USER=

Default none

Requirement  If it is required by your database or data source, then 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 8.6  Supported PostgreSQL Data Types

<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>PostgreSQL Data Type</td>
<td>CAS Data Type</td>
</tr>
<tr>
<td>---------------------------</td>
<td>--------------</td>
</tr>
<tr>
<td>CHARACTER(n)</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>TEXT</td>
<td>VARCHAR</td>
</tr>
<tr>
<td>INTEGER</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>BIGINT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>SMALLINT</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DOUBLE PRECISION</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>DECIMAL(p, s)</td>
<td>DOUBLE</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>TIME(p)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(without time zone)</td>
<td></td>
</tr>
<tr>
<td>TIMESTAMP(p)</td>
<td>DOUBLE</td>
</tr>
<tr>
<td>(without time zone)</td>
<td></td>
</tr>
<tr>
<td>JSON, JSONB</td>
<td>VARCHAR</td>
</tr>
</tbody>
</table>

### 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.

```sas
   caslib PostgreSQLcaslib desc='PostgreSQL Caslib' 
       dataSource=(srctype='postgres', 
       server='PGserver' 
       username='user1', 
       password='myPwd', 
       database='PGdatabase');
```
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.

SAS Data Connector to SAS Data Sets

Specifies the settings to use when loading data from SAS data sets into SAS Cloud Analytic Services.

Valid in: CASLIB statement

Applies to: CAS procedure, CASUTIL procedure

See: Chapter 9, "Data Types,"

Example: Load a password-protected SAS data set into SAS Cloud Analytic Services.

```sas
proc casutil;
   load casdata="salary" casout="salary"
      importoptions=(filetype="basesas" password="myPwd");
run;
```

Syntax

Data Connector Options for SAS Data Sets

CHARMULTIPLIER=double

specifies to increase the width of fixed-byte-width character columns. The number of bytes needed for multibyte characters depends on the particular characters in the string. Although specifying 1.5 is common, sometimes it is an overestimate and sometimes it truncates.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>importOptions option on the loadTable action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>1.0</td>
</tr>
</tbody>
</table>
Range 1.0–5.0

**COMPRESS=*/"YES | NO | BINARY | CHAR**
specifies the compression method to use.

Default NO

**DATATRANSFERMODE=*/"AUTO | PARALLEL | SERIAL**
specifies the mode of data transfer. This value overrides a value of DATATRANSFERMODE= that was set in the CASLIB statement.

Here are the valid values.

- **AUTO** specifies to first try to load the data in parallel (using embedded processing). If this fails, a note-level message is issued and serial processing is then attempted.
- **PARALLEL** specifies that the worker nodes should access and read data from the SAS data set.
- **SERIAL** specifies that the SAS Cloud Analytic Services controller node should access and read the SAS data set.

**Valid in** importOptions option on the loadTable action

**Aliases** DATATRANSFER=

**Default** AUTO

**Restriction** Data is loaded serially if only the controller has access to the SAS data set.

**Requirements**
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.

**Tip** Specify parallel if you know that all nodes have access to the data set. Specify serial if you know that only the controller has access to the data set. If you are not sure what type of access the nodes have to the data set, specify auto or do not specify a value for this option. In this case parallel is used if all nodes can access the data set. Otherwise, serial is used.

**ENCRYPT=*/"YES | NO | AES**
specifies the form of encryption to use to access the table.

**Valid in** CASLIB statement

**CAS actions: addCaslib**

**Default** NO

**ENCRYPTIONPASSWORD=*/"string**
specifies a passphrase to use for a data set that is encrypted using the Advanced Encryption Standard (AES).
Valid in: importOptions option on the loadTable action

Default: none

Restriction: You can specify this option only with the DATASOURCE= option.

Requirement: This option is required only if the SAS data set uses AES encryption. When you create a table, the encryption key must be in uppercase letters to ensure that the table can be read.

Supports: serial

FILETYPE="BASESAS"

specifies the file type.

Valid in: importOptions option on the loadTable action

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 filename.

PASSWORD="string"

specifies the password for a password-protected SAS data set.

Valid in: importOptions option on the loadTable action

Aliases: PASS=

PWD=

Default: none

Restriction: You can specify this option only with the DATASOURCE= option.

Requirements: This option is required only if the SAS data set uses AES encryption.

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 filenames. For more information, see “Using Wildcard Characters for Filename Matching” on page 101.

Valid in: CAS actions

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 filename or a subdirectory with a filename. 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.

**VARCHARCONVERSION=** _double_

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).

<table>
<thead>
<tr>
<th>Valid in</th>
<th>importOptions option on the loadTable action</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>none</td>
</tr>
</tbody>
</table>

**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 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. See the CASUTIL procedure for details about the LOAD CASDATA and LOAD DATA syntax.

**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";
```
SAS Data Connector to Teradata and SAS Data Connect Accelerator for Teradata

SAS Data Connector to Teradata enables you to load data serially from Teradata into SAS Cloud Analytic Services. All users can use SAS Data Connector to Teradata. SAS Data Connect Accelerator for Teradata is a separately licensed product that enables you to load data in parallel using the SAS Embedded Process. Data connector options are used in the context of different statements that connect your data in Teradata with CAS.

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

Examples:
Load a Teradata data source and add a caslib to it.
```sas
caslib tdlib
  datasource=(srctype='teradata'
    dataTransferMode='parallel'
    server='TDserver'
    username='user1'
    password='myPwd'
    database='TDdatabase');
```

Load Teradata source data using PROC CASUTIL.
```sas
proc casutil;
  load incaslib='tdlib' casdata='cars'
    casout='cars_CAS'
    options=(dbmsWhere='cylinders=8',
      dataTransferMode='parallel');
quit;
```

Syntax

Data Connector Options for Teradata

For each option described, the applicable statements where you can use that option are indicated. For information about where to specify these options within statements, see “Where to Specify Data Connector Options” on page 147.

AUTHENTICATIONDOMAIN="domain"

specifies the name of the authentication domain that contains credentials (user name 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, then 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, LIST, LOAD statements</td>
</tr>
<tr>
<td>Alias</td>
<td>AUTHDOMAIN=</td>
</tr>
<tr>
<td>Requirement</td>
<td>If your database or data source requires authentication, then 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>

BUFFERSIZE=bytes

specifies the buffer size length, in bytes, of the buffer that is used to receive data from SAS embedded processes.

This value overrides a value of BUFFERSIZE= that is 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 statement</td>
</tr>
<tr>
<td>Default</td>
<td>1048576</td>
</tr>
<tr>
<td>Restriction</td>
<td>This option applies only with SAS Data Connect Accelerator for Teradata (dataTransferMode=&quot;parallel&quot;).</td>
</tr>
</tbody>
</table>

CATALOG="catalog-name"

specifies a logical catalog name for data sources that do not natively support catalogs. 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>Default</td>
<td>active caslib</td>
</tr>
<tr>
<td>Restriction</td>
<td>This option applies only with SAS Data Connector to Teradata (dataTransferMode=&quot;serial&quot;).</td>
</tr>
</tbody>
</table>

CHARMULTIPLIER=value

specifies an increase to the width of fixed-byte-width character columns. The number of bytes that are needed for multibyte characters depends on the characters in a string. For double-byte character sets, set CHARMULTIPLIER=2.0. 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 statement</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 DBMS encoding in the Teradata database. This value is independent of the NLS_LANG environment variable. For encodings other than UTF-8, the data is transcoded into UTF-8 when data is loaded into CAS.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

DATABASE="Teradata-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, then the table myDB.Studydata is accessed.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Alias DB=

Default "" (empty string)

Interaction If you supply values for SCHEMA= and DATABASE=, then the value for SCHEMA= is used to qualify a table name.

DATATRANSFERMODE="AUTO | PARALLEL | SERIAL"
specifies the mode of data transfer. This value overrides a value of DATATRANSFERMODE= that was set in the CASLIB statement.

Here are the valid values.

AUTO specifies to first try to load the data in parallel (using embedded processing). If this fails, a note-level message is issued and serial processing is then attempted.

PARALLEL specifies to load the data in parallel by using the SAS Data Connect Accelerator to your database or data source.

SERIAL specifies to load the data serially by using the SAS Data Connector to your database or data source.

Valid in CASLIB statement

PROC CASUTIL: LOAD statement

Aliases DATATRANSFER=

DTM=

Default SERIAL

Requirement To use the PARALLEL option, you must have a licensed copy of the SAS Data Connect Accelerator for your database or data source.

Tip Specify parallel if you know that all nodes have access to the data set. Specify serial if you know that only the controller has access to the data set. If you are not sure what type of access the nodes have to the data set, specify auto or do not specify a value for this option. In this
case parallel is used if all nodes can access the data set. Otherwise, serial is used.

**DBMSWHERE**="WHERE-clause"

specifies a database- or data source-specific SQL WHERE clause to submit to the database or data source. The WHERE clause that you specify is passed to the database or data source exactly as you enter it. This option is used to filter the rows that are read into a CAS table.

Valid in  
PROC CASUTIL: LOAD statement

Default  
"" (zero-length string)

Restriction  
This option is not allowed when CAS column security is defined for a table. Also, you can use this option only on tables that do not have restrictions on column access. For more information, see “Loading a Subset of Table Rows”.

**NAME**="source-name"

specifies the type of data source.

**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, then these credentials override any credentials that were specified when the caslib was added.

Valid in  
CASLIB statement [see Requirement]

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Aliases  
PASS=

PWD=

Default  
"" (zero-length string)

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**="integer"

specifies the number of rows to fetch per block of data.

Valid in  
CASLIB statement

PROC CASUTIL: LOAD statement

Aliases  
RAS=

ROW_ARRAY_SIZE=

Default  
calculated automatically based on row size
**ROLE=**"name"

specifies the Teradata role name.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default</td>
<td>&quot;&quot; (empty string)</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, then the table mySchema.Studydata is accessed. If you specify a value for SCHEMA= in a statement other than the CASLIB statement, then 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>Default</td>
<td>&quot;default&quot;</td>
</tr>
</tbody>
</table>

**SERVER=**"Teradata-server-identifier"

specifies the server identifier for the Teradata server.

<table>
<thead>
<tr>
<th>Valid in</th>
<th>CASLIB statement [Required]</th>
</tr>
</thead>
</table>

**SRCTYPE=**"teradata"

specifies that the data source is a Teradata 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>

**STATUSINTERVAL=**number

specifies whether to print a message to the client when N buffers have been added to the table by a node. The value N is the value of this option.

This value overrides a value of STATUSINTERVAL= that is set in the 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>
</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, then these credentials override any credentials that were specified when the caslib was added.

**Valid in**
- CASLIB statement [see Requirement]
- PROC CASUTIL: CONTENTS, LIST, LOAD statements

**Aliases**
- UID=
- USER=

**Default**
- none

**Requirement**
If it is required by your database or data source, then 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.

#### 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.
Loading Data in Parallel

If the SAS Data Connect Accelerator for Teradata is installed on your system, you can use it to load data in parallel, by specifying `DATATRANSFERMODE="parallel"`. When you load data in parallel, the data connect accelerator uses the Teradata database’s hash distribution of the data to spread the data across multiple connections for parallel loading into CAS. The data distribution in the database is determined by the Teradata Primary Index (PI) for tables or by a hash that is calculated by the database for Teradata views. Talk to your Teradata administrator or review the Teradata user documentation to get more information about how Teradata distributes data across its units of parallelism (AMPs). The more evenly the data is divided, the more efficiently the data can be loaded into CAS.

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 8.7 Supported Teradata Data Types

<table>
<thead>
<tr>
<th>Teradata Data Type</th>
<th>CAS Data Type</th>
<th>Default Format in CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHAR</td>
<td>CHAR</td>
<td>$CHARw:</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>VARCHAR</td>
<td>$CHARw:</td>
</tr>
<tr>
<td>DOUBLE</td>
<td>DOUBLE</td>
<td>w.d</td>
</tr>
<tr>
<td>DATE (DA)</td>
<td>DOUBLE</td>
<td>DATEw:</td>
</tr>
<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>
<tr>
<td>INTEGER</td>
<td>DOUBLE</td>
<td>w.d</td>
</tr>
<tr>
<td>BYTEINT (I1)</td>
<td>DOUBLE</td>
<td>w.d</td>
</tr>
<tr>
<td>SMALLINT (I2)</td>
<td>DOUBLE</td>
<td>w.d</td>
</tr>
<tr>
<td>BIGINT (I8)</td>
<td>DOUBLE</td>
<td>w.d</td>
</tr>
</tbody>
</table>

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.
Examples

**Example 1: Specify a Teradata Database as a Data Source for a Caslib in SAS Cloud Analytic Services**

Use the CASLIB statement to initialize the data source and add the caslib for Teradata. No connection is made to the database until a statement that accesses the data is called.

The data is read serially into the caslib Teradatacaslib.

Teradata credentials are required to access the data. You can specify these in the CASLIB statement or when you use statements that access the data.

```sas
caslib TDlib desc='Teradata Caslib'
dataSource=(srctype='Teradata',
  dataTransferMode='serial',
  server='teradataServer',
  username='user1',
  password='******',
);
```

**Example 2: Load Teradata Data into SAS Cloud Analytic Services Using PROC CASUTIL**

```sas
proc casutil;
  list files incaslib="Teradatacaslib";       1
  load casdata="myTDdata" incaslib="Teradatacaslib" outcaslib="casuser"
    casout="TDdata_from_Teradatacaslib";  2
  list tables incaslib="casuser";            3
  contents casdata="TDdata_from_Teradatacaslib" incaslib="casuser";  4
quit;
```

1. List the tables in Teradatacaslib before loading your data.
2. Load the table myTDdata from Teradata into caslib Casuser. Call the new table TDdata_from_Teradatacaslib.
3. List the tables in caslib to see the newly created table, TDdata_from_Teradatacaslib, that you loaded.
4. List information about the newly loaded table, including column names, data types, and so on.
Chapter 9
Data Types

SAS Cloud Analytic Services Data Types

Overview

A data type is an attribute of every column in a table that specifies the type of data that the column stores. For example, the data type is the characteristic of a piece of data that indicates whether it is a character string, an integer, a floating-point number, a date, or a time. The data type also determines how much memory to allocate for the column value.

SAS Cloud Analytic Services currently supports the data types that are covered in this section, which support missing values.

Character Data

CHAR(n) stores a fixed-length character string, where \( n \) is the maximum number of characters to store. This maximum is required to store each value regardless of the actual size of the value. If CHAR(10) is specified and the character string is only five characters long, the value is right-padded with spaces.

Note: This data type cannot contain ANSI SQL null values.

VARCHAR(n) stores a varying-length character string, where \( n \) is the actual number of characters to store. If VARCHAR(10) is specified and the character string is only 5 characters long, the value is 5. It is not padded with spaces.

Note: This data type cannot contain ANSI SQL null values.

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 UTF8 session.

**Numeric Data**

DOUBLE
Stores a signed, approximate, double-precision, floating-point number. Allows numbers of large magnitude and permits computations that require many digits of precision to the right of the decimal point. For SAS Cloud Analytic Services, this is a 64-bit double precision, floating-point number.

**Data Types for SAS Cloud Analytic Services Table Columns**

<table>
<thead>
<tr>
<th>Data Type Definition</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>
Chapter 10
Functions

Dictionary

CLIBEXIST Function
returns 0 when the specified caslib name is not found and 1 when the caslib is found.

Category: CAS

Alias: CLIBFOUND

Syntax

CLIBEXIST(session name,caslib name)

Arguments

session name
a valid session name
caslib name
a valid caslib name.

Details

The CLIBEXIST function has access to sessions that are connected to the SAS client interfaces. To return a full set of caslibs that are known to the CAS server use the following:

caslib _ALL_ list;
Example

<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>

GETCASURL Function

Returns the value for a URL for connecting to the CAS Server Monitor.

**Category:** CAS

**Requirement:**
- The server name identified by the SAS CASHOST= option is used when constructing the URL.
- The value provided is a valid session name. If the value is not provided, the SAS SESSREF= option is valid.

**Syntax**

GETCASURL(<session>)

**Optional Argument**

*session*

if 0 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>caslib name=casuser(userid);</td>
</tr>
<tr>
<td>%put &quot;caslib name=&quot; %sysfunc(getlcaslib(mycas));</td>
<td></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>caslib name=mycaslib;</td>
</tr>
<tr>
<td>%put &quot;caslib name=&quot; %sysfunc(getlcaslib(mycas));</td>
<td></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>cas mysess UUIDMAC=sessuuid;</td>
<td></td>
</tr>
<tr>
<td>libname mycas cas;</td>
<td></td>
</tr>
<tr>
<td>%put &quot;session name=&quot; %sysfunc(getlsessref(mycas));</td>
<td>session name=mysess;</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>cas mysess UUIDMAC=sessuuid;</td>
<td></td>
</tr>
<tr>
<td>libname mycas cas tag=sometag;</td>
<td></td>
</tr>
<tr>
<td>%put &quot;tag=&quot; %sysfunc(getltag(mycas));</td>
<td>tag=sometag;</td>
</tr>
</tbody>
</table>

GETSESSOPT Function

Returns the value for a SAS Cloud Analytic Services session option.

Syntax

GETSESSOPT (session-name session-option-name)
Required Arguments

session-name
CAS session name.

session-option-name
CAS session option name.

For more information, see Chapter 12, “Session Options,” on page 209.

TIP You can list the session option names with this code:

CAS mysess LISTSESSOPTS;

Example: Listing the Active CASLIB

This example returns the CASLIB option value.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</th>
</tr>
</thead>
<tbody>
<tr>
<td>%put caslib = %sysfunc(GETSESSOPT(mysess, caslib));</td>
<td>x=CASUSERHDFS(userid)</td>
</tr>
</tbody>
</table>

   data work.one ;
   x = GETSESSOPT("mysess", "caslib") ;
   put x= ;
   run;

SESSFOUND Function

Returns a 0 when a CAS session is not connected to a server and a 1 when the session is connected to a server.

Syntax

SESSFOUND (session-name)

Required Argument

session-name
returns the name of the session that you are trying to find.

Details

This function has access to sessions that you started in your SAS session only. You can use the CAS statement with the LISTSESSIONS option to identify all your CAS sessions on a server.

When SAS writes data to the CAS server using the CAS engine, the engine estimates the number of bytes that are needed to transcode the data to UTF-8 based on the character set of the SAS session. SBCS environments estimate one byte in UTF-8 for every one byte in the local encoding. DBCS environments estimate 1.5 bytes in UTF-8 for every one byte in the local encoding. You can use the “CASNCHARMULTIPLIER= System Option” on page 229 system option to replace the estimate with an explicit value of the
byte multiplier when you know the number of bytes that are needed to represent the data in UTF-8.

The ENCODING= option identifies the character set.

**Example: Determining CAS Sessions**

This example shows whether the CAS session is found.

<table>
<thead>
<tr>
<th>Statements</th>
<th>Results</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));</code></td>
<td>doIExist= 1</td>
</tr>
<tr>
<td>run;</td>
<td></td>
</tr>
</tbody>
</table>
Chapter 11
Macro Variables

Dictionary

_CASHOST_ Macro Variable
Specifies the name of the CAS server.

Default: Not defined
Range: 256 characters
Interactions: When set, this macro variable overrides SAS system option CASHOST. The CAS statement HOST= option overrides this macro variable.
Tip: If you want to delete macro variable _CASHOST_ and allow SAS system option CASHOST to prevail, use the following statement:

`%symdel _CASHOST_;`

Syntax

```sas
%let _CASHOST_=cloud.example.com
```

_CASPORT_ Macro Variable
Specifies the CAS server port.

Default: Not defined
Range: 0–65535
Interactions: When set, this macro variable overrides SAS system option CASPORT. The CAS statement PORT= option overrides this macro variable.
Note: When set to 0, CAS selects a port number.
Tip: If you want to delete macro variable _CASPORT_ and allow SAS system option CASPORT to prevail, use the following statement:

%symdel _CASPORT_;

Syntax

%let _CASPORT_=5570

_SESSREF_ Macro Variable

Stores the name of the active SAS Cloud Analytic Services session.

Default: Not defined

Interactions: When you use the CAS statement to create a new session, this macro variable is automatically set to the name of the new session, which is CASAUTO by default. When you set SAS system option SESSREF, this macro variable is automatically set to the same value.
Chapter 12
Session Options

Setting Session Options

About the Session Options

The session options control various properties of your SAS Cloud Analytic Services session. To list the properties and their current setting for a session, use the LISTSESSOPTS option in a CAS statement. See “LISTSESSOPTS” on page 11. To see the setting for a specific property, use the GETSESSOPT function. See “GETSESSOPT Function” on page 204.
Setting Session Options for a New Session

How the Session Option Values Are Determined
When you create a new session, the value for each of the session properties is provided by the following sources in descending order of precedence:

• options specified in the SESSOPTS= option in the CAS statement.

• when you set SAS system options CASTIMEOUT and CASNWORKERS in SAS, CASTIMEOUT for session option TIMEOUT and CASNWORKERS for session option NWORKERS.

  Note: The CASTIMEOUT and CASNWORKERS system options have effect only after you set them in SAS. Otherwise, they are ignored.

• configuration parameters specified in the SAS Cloud Analytic Services configuration file

• command-line options that are used in the server start-up command

• SAS Cloud Analytic Services system defaults

You can browse the configured and default option values on the Configuration page of the CAS Server Monitor. For each option, this page shows the current value and the source of the value. The values for the session options are used as defaults for the session options, unless they are overridden. For information about the Server Monitor, see SAS Viya Administration: Using CAS Server Monitor.

Overriding the Default Session Option Values
To override the default session option values for a new session, use the option shown in the following table to complete the desired task.

<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>
<tr>
<td>Override the TIMEOUT option value for all subsequently created sessions.</td>
<td>SAS system option CASTIMEOUT=****</td>
</tr>
<tr>
<td>Override the NWORKERS option value for all subsequently created sessions.</td>
<td>SAS system option CASNWORKERS=*****</td>
</tr>
</tbody>
</table>

* TIMEOUT in the CAS statement SESSOPTS= option overrides this option.
** NWORKERS in the CAS statement SESSOPTS= option overrides this option.
*** Setting this option does not affect existing sessions.
### Setting Options for Existing Sessions

For an existing session, use the option shown in the following table to complete the desired task.

<table>
<thead>
<tr>
<th>Task</th>
<th>Option to use:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Change one or more session options for a specific session.</td>
<td>CAS statement SESSOPTS option</td>
</tr>
<tr>
<td>Change one or more session options for the active session.*</td>
<td>SAS system option CASSESSOPTS= (alias SESSOPTS=)</td>
</tr>
<tr>
<td>Specify a caslib for the active session.*</td>
<td>SAS system option CASLIB=</td>
</tr>
</tbody>
</table>

* SAS system option SESSREF stores the name of the currently active session.

### Session Options by Category

<table>
<thead>
<tr>
<th>Category</th>
<th>Language Elements</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Action</td>
<td>APPTAG= Session Option (p. 212)</td>
<td>specifies the string to prefix to log messages.</td>
</tr>
<tr>
<td>Caslib</td>
<td>CASLIB= Session Option (p. 213)</td>
<td>specifies the caslib name to set as the active caslib.</td>
</tr>
<tr>
<td></td>
<td>MAXTABLEMEM= Session Option (p. 220)</td>
<td>specifies the maximum amount of memory, in bytes, to allocate for a table.</td>
</tr>
<tr>
<td>Data Processing</td>
<td>EVENTDS Session Option (p. 217)</td>
<td>specifies one or more data sets that define events.</td>
</tr>
<tr>
<td></td>
<td>INTERVALDS Session Option (p. 218)</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>Data Quality</td>
<td>DQLOCALE Session Option (p. 217)</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. 217)</td>
<td>specifies the name of the default SAS Quality Knowledge Base (QKB) to use for data quality operations.</td>
</tr>
<tr>
<td>DATA step</td>
<td>DATASTEPFMTERR= Session Option (p. 215)</td>
<td>specifies how the DATA step reacts when SAS cannot find a specified variable format.</td>
</tr>
<tr>
<td></td>
<td>DATASTEPMESSGSUMLEVEL= Session Option (p. 216)</td>
<td>specifies the DATA step message summary level.</td>
</tr>
<tr>
<td>Category</td>
<td>Language Elements</td>
<td>Description</td>
</tr>
<tr>
<td>----------</td>
<td>------------------</td>
<td>-------------</td>
</tr>
<tr>
<td><strong>DATASTEPREPLACETABLE</strong></td>
<td>Session Option (p. 216)</td>
<td>specifies whether a DATA step can replace an existing table.</td>
</tr>
<tr>
<td><strong>Formats</strong></td>
<td>FMTCASLIB Session Option (p. 218)</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. 217)</td>
<td>specifies one or more data sets that define events.</td>
</tr>
<tr>
<td></td>
<td>INTERVALDS Session Option (p. 218)</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. 219)</td>
<td>specifies the locale to use for sorting and formatting.</td>
</tr>
<tr>
<td><strong>Log</strong></td>
<td>LOGFLUSHTIME= Session Option (p. 219)</td>
<td>specifies the log flush time, in milliseconds.</td>
</tr>
<tr>
<td></td>
<td>MESSAGELEVEL= Session Option (p. 220)</td>
<td>specifies the log message level.</td>
</tr>
<tr>
<td></td>
<td>METRICS= Session Option (p. 221)</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. 222)</td>
<td>specifies the number of worker nodes for a new session.</td>
</tr>
<tr>
<td></td>
<td>TIMEOUT= Session Option (p. 222)</td>
<td>specifies the SAS Cloud Analytic Services session time-out in seconds for a new or existing session.</td>
</tr>
<tr>
<td></td>
<td>TIMEZONE= Session Option (p. 223)</td>
<td>specifies the time zone offset, in hours, from UTC.</td>
</tr>
<tr>
<td><strong>Sort</strong></td>
<td>COLLATE= Session Option (p. 215)</td>
<td>specifies the collating sequence for sorting.</td>
</tr>
<tr>
<td><strong>System Administration: Performance</strong></td>
<td>CMPOPT= Session Option (p. 213)</td>
<td>specifies the type of code generation optimizations to use in the SAS language compiler.</td>
</tr>
</tbody>
</table>

**Dictionary**

**APPTAG= Session Option**

specifies the string to prefix to log messages.

**Valid in:** CAS statement SESSOPTS option

OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Action
Default: No prefix

Syntax

APPTAG="tag-string"

CASLIB= Session Option

specifies the caslib name to set as the active caslib.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Caslib
See: “CASLIB Statement” on page 69

Syntax

CASLIB="caslib-name"

CMPOPT= Session Option

specifies the type of code generation optimizations to use in the SAS language compiler.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: System Administration: Performance
Default: ALL

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. 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

Tip EXTRAMATH, MISSCHECK, PRECISE, GUARDCHECK, and FUNCDIFFERENCING can be specified in any combination.

Example Specify EXTRAMATH, MISSCHECK, and PRECISE:
\[ \text{cas casauto sessopts=\{cmpopt=\"extramath misscheck precise\"\};} \]

ALL
specifies that the compiler is to optimize the machine language code by using the NOEXTRAMATH, NOMISSCHECK, NOPRECISE, NOGUARDCHECK, and NOFUNCDIFFERENCING 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, and FUNCDIFFERENCING optimization values.

Restriction NONE cannot be specified with other values.
COLLATE= Session Option

specifies the collating sequence for sorting.

Valid in:  
CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category:  Sort
Default:  UCA

Syntax

COLLATE="MVA" | "UCA"

Parameter Values

MVA  
specifies SAS client collating.

UCA  
specifies a locale-appropriate collating sequence.

DATASTEPFMTERR= Session Option

specifies how the DATA step reacts when SAS cannot find a specified variable format.

Valid in:  
CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category:  DATA step
Default:  TRUE

Restriction:  This option applies only when the DATA step runs in SAS Cloud Analytic Services. See “Controlling Where the DATA Step Runs” in SAS Cloud Analytic Services: DATA Step Programming.

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 DATA step processing. This is the default.
FALSE
when SAS cannot find a specified variable format, it writes a note to the SAS log, substitutes the $w. or BEST12. format for the missing format, and then proceeds with DATA step processing.

**DATASETPMSGSUMLEVEL= Session Option**

specifies the DATA step message summary level.

**Valid in:** CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

**Category:** DATA step

**Default:** ALL

**Tip:** When the DATA step runs on multiple threads, the same message can be generated on each thread. In that case, use this option to control the summary level of the duplicate messages to help reduce the client log output.

**Syntax**

**DATASETPMSGSUMLEVEL=** 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.

**DATASETPREPLACE= Session Option**

specifies whether a DATA step can replace an existing table.

**Valid in:** CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function
Category: DATA step
Default: TRUE

**Syntax**

DATASTEPREPLACEABLE=TRUE | FALSE

---

**DQLOCALE Session Option**

specifies the default 5-letter SAS Quality Knowledge Base (QKB) ISO locale code to use as the default locale for data quality (DQ) operations.

**Valid in:**
- CAS statement SESSOPTS option
- OPTIONS statement CASSESSOPTS option
- GETSESSOPT function

**Category:** Data Quality

**Syntax**

DQLOCALE="5-letter-locale-code"

**Details**

For a list of the 5-letter QKB ISO locale codes, see QKB Locale ISO Codes.

---

**DQSETUPLOC Session Option**

specifies the name of the default SAS Quality Knowledge Base (QKB) to use for data quality operations.

**Valid in:**
- CAS statement SESSOPTS option
- OPTIONS statement CASSESSOPTS option
- GETSESSOPT function

**Category:** Data Quality

**Syntax**

DQSETUPLOC="QKB-name"

---

**EVENTDS Session Option**

specifies one or more data sets that define events.

**Valid in:**
- CAS statement SESSOPTS option
- OPTIONS statement CASSESSOPTS option
- GETSESSOPT function

**Categories:** Input Control
- Data Processing
The SAS predefined holiday events

Syntax

EVENTDS="event-data-set(s)"

Parameter Value

Event-data-set specifies the name of a data set that contains event definitions. You can use a one-level name or a two-level name such as libref.dataset. When specifying multiple names, separate each name with a space.

FMTCASLIB Session Option

specifies the caslib where persisted format libraries are retained.

Valid in: CAS statement SESSOPTS option
          OPTIONS statement CASSESSOPTS option
          GETSESSOPT function

Category: Formats
Default: FORMATS
Note: This option is set by the system administrator.

INTERVALDS Session Option

specifies one or more interval-name=value pairs, where the value is the name of a data set that contains user-defined intervals.

Valid in: CAS statement SESSOPTS option
          OPTIONS statement CASSESSOPTS option
          GETSESSOPT function

Categories: Input Control
            Data Processing
Interaction: This option overrides system option INTERVALDS for your session.
Tip: To specify interval data sets for all of your sessions, use the INTERVALDS system option instead.

Syntax

INTERVALDS="interval-1=libref.dataset-name-1 <interval-2=libref.dataset-
            name-2 ...>"

Parameter Value

Interval specifies the name of an interval. The value of interval is used to represent the set of intervals that is specified in data set libref.dataset-name. Libref.dataset-name specifies the libref and the data set name of the file that contains the user-defined intervals.
See Also

“INTERVALDS= System Option” in *SAS System Options: Reference*

---

**LOCALE= Session Option**

specifies the locale to use for sorting and formatting.

- **Valid in:** CAS statement SESSOPTS option
  OPTIONS statement CASSESSOPTS option
  GETSESSOPT function
- **Category:** Localization
- **Default:** en_US

**Syntax**

```
LOCALE="locale"
```

See Also


---

**LOGFLUSHTIME= Session Option**

specifies the log flush time, in milliseconds.

- **Valid in:** CAS statement SESSOPTS option
  OPTIONS statement CASSESSOPTS option
  GETSESSOPT function
- **Category:** Log
- **Default:** 100
- **Range:** –1–86400

**Syntax**

```
LOGFLUSHTIME=-1 | 0 | number
```

**Parameter Values**

- **–1**
  flushes logs after each action completes.
- **0**
  flush logs as they are produced.
- **number**
  flushes logs in *number* milliseconds.
MAXTABLEMEM= Session Option

specifies the maximum amount of memory, in bytes, to allocate for a table.

Valid in:  
- CAS statement SESSOPTS option
- OPTIONS statement CASSESSOPTS option
- GETSESSOPT function

Category: Caslib
Default: 16M

Note: After this threshold is reached, the server uses temporary files and operating system facilities for memory management.

Tip: You can enclose the value in quotation marks and specify B, K, M, G, or T as a suffix to indicate the units. For example, "8M" specifies eight megabytes.

Syntax

MAXTABLEMEM=number

Details

When data is added to the server (loading a table, appending rows, and so on) the server organizes the data in-memory blocks and allocates memory for the blocks. The MAXTABLEMEM= value is used to specify the amount of memory that each thread should allocate before converting to a memory-mapped file. Files are written in the directories that are specified in the CAS_DISK_CACHE environment variable. See SAS Cloud Analytic Services: Reference in SAS Viya Administration: SAS Cloud Analytic Services.

See Also

"Memory" in SAS Cloud Analytic Services: Fundamentals

MESSAGELEVEL= Session Option

specifies the log message level.

Valid in:  
- CAS statement SESSOPTS option
- OPTIONS statement CASSESSOPTS option
- GETSESSOPT function

Category: Log
Default: ALL

Syntax

MESSAGELEVEL="ALL" | "DEFAULT" | "ERROR" | "NONE" | "NOTE" | "WARNING"
**METRICS= Session Option**

specifies whether to include default detailed performance metrics reports in the SAS log.

- **Valid in:** CAS statement SESSOPTS option
  OPTIONS statement CASSESSOPTS option
  GETSESSOPT function

- **Category:** Log

- **Default:** FALSE

- **Example:** Enable metrics for session Casauto:

  ```
  cas casauto sessopts=(metrics=true);
  ```

**Syntax**

```
METRICS=TRUE | FALSE
```

**Details**

Session option METRICS= enables you to display information about the resources that your session consumes as each action in your program is executed. You can use the metrics information to track the resources that your session consumes and make adjustments, if necessary. By default, metrics are disabled for your session. Specify session option METRICS=TRUE to enable default metrics for your session. When enabled, after each action is executed, the metrics are written to the SAS log as notes. One note is written for each available metric. The following table lists the default metrics that are written to SAS the log when metrics are enabled.

<table>
<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>
</tbody>
</table>
Metric | Note Written to the SAS Log
--- | ---
The total available memory. | NOTE: total memory number<units>
The total number of nodes and cores in the cluster. | NOTE: total nodes number (number cores)

Here is an example of the default metrics that are written to the SAS log when the MDSUMMARY procedure is executed.

| NOTE: real time               | 0.126269 seconds |
| NOTE: cpu time                | 1.036837 seconds (821.13%) |
| NOTE: data movement time      | 0.013819 seconds |
| NOTE: total nodes             | 27 (1296 cores) |
| NOTE: total memory            | 6.65T |
| NOTE: memory                  | 57.25M (0.00%) |
| NOTE: bytes moved             | 4.80K |
| NOTE: The SAS Cloud Analytic Server processed the request in 0.126269 seconds. |
| NOTE: The data set MYCAS.MPGHW_SUM has 15 observations and 19 variables. |

Here is an example of the default metrics that are written to the SAS log when the MDSUMMARY procedure is executed.

NWORKERS= Session Option

specifies the number of worker nodes for a new session.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Session

Default: In order of descending precedence:
1. SAS system option CASNWORKERS, if you explicitly set it in SAS
2. 0 (all)

Range: 0–5000

Restriction: The number of workers can be set for new sessions only.

See: “CASNWORKERS= System Option” on page 230

Syntax

NWORKERS=number

TIMEOUT= Session Option

specifies the SAS Cloud Analytic Services session time-out in seconds for a new or existing session.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Session

Default: In order of descending precedence:
1. SAS system option CASTIMEOUT=, if you explicitly set it in SAS to a value greater than 0
2. 60

Range: 0–31536000

Notes: The session time-out starts when the number of connections to the session becomes zero and no actions are executing.

If a connection is established before the time-out expires, the time-out is canceled.

Otherwise, the session is automatically terminated when the time-out expires.

When set to 0, the session is terminated immediately when the connection count becomes zero and no actions are executing.

See: “CASTIMEOUT= System Option” on page 233

Syntax

TIMEOUT=number

TIMEZONE= Session Option

specifies the time zone offset, in hours, from UTC.

Valid in: CAS statement SESSOPTS option
          OPTIONS statement CASSESSOPTS option
          GETSESSOPT function

Category: Session

Default: −1

Range: −1–23

Syntax

TIMEZONE=number
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=

Alias: CASAUTHINFO=

Interaction: SAS Studio user credentials are used to authenticate the connection to CAS.

Syntax

AUTHINFO='authinfo_file_path';
Syntax Description

authinfo_file_path

specifies the path where an authinfo file is located. End users can store an encoded password in an authinfo file. The file provides an alternative to including passwords in programs. For more information, see “Create an Authinfo File” in Client Authentication Using an Authinfo File.

Details

A common use of this option is to submit code to CAS from the command line, in batch mode.

AUTHINFO= is also an environment variable and an option in the CAS statement. The environment variable can hold the name of one or more files. This variable is formatted like a PATH environment variable where a colon separates the filenames.

Here is the order of precedence to using the AUTHINFO options:
1. The AUTHINFO environment variable overrides the authinfo file.
2. The AUTHINFO= system option overrides the AUTHINFO environment variable.
3. The CAS statement AUTHINFO option overrides the AUTHINFO system option and the environment variable.

The shipped default for authorization file is $HOME/authInfo-file.

Examples

Example 1
Set AUTHINFO= system option. This option overrides the authinfo file pointed to by the AUTHINFO environment variable.

Options AUTHINFO='$HOME/authInfo-file';

Example 2
AUTHINFO can also be set as an environment variable. This option overrides the authinfo file.

Options insert=(set=AUTHINFO='$HOME/authInfo-file');

See Also

• “Create an Authinfo File” in Client Authentication Using an Authinfo File
• “CAS Statement” on page 1

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

\[
\text{CASDATALIMIT=} \text{integer} \mid \text{integer}K \mid \text{integer}M \mid \text{integer}G \mid \text{ALL}
\]

### 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 Option” on page 49
- “DATALIMIT= Data Set Option” on page 58

---

**CASHOST= System Option**

Specifies the CAS host name that is associated with a CAS session.

**Valid in:** SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window

- SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

**Category:** CAS
PROC OPTIONS
GROUP=

Note: This option can be restricted by a site administrator. For more information, see “Restricted Options” in SAS System Options: Reference.

---

Syntax

CASHOST= "host-name"

Syntax Description

*host-name*

specifies the CAS server name.

Default: None

Range: 1–256 characters

Example

options cashost="cloud.example.com";

---

CASLIB= System Option

Specifies the caslib name for the session that is identified by the SESSREF= option.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window
SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS
GROUP=

Interaction: The CAS statement session option CASLIB= overrides this option. For more information, see “CASLIB= Session Option” on page 213.

Note: This option can be restricted by a site administrator. For more information, see “Restricted Options” in SAS System Options: Reference.

Tip: A best practice is to explicitly set options for a session using the CAS statement SESSOPTS= option. Here is an example:

cas mysess sessopts=(caslib=mycaslib collate=UCA);

See “SESSOPTS=(session-option(s))” on page 13.

---

Syntax

CASLIB= "name"
Optional Argument

name

specifies the caslib name.

Default  None
Range  1–128 characters

Example: Set the Default Caslib

```plaintext
options caslib="casuser";
```

See Also

Statements:

- “CAS Statement” on page 1
- “CASLIB Statement” on page 69

CASNCHARMULTIPLIER= System Option

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

Valid in:
- SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window
- SAS Viya: Not supported

Category: CAS

PROC OPTIONS GROUP=

CAS

Defaults:
- For SBCS environments, the shipped default is 1.
- For DBCS environments, the shipped default is 1.5.

Restriction:
- This option is valid for SAS 9.4M5. In SAS Viya, use the NCHARMULTIPLIER= LIBNAME option or data set option.

Note:
- This option cannot be restricted by a site administrator. For more information, see “Restricted Options” in SAS System Options: Reference.

Syntax

CASNCHARMULTIPLIER=n

Syntax Description

n
specifies a number that is used as a multiplication factor to control the byte size for fixed character data that is transcoded to UTF-8 when the CAS engine writes data from SAS to the CAS server.
Details

When the SAS session encoding is not UTF-8, the CAS engine transcodes CHAR and VARCHAR values from the SAS session encoding to UTF-8. When DBCS characters and some SBCS characters are transcoded to UTF-8, they require additional bytes to represent a character. The CAS engine makes a best guess for the number of bytes that are needed to hold the transcoded character data. You can use the CASNCHARMULTIPLIER= option to replace the best guess with an explicit value of the byte multiplier when you know the number of bytes that are needed to represent the data in UTF-8.

Note: For example, when the SBCS characters for the accented characters in the Latin1 encoding are transcoded, they require additional bytes.

CAUTION: Data truncation can occur. If the value of CASNCHARMULTIPLIER= option is too small and the data is truncated, an error occurs. A best practice is to test reading DBCS data by the CAS engine before the data is used in a production environment.

See Also

• “NCHARMULTIPLIER= LIBNAME Option” on page 50
• “NCHARMULTIPLIER= Data Set Option” on page 60

CASNWORKERS= System Option

Specifies the number of worker nodes to use for a CAS session.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window
SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Default: ALL

Restriction: The number of worker nodes can be set for new sessions only during session creation.

Note: This option can be restricted by a site administrator. For more information, see “Restricted Options” in SAS System Options: Reference.

Tip: The CAS statement session option NWORKERS= overrides this option. For more information, see “CAS Statement” on page 1 and “NWORKERS= Session Option” on page 222.

Syntax

CASNWORKERS= ALL | number
**Syntax Description**

**ALL**

specifies to use all of the worker nodes.

**number**

specifies the number of worker nodes to use.

In SMP mode, *number* is always 0, whether you set CASNWORKERS= to 0 or 1.

In MPP mode, specify *number*=0 to use the maximum number of worker nodes that are available. You can set CASNWORKERS= to a number that is less than or equal to the maximum number of worker nodes.

Range: 0–5000

**Example**

```plaintext
options casnworkers=10;
```

---

**CASPORT= System Option**

Specifies the port to use when connecting to CAS.

**Valid in:**
- SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window
- SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

**Category:** CAS

**PROC OPTIONS GROUP=** CAS

**Default:** 0

**Note:** This option can be restricted by a site administrator. For more information, see “Restricted Options” in SAS System Options: Reference.

---

**Syntax**

**CASPORT=port-number**

**Syntax Description**

**port-number**

specifies the CAS server port number.

Range: 0–65535

**Note**

When *port-number* is set to 0, CAS selects a port number.

**Example**

```plaintext
options casport=12345;
```
CASSESSOPTS= System Option

Specifies one or more session options for the active CAS session.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window
          SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS
GROUP=

Alias: SESSOPTS=

Default: None

Note: This option can be restricted by a site administrator. For more information, see “Restricted Options” in SAS System Options: Reference.

Tip: A best practice is to explicitly set options for a session using the CAS statement SESSOPTS= option. Here is an example:

    cas mysess sessopts=(caslib=mycaslib collate=UCA);

See “SESSOPTS=(session-option(s))” on page 13.

Syntax

CASSESSOPTS=(session-option(s))
SESSOPTS=(session-option(s))

Syntax Description

session-option(s)

specifies one or more session options as option=value pairs separated by a space and enclosed in parentheses.

Tip To reflect a session option value, use this statement:

    %put caslib=%sysfunc(GETSESSOPT(session, option));

See Chapter 12, “Session Options,” on page 209 for a list of the options that you can specify for session-option(s).

Example

For the default session, set the caslib to MYCASLIB and the session connection time-out to 60 minutes:

    options sessopts=(caslib="mycaslib" timeout=3600)

See Also

Functions:
CASTIMEOUT= System Option

Specifies the CAS session time-out in seconds for new sessions. The session time-out starts when the number of connections to the session becomes zero and all session activity is complete.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window
SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS GROUP= CAS

Default: 60

Interaction: The TIMEOUT= session option overrides this option. For more information, see “TIMEOUT= Session Option” on page 222.

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 | nK | 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 seconds.
MAX
sets the time out value to 31536000.

MIN
sets the time out value to 0.

Details
If a connection is established before the time-out expires, the time-out is canceled. Otherwise, the session is automatically terminated when the time-out expires.

Example
options castimeout=28800;

CASUSER= System Option
Specifies the user ID to use when connecting to CAS.

Valid in:
- SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window
- SAS Viya: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS
PROC OPTIONS GROUP=

Alias: CASUSERID=
Default: None

Note: This option cannot be restricted by a site administrator. For more information, see “Restricted Options” in SAS System Options: Reference.

Syntax

CASUSER= user-ID
CASUSERID= user-ID

Syntax Description

user-ID
specifies your user ID.

Requirement
The user ID that you specify must match a user ID in your personal .authinfo file. For more information about the .authinfo file, see “Authinfo File Requirement” in SAS Viya: System Programming Guide.

Interactions
SAS Studio user credentials are used to authenticate your connection to CAS. After you are logged in to SAS Studio, you can submit code to CAS without using this option. The most frequent use of this option is to submit code to CAS from the command line, in batch mode.
The CAS statement USER= option overrides the user ID specified by this option. If the user ID is not specified in the CAS statement, SAS looks for a user ID that is set by the CASUSER= option. For more information, see “USER= user-ID” on page 13.

Example

options casuser=myid;

DSCAS System Option

Specifies whether the DATA step runs in the CAS server or in SAS by default.

Valid in: SAS 9.4: Configuration file, SAS invocation, OPTIONS statement, SAS System Options window
SAS Viya: Not supported

Categories: Environment Control: Language Control
CAS

PROC OPTIONS GROUP=

LANGUAGECONTROL

Default: The shipped default is DSCAS.

Restriction: This option is valid for SAS 9.4M5.

Interaction: If the SESSREF option is specified in the DATA statement, SAS always attempts to run the DATA step in CAS, regardless of the value of the DSCAS system option.

Note: This option can be restricted by a site administrator. For more information, see “Restricted Options” in SAS System Options: Reference.

Syntax

DSCAS | NODSCAS

Syntax Description

DSCAS
specifies that by default, the DATA step runs in the CAS server.

NODSCAS
specifies that by default, the DATA step runs in SAS.

See Also

“Controlling Where the DATA Step Runs” in SAS Cloud Analytic Services: DATA Step Programming

SESSREF= System Option

Specifies the name of the default CAS session to use if a statement or procedure does not explicitly identify a session reference name.
Valid in: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

Category: CAS

PROC OPTIONS
GROUP=

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 1.

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 1
Gain Greater Insight into Your SAS® Software with SAS Books.

Discover all that you need on your journey to knowledge and empowerment.

support.sas.com/bookstore for additional books and resources.