SAS® Cloud Analytic Services 3.1: Language Reference

SAS® Documentation

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2016. *SAS® Cloud Analytic Services 3.1: Language Reference*. Cary, NC: SAS Institute Inc.

SAS® Cloud Analytic Services 3.1: Language Reference

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

October 2016

3.1-P1:casref

Contents

About This Book

Audience

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

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

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

CAS server information options

LISTABOUT
: lists information about SAS Cloud Analytic Services.

LISTSERVERSTARTOPTS
: lists the SAS Cloud Analytic Services server 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 the SAS Cloud Analytic Services server listens for client connections.

RECONNECT

reconnects to a session using a session name.

TERMINATE
terminates the SAS Cloud Analytic Services session.

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

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
lists information about a session in your SAS client.

LISTHISTORY <*history_count* | _ALL_>
prints to the SAS log the action history for the current session.

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

Session property options

LISTSESSOPTS
lists the session options.

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

User-defined format options

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

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

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

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

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

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

LISTFMTSEARCH
lists the format library search order for the session.

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

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

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

SAVEFMTLIB FMTLIBNAME=*format-library-name* TABLE=*table-name* <CASLIB=*caslib*> <REPLACETABLE>

SAVEFMTLIB FMTLIBNAME=*format-library-name* PATH=*path*

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 created, and macro variable _SESSREF_ and SAS system option CASNAME (alias SESSREF) are set to *session-name*.

Optional Arguments

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

ADDFMTLIB FMTLIBNAME=*format-library-name* PATH=*path* <POSITION=APPEND | INSERT | REPLACE | NONE> <PROMOTE> <REPLACEFMTLIB>

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

FMTLIBNAME=*format-library-name*

specifies the name of the format library to add.

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

TABLE=*table-name*

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

Note Do not specify this option with PATH=.

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

Examples Add format library myFmtLib, which is stored in table myFmtLib in caslib CASFMTS, to session Casauto:

```
cas casauto addfmtlib fmtlibname="myFmtLib"
   table="myFmtLib" caslib=casfmts;
```

Add format library myFmtLib, which is stored in file **`/user/sasdemo/formats/myFmtLibs`**, to session Casauto and promote it to global scope:

```
cas casauto addfmtlib fmtlibname="myFmtLib"
   path="/user/sasdemo/formats/myFmtLib"
   promote;
```

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.

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

Default File $HOME/.authinfo on Linux hosts.

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

See *SAS Viya Administration: Authentication*

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 7: Connect to an Existing Session" on page 20.

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 executing. After the time-out expires, the session is terminated.

See "Example 6: Disconnect from a Session" on page 19

DROPFMTLIB FMTLIBNAME=*format-library-name* <FMTSEARCHREMOVE>
drops a session-scope or a global-scope format library. If the format library has both session scope and global scope, the format library is dropped from session scope. To

drop the format library from global scope in that case, execute the **CAS *session-name* DROPFMTLIB** statement again.

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

FMTSEARCHREMOVE
removes the format library from the format search order.

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

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

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

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

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

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.

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

See "_CASHOST_ Macro Variable" on page 175

"CASHOST= System Option" on page 191

LIST

lists information about a session in your SAS client. The information includes the session name, the session state, the host and port of the SAS Cloud Analytic Services server 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 17

LISTABOUT

lists information about SAS Cloud Analytic Services. The information is written to the SAS log and is organized as shown in the following table.

Section	Field	Description
About	CAS	Product name: SAS Cloud Analytic Services.
	Version	Short-form SAS Cloud Analytic Services version.
	VersionLong	Long-form SAS Cloud Analytic Services version.
	Copyright	Copyright information.
System	Hostname	SAS Cloud Analytic Services host name.
	OS Name	SAS Cloud Analytic Services host information.
	OS Family	
	OS Release	
	OS Version	
	Model Number	
	Documentation	SAS Cloud Analytic Services documentation URL.

Section	Field	Description
License	Site	Site information.
	SiteNum	
	Expires	License expiration date and time.
	GracePeriod	Grace period in days.
	WarningPeriod	Expiration warning period in days.
	Setinit	Path to the SETINIT file.

LISTFMTRANGES FMTNAME=*format-name*
lists the ranges 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 order. The ranges are listed for the first instance of *format-name* that is found.

Example List the ranges in format MYFORMAT:

```
cas casauto listfmtranges fmtname=myformat;
```

LISTFMTSEARCH
lists the format library search order for the session.

Example List the format library search order for session Casauto:

```
cas casauto listfmtsearch;
```

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 SESSION

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

Example List the global and session-local formats and their members for session Casauto:

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

LISTHISTORY <*history_count* | _ALL_>

prints to the SAS log the action history for the current session. For information about actions, see "Programming with CAS Actions" in *An Introduction to SAS Viya Programming for SAS 9 Programmers*.

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 server 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

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

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

See "Example 3: List Information about All of Your Sessions" on page 18

Example Use session Casauto to list all of your sessions.

```
cas casauto listsessions;
```

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

LISTSESSOPTS
lists the session options.

See Chapter 12, "Session Options," on page 177 for a list of the session options.

"Example 4: List the Properties for a Session" on page 18

Example List the session option settings for session Casauto:

```
cas casauto LISTSESSOPTS;
```

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

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

Range 1-65535

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

See "_CASPORT_ Macro Variable" on page 175

"CASPORT= System Option" on page 194

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.

Example Promote session-local format library MYFMTLIB in session Casauto:

```
cas casauto promotefmtlib fmtlibname=myfmtlib;
```

RECONNECT
reconnects to a session using a session name.

Tip If the session is not known to SAS, connect using the session UUID. Do not specify the RECONNECT keyword in that case. See "UUID="*session-uuid*"" on page 11.

See "Example 7: Connect to an Existing Session" on page 20

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

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

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

Note Do not specify this option with PATH=.

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

Note Do not specify this option with PATH=.

REPLACETABLE
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=.

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 177 for a list of the options that you can specify for *session-option(s)*.

"Example 5: Change a Property for a Session" on page 19

"Program: Create a Session with Custom Properties" on page 16

TERMINATE
terminates the SAS Cloud Analytic Services session.

Alias CLEAR

Note When you terminate the last session that was created, SAS system option CASNAME (alias SESSREF) and macro variable _SESSREF_ are not updated. They continue to reference the terminated session until a new session is created or they are manually set. See "CASNAME= System Option" on page 192.

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

See "Example 8: Terminate a Session" on page 21

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

Alias	CASUSER
Default	SAS system option CASUSER=, if set.
Requirement	*User-ID* must match a user ID in your authinfo file. See *SAS Viya Administration: Authentication*.
Interaction	This option overrides SAS system option CASUSER=.
Note	When you use SAS Studio, the user credentials that you used to sign on are used to authenticate your connection to CAS. The USER= option is not needed in that case. The USER= option or SAS system option CASUSER= is used when submitting code to CAS from the command line, in batch mode.
See	"CASUSER= System Option" on page 195

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 6
	"Program: Connect to an Existing Session Using the Session Name and UUID" on page 20

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: Create a Session and Store the UUID in a Macro Variable" on page 16

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
- create a session
- terminate a session

Creating Your Initial Session

After you sign in to SAS Studio, you must create a session in order to connect to the SAS Cloud Analytic Services server. After you create 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 create a session. Alternatively, you can submit your own CAS statement to create a customized session. See "Example 1: Create a Session" on page 16 for examples. Use the CAS statement to perform management tasks on your session, such as listing or changing properties, managing format libraries, and so on.

CAS Statement Status

When you execute a CAS statement for the first time in your SAS session, global macro variable CASSTMTERR is created. It is set to 0 if the CAS statement was successful or to 2 if an error occurred. 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 6: Disconnect from a Session" on page 19.

Managing Sessions with 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 171.

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

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
- Session Cannot Be Resolved
- Session Is Not Recognized
- Session Connection for a Session Is Not Active

- Unable to Connect to Cloud Analytic Services

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

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

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

Note: The warning message might not appear.

Cause	Solution
A reference to session *session-name* was made in the CASUTIL procedure SESSREF= option but the specified session is disconnected, does not exist, or is not known to SAS.	• If a warning message indicates that *session-name* is disconnected, reconnect to session *session-name*. See "Example 7: Connect to an Existing Session" on page 20. • Verify that session *session-name* exists. See "Example 3: List Information about All of Your Sessions" on page 18. If session *session-name* does not exist, specify a different session or create session *session-name*. See "Example 1: Create a Session" on page 16.

Could Not Find Netrc or Authinfo File

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

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

Missing Session Information

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

ERROR: Missing session information.
```

Note: The warning message might not appear.

Cause	Solution
A reference to session *session-name* was made in a LIBNAME statement, but the specified session is disconnected, does not exist, or is not known to SAS.	• If a warning message indicates that *session-name* is disconnected, reconnect to session *session-name*. See "Example 7: Connect to an Existing Session" on page 20. • Verify that session *session-name* exists. See "Example 3: List Information about All of Your Sessions" on page 18. If session *session-name* does not exist, specify a different session or create session *session-name*. See "Example 1: Create a Session" on page 16.

Request to Connect Failed for UUID

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

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

Note: The authentication error message might not appear.

Cause	Solution
An attempt was made to connect to a session using its name and UUID, but the session was not found or a connection could not be established.	• If authentication failed, verify that your credentials are correct. For information about authentication in SAS Cloud Analytic Services, see *SAS Viya Administration: Authentication*. • Verify that session *session-uuid* exists. See "Example 3: List Information about All of Your Sessions" on page 18. If session *session-uuid* does not exist, specify a different session UUID or create a new session. See "Example 1: Create a Session" on page 16.

Session Cannot Be Resolved

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

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

Note: The warning message might not appear.

Cause	Solution
A reference to session *session-name* was made in the CAS procedure SESSION statement, but the specified session is disconnected, does not exist, or is not known to SAS.	• If a warning message indicates that *session-name* is disconnected, reconnect to session *session-name*. See "Example 7: Connect to an Existing Session" on page 20. • Verify that session *session-name* exists. See "Example 3: List Information about All of Your Sessions" on page 18. If session *session-name* does not exist, specify a different session or create session *session-name*. See "Example 1: Create a Session" on page 16.

Session Is Not Recognized

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

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

Note: The warning message might not appear.

Cause	Solution
A request was made on session *session-name*, but session *session-name* is disconnected, does not exist, or is not known to SAS.	• If a warning message indicates that *session-name* is disconnected, reconnect to session *session-name*. See "Example 7: Connect to an Existing Session" on page 20. • Verify that session *session-name* exists. See "Example 3: List Information about All of Your Sessions" on page 18. If session *session-name* does not exist, specify a different session or create session *session-name*. See "Example 1: Create a Session" on page 16.

Session Connection for a Session Is Not Active

An error message similar to the following appears:

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

Cause	Solution
A request such as LISTSESSOPTS or LISTFORMATS was made on session *session-name*, but session *session-name* is disconnected.	Reconnect to session *session-name*. See "Example 7: Connect to an Existing Session" on page 20.

Unable to Connect to Cloud Analytic Services

One of the following error messages appears:

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

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

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

Examples

Example 1: Create a Session

Program: Create a Session with Default Properties

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

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

SAS Log

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

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

Program: Create a Session with Custom Properties

This example creates session Casauto with caslib CASUSER and with metrics enabled:

```
/* 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: Create a Session and Store the UUID in a Macro Variable

This example creates session Casauto and stores the session UUID in macro variable CASAUTO_UUID for later use:

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

SAS Log

The %PUT 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. See "CASLIB Statement" on page 39 and "CAS LIBNAME Statement" on page 24.
- 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, your session is automatically terminated if a connection is not established within the time-out period. See TIMEOUT= on page 187.
- Always terminate your session when you are finished with it.

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

Program: List a Specific Session in Your SAS Client

This example lists information about SAS client session Casauto:

```
cas casauto list;
```

SAS Log

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

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

Program: List All of the Sessions in Your SAS Client

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

```
cas _all_ list;
```

SAS Log

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

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

Example 3: List Information about All of Your Sessions

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

```
cas casauto listsessions;
```

SAS Log

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

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

Example 4: List the Properties for a Session

Program

This example lists the properties for session Casauto:

```
cas casauto listsessopts;
```

SAS Log

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

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

Additional Information

For information about the session properties, see Chapter 12, "Session Options," on page 177. The GETSESSOPT function enables you to get the value of a single property. See "GETSESSOPT Function" on page 171.

Example 5: Change a Property for a Session

Program

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

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

Additional Information

For information about the session properties that you can change, see Chapter 12, "Session Options," on page 177. 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: Disconnect from a Session

Before You Disconnect from Your Session

If no other client connection exists 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 7: Connect to an Existing Session" on page 20.

Program

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

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

SAS Log

The following notes are written to the SAS log:

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

Example 7: 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 created 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 created previously in the current SAS session:

```
cas casauto reconnect;
```

SAS Log

The following note is written to the SAS log:

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

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

If you want to connect to a session that you created in a different SAS session, you must specify the session name and UUID to connect. If you do not know the UUID of the session, use the LISTSESSIONS= option as described in “Example 3: List Information about All of Your Sessions” on page 18. This example uses session name Mysess and the UUID option to connect to session Mysess, which was created in a different SAS session.

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

SAS Log

The following notes are written to the SAS log.

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

Example 8: Terminate a Session

Program

Terminate session Casauto:

```
cas casauto terminate;
```

SAS Log

The following notes are written to the SAS log:

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

Chapter 2

CAS LIBNAME Statement

What You Can Do with the CAS LIBNAME Engine

You can use the CAS LIBNAME engine to work with SAS Cloud Analytic Services sessions, create a new session, connect to an existing session, or reference SAS Cloud Analytic Services libraries.

You can also use it to work with SAS Cloud Analytic Services tables—to add, delete, replace, or append to them—or you can run SAS procedures on them.

When you use this engine with a library in SAS Cloud Analytic Services, the active caslib is used unless you specify another caslib using the CASLIB= LIBNAME option. You can override both the active caslib and the CASLIB= LIBNAME option with the CASLIB= data set option.

You can also specify options on the SAS Cloud Analytic Services LIBNAME statement or in a DATA step. Specify a data set option in parentheses after the table name. When

you specify a value for a data set option that has a corresponding CAS LIBNAME statement option (such as PROMOTE=), the data set option value takes precedence over the value for the CAS LIBNAME statement option.

Dictionary

CAS LIBNAME Statement

Associates a SAS libref with tables on the SAS Cloud Analytic Services server.

Valid in: CAS LIBNAME engine

Category: Data Access

Default: none

Requirement: For a valid connection, you must specify only one of these mutually exclusive methods:

- host= and port=
- sessref=
- uuid or uuidmac

Data type: Supports varying-length data types

See: CASLIB statement, COMPRESS= LIBNAME option

Syntax

Form 1: **libname** *libref* cas (host="*controller-host-name*" port="*network-port-number*") | *options*;

Form 2: **libname** sessref=*session-name* | *options*;

Form 3: **libname** *libref* cas (uuid=*identifier* | uuidmac=*macro_variable*) | *options*;

Form 4: **libname** *libref* cas;

Required Arguments

libref
specifies a valid SAS name that serves as a shortcut name to associate with the tables on the SAS Cloud Analytic Services server. The name must conform with SAS naming conventions. It can be up to eight bytes long and is the handle to the SAS Cloud Analytic Services library or session.

cas
specifies the CAS LIBNAME engine.

host
specifies the machine for the SAS Cloud Analytic Services server control node. It is recommended that you specify a host. Otherwise, the engine checks for a value in the CASHOST= system option and then in the _CASHOST_ macro variable. If neither of these contain a value, it uses the server on the local machine (`host="localhost"`).

Alias server=

port=*network-port-number*
specifies an integer for the port where the SAS Cloud Analytic Services control node listens for client connections. It is recommended that you specify a port. Otherwise, the engine checks for a value in the CASHOST= system option and then in the _CASHOST_ macro variable.

Optional Arguments

caslib=*caslib*
specifies the name of a caslib to bind to the libref. By default, engine operations with the libref use the active caslib from your session. The active caslib can change as a result of adding or dropping caslibs. Use this option to specify an alternative caslib. 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.

sessref=*session-name*
specifies the name of the SAS Cloud Analytic Services session to which you want to connect. Using this option is a preferred alternative to HOST= and PORT=.

uuid=*identifier*
specifies a universally unique identifier (UUID) as a name or a quoted string. If you specify this option, the engine connects to the SAS Cloud Analytic Services session as identified in the UUID.

Tip Most people prefer to use sessref= instead of uuid=.

Example This example starts a SAS Cloud Analytic Services session through the CAS statement, saves the UUID in the MyUUID macro variable, connects the CAS LIBNAME to that session, and loads the FOO table into it.

```
cas mysess port=19999 host="mycashost001" UUIDMAC=MyUUID;

< Perform some additional tasks. >
libname mycas cas host="mycashost001" port=19999 uuid="&MyUUID";

data mycas.foo;
   < some DATA step code >
run;
```

uuidmac=*macro-variable*
specifies a SAS macro variable as a name or as a quoted string. If you specify this option, the engine saves the UUID of the last SAS Cloud Analytic Services session that it created. This option is useful if you want a subsequent SAS step to connect to a session that the CAS LIBNAME engine created or modified. If you do not specify this option, the UUID is stored in the _IOCASUUID_macro variable.

Details

Use of WHERE with the CAS LIBNAME Engine

If you use the WHERE clause or statement with the CAS LIBNAME engine, it is resolved on the SAS Cloud Analytic Services server if it is able to handle it. Otherwise, records are pulled back to the SAS client. In this case, a note is written to the log only if SAS Cloud Analytic Services processed the WHERE clause or statement and you specified MSGLEVEL=i. For more information about WHERE processing, see *SAS System Options: Reference*.

Connect to a Session or a Library

The CAS LIBNAME engine can connect you to an existing SAS Cloud Analytic Services session through the session name or the session UUID. The LIBNAME then becomes your handle to communicate from MVA SAS with the specific session.

You can also point the engine at a SAS Cloud Analytic Services library that contains data more globally visible to users who are authorized to use that library. In this case, the engine creates separate sessions for actions on the global tables and terminates the sessions once actions complete.

Example: Execute a DATA step

Once you have established the LIBNAME, you can run a DATA step as you would for any other data source. It is important to understand that this DATA step does not execute *in* the server. It executes on the SAS client, and the rows that the DATA step generates are sent to the server to create a new table there.

This example loads SASHELP.CARS into the server, as identified by libref mycas.

```
libname mycas cas host=mycashost port=myport;

   data mycas.foo;
      set sashelp.cars;
   run;

   /* Show tables known to the CAS Server */
   /* Table name:  MYCAS.FOO              */
   Proc Cas;
      session mysession;
   tableinfo result=r;
   print r;run;

   libname mycas2 cas tag=tagForMycas2;
   data mycas2.foo2;
      set sashelp.cars;
   run;

   /* Show tables using names known to the CAS server */
   /* Table name:  MYCAS.FOO                          */
   /* Table name:  TAGFORMYCAS2.FOO2                  */
   Proc Cas;
      session mysession;
   tableinfo result=r / table="mycas.foo"; run; ;
   print r; run;
   tableinfo result=r / table="tagForMycas2.foo2"; run; ;
   print r; run;
```

CASLIB= LIBNAME Option

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

Valid in: CAS LIBNAME statement

Category: Data Access

Default: the active caslib

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

To override the active caslib or the CASLIB= LIBNAME option, use the CASLIB= data set option.

See: CASLIB= statement, CASLIB= data set option, CASLIB= system option

Syntax

CASLIB=*caslib*

COMPRESS= LIBNAME Option

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

Valid in: CAS LIBNAME statement

Category: Data Access

Default: NO

Restrictions: The CAS LIBNAME engine does not support the COMPRESS= system option.

This option is ignored when tables are opened for input.

See: COMPRESS= data set option

Syntax

COMPRESS=<YES | NO>

DATALIMIT= LIBNAME Option

Specifies the maximum total amount of bytes that can be read from a file.

Valid in: CAS LIBNAME statement

Category: Data Access

Default: 100MB

Restriction: This option affects only read access.

See: DATALIMIT= data set option, READTRANSFERSIZE= LIBNAME option, READTRANSFERSIZE= data set option, WRITETRANSFERSIZE= LIBNAME option, WRITETRANSFERSIZE= data set option

Syntax

DATALIMIT=<(*integer*) | K | M | G | "ALL">

Optional Arguments

integer
specifies the total number of bytes to read.

K
specifies the total number of kilobytes to read.

M
specifies the total number of megabytes to read.

G
specifies the total number of gigabytes to read.

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

READTRANSFERSIZE= LIBNAME Option

Specifies the maximum data transfer size in bytes that can be used when reading a table from SAS Cloud Analytic Services.

Valid in: CAS LIBNAME statement

Category: Data Access

Alias: RTS

Default: 500MB

Restriction: This option affects only read access.

Interaction: READTRANSFERSIZE= is the maximum amount of data that is transferred with each read request that is made to SAS Cloud Analytic Services. 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 READTRANSFER= option.

See: DATALIMIT= LIBNAME option, DATALIMIT= data set option, READTRANSFERSIZE= data set option, WRITETRANSFERSIZE= LIBNAME option, WRITETRANSFERSIZE= data set option

Examples: 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 but larger than READTRANSFERSIZE=, this error results:

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

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

Syntax

READTRANSFERSIZE=<(*integer*) | K | M | G>

Optional Arguments

integer
specifies the total number of bytes to read.

K
specifies the total number of kilobytes to read.

M
specifies the total number of megabytes to read.

G
specifies the total number of gigabytes to read.

WRITETRANSFERSIZE= LIBNAME Option

Specifies the maximum data transfer size in bytes that can be used when writing to a table in SAS Cloud Analytic Services.

Valid in: CAS LIBNAME statement

Category: Data Access

Alias: WTS

Default: 512KB

Restriction: This option affects only write access.

See: DATALIMIT= LIBNAME option, DATALIMIT= data set option, READTRANSFERSIZE= LIBNAME option, READTRANSFERSIZE= data set option, WRITETRANSFERSIZE= data set option

Syntax

WRITETRANSFERSIZE=<(*integer*) | K | M | G>

Optional Arguments

integer
specifies the total number of bytes to write.

K
specifies the total number of kilobytes to write.

M
specifies the total number of megabytes to write.

G
specifies the total number of gigabytes to write.

APPEND= Data Set Option

Specifies whether to append rows to the SAS Cloud Analytic Services table from the DATA step.

Valid in: DATA and PROC steps

Category: Data Access

Default: NO

Syntax

APPEND=<YES | NO>

CASLIB= Data Set Option

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

Valid in: DATA and PROC steps

Category: Data Access

Default: the active caslib

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

To override the active caslib or the CASLIB= LIBNAME option, use the CASLIB= data set option.

See: CASLIB= statement, CASLIB= LIBNAME option, CASLIB= system option

Syntax

CASLIB=*caslib*

COMPRESS= Data Set Option

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

Valid in: DATA and PROC steps

Category: Data Access

Default: NO

Restrictions: The CAS LIBNAME engine does not support the COMPRESS= system option.

This option is ignored when tables are opened for input.

See: COMPRESS= LIBNAME option

Syntax

COMPRESS=<YES | NO>

COPIES= Data Set Option

Specifies the number of replicate copies for a replicated table.

Valid in: DATA and PROC steps

Category: Data Access

Default: NO

Syntax

COPIES=*number*

DATALIMIT= Data Set Option

Specifies the maximum total amount of bytes that can be read from a file.

Valid in: DATA step

Category: Data Access

Default: 100MB

Restriction: This option affects only read access.

See: DATALIMIT= LIBNAME option, READTRANSFERSIZE= LIBNAME option, READTRANSFERSIZE= data set option, WRITETRANSFERSIZE= LIBNAME option, WRITETRANSFERSIZE= data set option

Syntax

DATALIMIT=<(*integer*) | K | M | G | "ALL">

Optional Arguments

integer
specifies the total number of bytes to read.

K
specifies the total number of kilobytes to read.

M
specifies the total number of megabytes to read.

G
specifies the total number of gigabytes to read.

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

DUPLICATE= Data Set Option

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

Valid in: DATA and PROC steps

Category: Data Access

Default: NO

Interaction: The value for COPIES= is ignored (no duplicated data is replicated) if you use COPIES= with DUPLICATE=.

Note: With a duplicated table, all nodes have all rows of the table, and these rows are active everywhere.

Syntax

DUPLICATE=<YES | NO>

ONDEMAND= Data Set Option

Specifies how to evaluate temporary computed columns.

Valid in: DATA and PROC steps

Category: Data Access

Default: NO

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

Syntax

ONDEMAND=<YES | NO>

Syntax Description

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.

ORDERBY= Data Set Option

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

Valid in: DATA and PROC steps

Category: Data Access

Default: ascending order

Requirement: To use this option, you must first specify the PARTITION= option.

See: PARTITION= data set option

Example: This example shows how to partition and order a data set into SAS Cloud Analytic Services using ascending ordering of the numeric variable Z and descending ordering of the CH character variable to order the data within partitions that are formed based on the variable KEY.

```
%let nn=10000;
data myCas.testPart(partition=(key)
     orderby  =(z descending ch));
  retain key;
  do i = 1 to %nn
     n = int(ranuni(1)*50)+1;
     do j = 1 to n;
        if (j=1) then do;
           key = int(ranuni(1)*%nn)+1;
```

```
        end; else if (j > n/2) then do;
           key = int(ranuni(1)*(%nn/100))+1;
        end;
        x = 1;
        y = round(rannor(1),0.25);
        z = x + y;
        ch = put(abs(j),best4.);
        output;
     end;
  end;
run;
```

Syntax

ORDERBY=<ascending | descending><*variable-list*>

Syntax Description

variable-list
specifies a list of variables for ordering data within a partition. To specify descending order, include the DESCENDING keyword before the variable name in *variable-list*.

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.

PARTITION= Data Set Option

Specifies the list of partitioning variables for the output table.

Valid in: DATA and PROC steps

Category: Data Access

Default: none

Requirement: You must first specify this option before you can use the ORDERBY= option.

See: ORDERBY= data set option

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 or that is in the ORDERBY= option.
- 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.
- Partitioning is also available for tables in SMP CAS servers.

PROMOTE= Data Set Option

Requests that the table to be created in SAS Cloud Analytic Services is added with global scope.

Valid in: DATA and PROC steps

Category: Data Access

Default: NO

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>

READTRANSFERSIZE= Data Set Option

Specifies the maximum data transfer size in bytes that can be used when reading a table from SAS Cloud Analytic Services.

Valid in: DATA and PROC steps

Category: Data Access

Alias: RTS

Default: 500MB

Restriction: This option affects only read access.

Interaction: READTRANSFERSIZE= is the maximum amount of data that is transferred with each read request that is made to SAS Cloud Analytic Services. 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 READTRANSFER= option.

See: DATALIMIT= LIBNAME option, DATALIMIT= data set option, READTRANSFERSIZE= LIBNAME option, WRITETRANSFERSIZE= LIBNAME option, WRITETRANSFERSIZE= data set option

Examples: 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 but larger than READTRANSFERSIZE=, this error results:

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

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

Syntax

READTRANSFERSIZE=<(*integer*) | K | M | G>

Optional Arguments

integer
specifies the total number of bytes to read.

K
specifies the total number of kilobytes to read.

M
specifies the total number of megabytes to read.

G
specifies the total number of gigabytes to read.

SCRIPT= Data Set Option

Specifies the file reference for the SAS script that defines the temporary computed columns.

Valid in: DATA and PROC steps

Category: Data Access

Alias: TEMPEXPRESS=

Default: none

Syntax

SCRIPT=*fileref*

TAG= Data Set Option

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

Valid in: DATA and PROC steps

Category: Data Access

Default: (null)

Tip: Use **`TAG="MYTAG"`** if you want the table name to be constructed with a tag.

Example: This example appends the rows in a SAS table to the SAS Cloud Analytic Services table, user.sassek.cas.test.sales_fact.

```
libname mycas cas host="rdcgrd001" port=19999;
data mycas.sales_fact(append=yes tag="user.sassek.cas.test");
```

```
   set local_sales;
run;
```

SAS syntax continues to use a two-level name (mycas.sales_fact). In communicating with SAS Cloud Analytic Services, the uppercased tagName replaces mycas.

Syntax

TAG=*tagName*

Details

A SAS table is identified by its libref and member name—for example, work.foo. Table names in SAS Cloud Analytic Services can have more than two levels. For example, you must be able to map the libref.member syntax to be able to work with the five-level name for the SAS Cloud Analytic Services user.sassek.cas.test.sales_fact table. The CAS LIBNAME engine constructs the table name in SAS Cloud Analytic Services as upper(tagName).member, so the tagName replaces the libref.

TEMPNAMES= Data Set Option

Lists the names of the temporary computed columns that are added to the input table.

Valid in: DATA and PROC steps

Category: Data Access

Length: Temporary columns can be character or numeric (8-byte doubles).

Default: none

Restriction: Temporary computed columns are supported only for tables that are opened for input.

Example: This 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 mycars.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 $ sign and the variable length in bytes. For example, t2 in the subsequent code is a character variable of length 10, while t1 and t3 are numeric variables.

```
data _null_;
   file "./mypgm.sas";
   put "t1 = round(mpg_highway/mpg_city,0.5);";
   put "t2 = round(cylinders/enginesize,0.2);";
   put "t3 = msrp / invoice;";
;

filename newcols "./mypgm.sas";

proc ccor data=mycas.cars(tempnames=(t1 t2 t3)
                          tempexpress=newcols);
```

```
   var mpg_city mpg_highway t1 t3 weight;
   by t2;
   where type ne 'Hybrid';
run;

/* Add a temporary character column to the input table. */
filename newcols "./myOtherpgm.sas";

proc ccor data=mycas.cars(tempnames=(t1 t2 $ 10 t3)
   tempexpress=newcols);
```

Syntax

TEMPNAMES=

WRITETRANSFERSIZE= Data Set Option

Specifies the maximum data transfer size in bytes that can be used when writing to a table in SAS Cloud Analytic Services.

Valid in: DATA and PROC steps

Category: Data Access

Alias: WTS

Default: 512KB

Restriction: This option affects only write access.

See: DATALIMIT= LIBNAME option, DATALIMIT= data set option, READTRANSFERSIZE= LIBNAME option, READTRANSFERSIZE= data set option, WRITETRANSFERSIZE= LIBNAME option,

Syntax

WRITETRANSFERSIZE=<(*integer*) | K | M | G>

Optional Arguments

integer
specifies the total number of bytes to write.

K
specifies the total number of kilobytes to write.

M
specifies the total number of megabytes to write.

G
specifies the total number of gigabytes to write.

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" in *SAS Viya Administration: SAS Cloud Analytic Services*.

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*> DATASOURCE=(SRCTYPE="*type*" <*data-source-options*>) <PATH=*directory-path*><*option(s)*>;

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

Form 3: **CASLIB** _ALL_ ASSIGN | _ALL_ LIST | _ALL_ DROP

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

Interaction You can use name literals (n-literals) in caslib names. However, the CAS LIBNAME engine statement does not support caslib names with name literals in them.

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 or DROP options.

For naming rules see Names in the SAS Language.

DATASOURCE= (SRCTYPE="*type*", <*data-source-options*> , <ENCRYPTIONPASSWORD="*string*">)
specifies data source options to use when connecting to a data source. The SRCTYPE="*type*" option specifies the data source type. Data sources can be either databases or path-based. The *data-source-options* syntax depends on the data source.

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

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

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 115.

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

SRCTYPE= Type	Option Syntax	Example	Supported File Types
PATH	DATASOURCE=(SRCTYPE =PATH, <ENCRYPTIONPASSWOR D="*password*>) PATH="*file-path*";	`caslib mycsvs path="/data/Myxlsxfiles/" datasource=(srctype="path", encryptpassword=password);`	SASHDAT SAS7BDAT CSV XLS, XLSX
DNFS	DATASOURCE=(SRCTYPE =DNFS, <ENCRYPTIONPASSWOR D="*password*>) PATH="*file-path*";	`caslib mycsvs datasource=(srctype="dnfs") path="/data/Mycsvfiles/";`	SASHDAT CSV
HDFS	DATASOURCE=(SRCTYPE =HDFS, <ENCRYPTIONPASSWOR D="*password*">) PATH="*file-path*";	`caslib mycsvs path="/data/Mycsvfiles/" datasource=(srctype="hdfs", encryptpassword=password);`	SASHDAT CSV

See For information about the PATH= option, see "PATH="*directory-path*"" on page 42.

Examples "Example 1: Add a Global Caslib" on page 44

"Example 6: Encrypt Tables in a Caslib" on page 53

Optional Arguments

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

Restriction	The CASLIB _ALL_ ASSIGN statement only creates a libref for caslibs that following SAS libref naming rules. For example, if the caslib has more than eight characters, then no libref is created for that caslib.
Requirement	The ASSIGN option must be used with the _ALL_ option.
Examples	CASLIB _ALL_ ASSIGN creates SAS librefs for all existing caslibs.
	CASLIB _ALL_ ASSIGN SESSREF=MySess creates SAS librefs for all existing caslibs in the session named MySess.

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

Example "Program 2: List All Caslibs" on page 47

CREATEDIRECTORY=TRUE | FALSE
when set to TRUE, creates a caslib directory.

Alias	CREATEDIR
Default	FALSE
Restriction	Directory creation is available only for global caslibs.

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

Alias DESC=

DROP
removes a caslib. The caslib is removed from all sessions that had access to it. If the removed caslib is the active caslib for a session, then the active caslib returns to the initially active caslib.

Alias	CLEAR
Tip	Specify *caslib-reference-name* to remove a single caslib. Specify _ALL_ to drop all caslibs.
Example	"Example 2: Drop a Caslib" on page 45

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. See "Adjust Caslib Management Privileges" in *SAS Viya Administration: SAS Cloud Analytic Services*.

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.

Examples "Example 2: Drop a Caslib" on page 45

"Example 3: List Caslib Settings" on page 46

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 43

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 57.

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.
- drop caslibs.
- 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 user starts. 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. For caslib authorization information, see "Adjust Caslib Management Privileges" in *SAS Viya Administration: SAS Cloud Analytic Services*.

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*.

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 180.

Examples

Example 1: Add a Global Caslib

Program

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

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

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

caslib Hadooplib desc="Hadoop Caslib"
               datasource=(srctype="hadoop",                                      /*2*/
               dataTransferMode="parallel",
               hadoopjarpath="Hadoo-jar-file-path",
               hadoopconfigdir="Hadoop-config-files-path",
               username="user-id",
               server="Hadoop-server-hostname",
               schema="schema-name") global;
```

1 The first CASLIB statement adds a global-scope caslib named Myvapublic. 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. Myvapublic is now the active caslib.

2 The second CASLIB statement adds a global-scope caslib named Hadooplib, which provides access to a Hadoop database. The DATASOURCE= option specifies the option to use when connecting to the database. The GLOBAL option enables you to promote tables to the caslib. Hadooplib is now the active caslib.

SAS Log

The notes in the SAS log verify that the caslibs Vapublic and Hadooplib were added. Note that Hadooplib is the active caslib, because it was added last.

```
 56         caslib vapublicCL path="/vapublic" datasource=(srctype="hdfs")
global;
 NOTE: 'VAPUBLIC' is now the active caslib.
 NOTE: Cloud Analytic Services added the caslib 'VAPUBLICCL'.

 NOTE: Action to ADD caslib VAPUBLICCL completed for session STUDIO.
                caslib Hadooplib desc="Hadoop Caslib"
                       datasource=(srctype="hadoop", /*2*/
                       dataTransferMode="parallel",
                       hadoopjarpath="Hadoo-jar-file-path",
                       hadoopconfigdir="Hadoop-config-files-path",
                       username="user-id",
                       server="Hadoop-server-hostname",
                       schema="schema-name") global;
 NOTE: 'Hadooplib' is now the active caslib.
 NOTE: Cloud Analytic Services added the caslib 'Hadooplib'.
 NOTE: Action to ADD caslib Hadooplib completed for session CASAUTO.
```

Additional Information

- For information about specifying Hadoop data source options, see "SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop" on page 120.
- You can add global caslibs only if you are authorized to do so, and the caslib must also be set to global in the CAS Server Monitor. See "Adjust Caslib Management Privileges" in *SAS Viya Administration: SAS Cloud Analytic Services*.

Example 2: Drop a Caslib

Program: Drop a Caslib

The following CASLIB statement drops the caslib Hadooplib. You can use a second CASLIB LIST statement to verify that the Hadooplib caslib has been removed.

```
caslib Hadooplib drop;
caslib _all_ list;
```

SAS Log

The notes in the SAS log verify that the caslib Hadooplib was dropped.

```
NOTE: Cloud Analytic Services removed the caslib 'HadoopLIB'.

NOTE: Session = STUDIO Name = CASUSER(casdemo)
         Type = PATH
         Description = Personal File System Caslib
         Path = /u/casdemo/
         Definition =
         Subdirs = Yes
         Local = No
         Active = Yes
         Personal = Yes
NOTE: Session = STUDIO Name = CASUSERHDFS(casdemo)
         Type = HDFS
         Description = Personal HDFS Caslib
         Path = /user/casdemo/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
         Personal = Yes
NOTE: Session = STUDIO Name = MYVAPUBLIC
         Type = HDFS
         Description =
         Path = /vapublic/
         Definition =
         Subdirs = No
         Local = No
         Active = No
         Personal = No
NOTE: Session = STUDIO Name = Formats
         Type = PATH
         Description = Format Caslib
         Path = /casdemo/formats/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
         Personal = No
NOTE: Session = STUDIO Name = HPS
         Type = HDFS
         Description = HDAT files on /hps
         Path = /hps/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
         Personal = No
```

Additional Information

- If you drop the active caslib from a session, the first active caslib becomes active again.
- You can also use the CASUTIL procedure to drop caslibs and view information about caslibs. For documentation about the CASUTIL procedure, see Chapter 4, “CASUTIL Procedure,” on page 57.

Example 3: List Caslib Settings

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

Program 1: List the Settings for a Specific Caslib

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

```
caslib CASUSER list;
caslib HPS list;
```

SAS Log

```
NOTE: Session = STUDIO Name = CASUSER(casdemo)
         Type = PATH
         Description = Personal File System Caslib
         Path = /u/casdemo/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
         Personal = Yes

NOTE: Session = STUDIO Name = HPS
         Type = HDFS
         Description = HDAT files on /hps
         Path = /hps/
         Definition =
         Subdirs = Yes
         Local = No
         Active = Yes
         Personal = No
```

Program 2: List All Caslibs

The CASLIB statement with the LIST option and the _ALL_ option displays all of the caslibs that are available and the caslib settings for each one.

```
caslib _all_ list;
```

SAS Log

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. Caslibs 5 and 6 were created by another session, but have global scope, so they are available to all sessions.

```
NOTE: Session = STUDIO Name = CASUSER(casdemo)                    1
         Type = PATH
         Description = Personal File System Caslib
         Path = /u/casdemo/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
         Personal = Yes
NOTE: Session = STUDIO Name = CASUSERHDFS(casdemo)                2
         Type = HDFS
         Description = Personal HDFS Caslib
         Path = /user/casdemo/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
         Personal = Yes
NOTE: Session = STUDIO Name = HADOOPLIB                           3
         Type = hadoop
         Description = 'Hadoop Caslib'
         Path =
         Definition = ui
         Subdirs = No
         Local = No
         Active = No
         Personal = No
NOTE: Session = STUDIO Name = MYVAPUBLIC                          4
         Type = HDFS
         Description =
         Path = /vapublic/
         Definition =
         Subdirs = No
         Local = No
         Active = No
         Personal = No
NOTE: Session = STUDIO Name = Formats                             5
         Type = PATH
         Description = Format Caslib
         Path = /casdemo/formats/
         Definition =
         Subdirs = Yes
         Local = No
         Active = No
         Personal = No
NOTE: Session = STUDIO Name = HPS                                 6
         Type = HDFS
         Description = HDAT files on /hps
         Path = /hps/
         Definition =
         Subdirs = Yes
         Local = No
         Active = Yes
         Personal = No
```

Additional Information

When you list caslibs, the following caslib setting information is displayed:

Type=string	indicates the caslib type specified by the SRCTYPE= option.
Description=string	displays the description specified by the DESCRIPTION= option.
Path=string	displays the path specified in the PATH= option.

Definition=string	displays the data source options specified by the DATASOURCE= option.
Subdirs=Yes | No	indicates whether the caslib can access subdirectories. Subdirs= is specified by the SUBDIRS option.
Local=Yes | No	indicates whether the caslib is session scope. **Local=yes** indicates session scope. Local= corresponds to the SESSION option in the CASLIB statement.
Active=Yes | No	indicates whether the caslib is the active caslib.
Personal=Yes | No	indicates whether the caslib is a personal caslib or not.

- 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 57.
- For the Hadoop connect string syntax, see "SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop" on page 120.

Example 4: Load and Save a Table

Program

The following program adds the caslib Myvapublic, which provides a place to copy an in-memory version of the Cars data set. The in-memory table, named carsWght, is then saved to the HDFS data source.

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

libname mycas cas;                                                        /*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 Vapublic directory. Myvapublic is now the active caslib.

2 The LIBNAME statement creates 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.

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 Vapublic directory unless you save it.

4 The CASUTIL procedure saves the table to Vapublic. The INCASLIB= option specifies the caslib that contains the file, and the OUTCASLIB= option specifies the

caslib that the file is being made available to. Use the PROC CASUTIL LIST statement to make sure that the table exists in the Myvapublic caslib.

SAS Log

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

```
60         save casdata="CarsWght";
NOTE: Cloud Analytic Services saved the file CarsWght.sashdat to HDFS in caslib
MYVAPUBLIC.
```

Results

The following results of the PROC CASUTIL LIST TABLES statement show the tables that are in the Myvapublic caslib.

The CASUTIL Procedure

Caslib Information	
Library	MYVAPUBLIC
Source Type	HDFS
Path	/vapublic/
Session local	Yes
Active	Yes
Personal	No

The CASUTIL Procedure

Table Information for Caslib MYVAPUBLIC									
Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
JCARS	7	5	utf-8	08Apr2016:12:53:08	08Apr2016:12:53:08	No	No	No	No

Additional Information

- Caslibs that use SRCTYPE="HDFS" are for distributed servers only. They use a Hadoop instance that is co-located with SAS Cloud Analytic Services.
- Caslibs provide a way to organize in-memory tables and an associated data source. They also provide a way to apply access controls to data. Once the caslib is dropped, the in-memory tables are dropped, too. Files in the caslib's data source are not removed or modified. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- Tables that are saved from a caslib are saved in SASHDAT format by default.
- For information about using the DATA step in CAS, see "DATA Step Programming" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.
- For documentation about the CASUTIL procedure syntax, see Chapter 4, "CASUTIL Procedure," on page 57.
- 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: Copy Data from One Data Source to Another

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

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

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

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

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

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

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

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

Results

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

Output 3.1 *Table Metadata*

Summary of Presentation and Taste Scores

The CASUTIL Procedure

CAS File Information					
Name	**Permission**	**Owner**	**Group**	**File Size**	**Last Modified**
donations.csv	-rwxr-xr-x			809.8MB	14May2014:12:32:46

Column Information for donations.csv in Caslib LDBETA			
Column	**Type**	**Length**	**Formatted Length**
donationid	varchar	0	0
projectid	varchar	0	0
donor_acctid	varchar	0	0
donor_city	varchar	0	0
donor_state	varchar	0	0
donor_zip	double	8	12
is_teacher_acct	varchar	0	0
donation_timestamp	varchar	0	0
donation_to_project	double	8	12
donation_optional_support	double	8	12
donation_total	double	8	12
dollar_amount	varchar	0	0
donation_included_optional_support	varchar	0	0
payment_method	varchar	0	0
payment_included_acct_credit	varchar	0	0
payment_included_campaign_gift_card	varchar	0	0
payment_included_web_purchased_gift_card	varchar	0	0
payment_was_promo_matched	varchar	0	0
via_giving_page	varchar	0	0
for_honoree	varchar	0	0
donation_message	varchar	0	0

The following results show the files available in the HPS caslib, displayed by the PROC CASUTIL LIST statement.

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

Table Information for Caslib HPS

Label	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table
	279979	64	utf-8	18Mar2016:14:14:13	18Mar2016:14:14:13	Yes
	2018	24	utf-8	18Mar2016:14:14:26	18Mar2016:14:14:26	Yes
All Data Types	20	8	latin1	18Mar2016:14:14:39	18Mar2016:14:14:39	Yes
QUERY_FOR_actual_plan_2010_2011.sas7bdat	55320	25	latin1	18Mar2016:14:14:51	18Mar2016:14:14:51	Yes
Amazon Reviews	1830	12	latin1	18Mar2016:14:15:04	18Mar2016:14:15:04	Yes
Asia cities map coordinates	193723	26	latin1	18Mar2016:14:15:17	18Mar2016:14:15:17	Yes
Big organics	111115	13	latin1	18Mar2016:14:15:54	18Mar2016:14:15:54	Yes
	399999	8	utf-8	18Mar2016:14:16:08	18Mar2016:14:16:08	Yes
Bodyfat2	252	19	latin1	18Mar2016:14:16:21	18Mar2016:14:16:21	Yes
Campaign for Investments	1660	33	latin1	18Mar2016:14:16:34	18Mar2016:14:16:34	Yes
Car Sales	20	9	latin1	18Mar2016:14:16:47	18Mar2016:14:16:47	Yes
Car Sales data from sashelp library	428	15	latin1	18Mar2016:14:17:01	18Mar2016:14:17:01	Yes
Census	129605	172	latin1	18Mar2016:14:17:14	18Mar2016:14:17:14	Yes
	3476	4	utf-8	18Mar2016:14:17:26	18Mar2016:14:17:26	Yes
Claim History	10302	36	latin1	18Mar2016:14:17:39	18Mar2016:14:17:39	Yes

Additional Information

- Caslibs provide a way to organize in-memory tables and an associated data source. They also provide a way to apply access controls to data. After a caslib is dropped, the in-memory tables are dropped, too. Files in the caslib's data source are not removed or modified in any way. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- Tables that are saved from a caslib are saved in SASHDAT format by default.
- For information about using the DATA step in CAS, see "DATA Step Programming" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.
- For documentation about the CASUTIL procedure syntax, see Chapter 4, "CASUTIL Procedure," on page 57.
- 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 6: Encrypt Tables in a Caslib

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

libname mycas cas;

proc casutil;
   load data=sashelp.cars groupby=("make") outcaslib="Encr";
   contents casdata="cars";
   list files incaslib="encr";

proc mdsummary data=mycas.cars;
   var msrp invoice;
   output out=mycas.mdsumstatEncr;
   groupby make;
run;
```

```
options obs=15;
proc print data=mycas.mdsumstatEncr;
   var Make _Column_ _NObs_ _Mean_ _Max_ _Min_ _Std_;
   title "Summary of MSRP and Invoice, Grouped by Make";
run;

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

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

2 Create 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 if 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*

The CASUTIL Procedure

Caslib Information	
Library	ENCR
Source Type	PATH
Path	[redacted]
EncryptionPassword	*****
Session local	Yes
Active	Yes
Personal	No

Results: CONTENTS Statement

Output 3.4 *Column Information for the Cars Data Set*

Column Information for CARS in Caslib ENCR					
Column	**Label**	**Type**	**Length**	**Formatted Length**	**Format**
Make		char	13	13	
Model		char	40	40	
Type		char	8	8	
Origin		char	6	6	
DriveTrain		char	5	5	
MSRP		double	8	8	DOLLAR
Invoice		double	8	8	DOLLAR
EngineSize	Engine Size (L)	double	8	12	
Cylinders		double	8	12	
Horsepower		double	8	12	
MPG_City	MPG (City)	double	8	12	
MPG_Highway	MPG (Highway)	double	8	12	
Weight	Weight (LBS)	double	8	12	
Wheelbase	Wheelbase (IN)	double	8	12	
Length	Length (IN)	double	8	12	

Results: LIST Statement Information for the Data Source

Output 3.5 *List of Files in the Data Source*

The CASUTIL Procedure

CAS File Information						
Name	**Permission**	**Owner**	**Group**	**Encryption Method**	**File Size**	**Last Modified**
mdsumstatEncr.sashdat	-rwxr-xr-x	[redacted]	[redacted]	AES	19.6KB	06Apr2016:16:50:26

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 "DATA Step Programming" in *SAS Cloud Analytic Services: Accessing and Manipulating Data*.
- For documentation about the CASUTIL procedure syntax, see Chapter 4, "CASUTIL Procedure," on page 57.
- 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 83.

Chapter 4

CASUTIL Procedure

Overview: CASUTIL Procedure

What Does the CASUTIL Procedure Do?

The CASUTIL procedure works with tables in SAS Cloud Analytic Services, SAS data sets in SAS libraries, and external files. The procedure has three functional areas:

- data transfer
- table and file information
- drops tables and deletes files

In the area of data transfer, you can perform the following operations:

- load a data set from a SAS library into a memory on SAS Cloud Analytic Services.
- save in-memory tables in a caslib to the data source that is associated with the caslib.
- load files from the data source that is associated with a caslib into memory on SAS Cloud Analytic Services.

For file and table information, you can perform the following operations:

- view column names, data types, and other column information.
- list the in-memory tables in a caslib.
- list the files in a caslib's data source.

In the area of table and file management, the procedure enables you to drop in-memory tables. Dropping a table frees resources in the server but leaves the file in the data source that is associated with the caslib untouched. The procedure also enables you to delete files from the data source that is associated with the caslib.

The CASUTIL procedure executes without using the RUN statement. After you submit the PROC CASUTIL statement, you can submit additional procedure statements without submitting the PROC statement again. Use the QUIT statement to terminate the procedure.

Terminology

The following terms are used throughout the CASUTIL procedure documentation:

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

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

Syntax: CASUTIL Procedure

PROC CASUTIL <*option(s)*>;
CONTENTS CASDATA="*table-name*" <INCASLIB="*caslib*"> <*option(s)*>;
DELETESOURCE CASDATA="*file-name*" <INCASLIB="*caslib*"> <QUIET>;
DROPTABLE CASDATA="*table-name*" <INCASLIB="*caslib*"> <QUIET>;
LIST <FILES | TABLES> <*options(s)*>;
LOAD CASDATA="*file-name*" | DATA=*SAS-data-set* | FILE="*SAS-file*" <*option(s)*>;
PROMOTE CASDATA="*table-name*" <INCASLIB="*caslib*"> <CASOUT="*table-name*"> <OUTCASLIB="*caslib*"> <DROP | KEEP>;
SAVE CASDATA="*table-name* " <INCASLIB="*caslib*"> <CASOUT="*file-name*" <OUTCASLIB="*caslib*">> <*option(s)*>;
QUIT;

PROC CASUTIL Statement

Manages tables and files in SAS Cloud Analytic Services.

See: For examples of the CASUTIL procedure in real-life scenarios, see *SAS Cloud Analytic Services: 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 78
"Example 4: Saving and Loading Encrypted SASHDAT Files" on page 79

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 64.

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 101 and Chapter 8, "Data Connectors," on page 115.

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 65.

Alias VARLIST=

Example: Viewing Column Names from a CSV File

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

DELETESOURCE Statement

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

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

Syntax

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

Required Argument

CASDATA="*file-name*"
specifies the name of the file to remove.

Optional Arguments

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

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

DROPTABLE Statement

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

Example: “Example 4: Saving and Loading Encrypted SASHDAT Files” on page 79

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 74

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 101 and Chapter 8, "Data Connectors," on page 115.

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.

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

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

Alias REPLICATION=

Default 1

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

DATASOURCEOPTIONS=(*data-source-options*)
specifies overrides for the DATASOURCE= options for the input caslib. For more information, see Chapter 6, “Platform Data Sources,” on page 101 and Chapter 8, “Data Connectors,” on page 115.

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 109 and Chapter 8, "Data Connectors," on page 115.

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.

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*

LOAD Statement Option	LOAD Statement Form
APPEND	LOAD DATA=
COMPRESS	LOAD DATA=
COPIES=	All LOAD statement forms. For more information, see "Data Redundancy" on page 101.
DATASOURCEOPTIONS=	LOAD CASDATA= and LOAD DATA=
GROUPBY=	LOAD CASDATA= and LOAD DATA=
IMPORTOPTIONS=	LOAD CASDATA= and LOAD FILE=
INCASLIB=	LOAD CASDATA=
LABEL=	All LOAD statement forms
ORDERBY=	LOAD CASDATA= and LOAD DATA= *Note:* ORDERBY= requires GROUPBY=
OUTCASLIB=	All LOAD statement forms
PROMOTE	All LOAD statement forms
REPEAT	LOAD DATA=
REPLACE	All LOAD statement forms *Note:* Keep in mind that global-scope tables cannot be replaced. Use the DROPTABLE statement before the LOAD statement.

LOAD Statement Option	LOAD Statement Form
VARS=	LOAD CASDATA=
WHERE=	LOAD CASDATA= *Note:* You can specify WHERE= as a data set option when you use the LOAD DATA= form.

PROMOTE Statement

The PROMOTE statement copies a session-scope table to global scope.

Note: The PROMOTE statement does not include a REPLACE option. The server does not support promoting a session-scope table and replacing a global-scope table in one operation. You must drop the global-scope table first.

Example: "Example 3: Promote a Table" on page 78

Syntax

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

Required Argument

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

Optional Arguments

CASOUT="*table-name*"
specifies the name to use for the promoted table.

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

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

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

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

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

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

SAVE Statement

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

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 79

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 64 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 101 and Chapter 8, “Data Connectors,” on page 115.

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 66 for the LOAD statement.

Enclose Values in Quotation Marks

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

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

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

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

Subdirectories and Filename Matching

Exclude Subdirectories from the File Information Results

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

```
cas casauto setsessopt=(caslib='casuser');

proc casutil;
   list files nosubdirs;
quit;
```

List the Files in a Subdirectory

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

To list the files in **/home/sasdemo/mydata**, you can use the SUBDIR= option in the LIST FILES statement.

```
cas casauto setsessopt=(caslib='casuser');

proc casutil;
   list files subdir='mydata';
quit;
```

The CASUTIL Procedure

CAS File Information

Name	Permission	Owner	Group	File Size	Last Modified
iris.csv	-rwxr-xr-x	[redacted]	[redacted]	3.1KB	08Aug2016:19:38:59
titanic3.xls	-rwxr-xr-x	[redacted]	[redacted]	277.5KB	09Aug2016:09:01:48
jta_free_wifi.csv	-rwxr-xr-x	[redacted]	[redacted]	19.1MB	09Aug2016:09:03:39

Using Wildcard Characters for Filename Matching

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

```
cas casauto setsessopt=(caslib='casuser');

proc casutil;
   list files subdir='mydata/%.csv';
quit;
```

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

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

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

Loading Files from a Subdirectory

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

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

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

Saving Files to a Subdirectory

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

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

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

```
            casout="mydata/setosa.csv"
            replace;

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

The CASUTIL Procedure

CAS File Information

Name	Permission	Owner	Group	File Size	Last Modified
iris.csv	-rwxr-xr-x	[redacted]	[redacted]	3.1KB	08Aug2016:19:38:59
titanic3.xls	-rwxr-xr-x	[redacted]	[redacted]	277.5KB	09Aug2016:09:01:48
jta_free_wifi.csv	-rwxr-xr-x	[redacted]	[redacted]	19.1MB	09Aug2016:09:03:39
setosa.csv	-rwxr-xr-x	[redacted]	[redacted]	0.9KB	09Aug2016:10:00:23

Limitations and Restrictions

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:

- PATH
- HDFS
- DNFS

Results: CASUTIL Procedure

Procedure Output

The CONTENTS statement provides detailed information for an in-memory table. The following program generated the results with a distributed server with seven worker nodes.

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

The CASUTIL Procedure

Table Information for Caslib CASUSER( )

Table Name	Label	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
IRIS	Fisher's Iris Data (1936)	150	5	utf-8	15Apr2016:12:01:26	15Apr2016:12:01:26	No	No	No	No

Detail Information for iris in Caslib CASUSER( ).

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated
ALL	14	7	150	7200	0	0	0	14	16864	0	0

Column Information for IRIS in Caslib CASUSER( )

Column	Label	Type	Length	Formatted Length
Species	Iris Species	char	10	10
SepalLength	Sepal Length (mm)	double	8	12
SepalWidth	Sepal Width (mm)	double	8	12
PetalLength	Petal Length (mm)	double	8	12
PetalWidth	Petal Width (mm)	double	8	12

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*

ODS Table	Description	Statement Used
CaslibInfo	Caslib information	LIST
ColumnInfo	Column information	CONTENTS
FileInfo	CAS file information	LIST with the FILES option
TableDetails	Detailed table information	CONTENTS
TableInfo	CAS table information	LIST with the TABLES option, CONTENTS

Examples: CASUTIL Procedure

Example 1: Load a CSV File into CAS

Program

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

```
proc casutil;
   list files;

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

   list tables;
quit;
```

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

2 The IMPORTOPTIONS= option is used to specify the file type and options for reading the data into the server.

Results: LIST FILES Statement for the CSVFILES Caslib

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

The CASUTIL Procedure

File Information for root of caslib CSVFILES.

Name	Permission	Owner	Group	File Size	Last Modified
Spreadsheet10k.xlsx	-rwxrwxrwx	nfsnobody	nfsnobody	563.5KB	08Nov2012:07:21:12
mailorder.csv	-rwxrwxrwx	nfsnobody	nfsnobody	117.2KB	29Sep2015:11:40:14
NST_EST2012_ALLDATA.csv	-rwxrwxrwx	nfsnobody	nfsnobody	20.0KB	15Jan2013:08:59:28
sine.csv	-rwxrwxrwx	nfsnobody	nfsnobody	309.3KB	15Jan2013:13:18:14
SC-EST2011-6RACE-ALL.csv	-rwxrwxrwx	nfsnobody	nfsnobody	138.3KB	15Jan2013:09:02:53
sine10k.csv	-rwxrwxrwx	nfsnobody	nfsnobody	582.9KB	09May2013:07:16:16
UNdata_Export_20130115_092219649.csv	-rwxrwxrwx	nfsnobody	nfsnobody	595.4KB	15Jan2013:09:22:20
2012_nfl_pbp_data_reg_season1.csv	-rwxrwxrwx	nfsnobody	nfsnobody	6.5MB	06Feb2013:11:19:02
County_Population.csv	-rwxrwxrwx	nfsnobody	nfsnobody	1.3MB	14Jan2013:14:15:41

Results: LIST TABLES Statement for the CSVFILES Caslib

The following display shows the results of the LIST TABLES statement. It shows that the County_Population table is the only in-memory table in the caslib.

The CASUTIL Procedure

Caslib Information	
Library	CSVFILES
Source Type	DNFS
Description	"Spreadsheets and CSV source data."
Path	[redacted]
Session local	Yes
Active	Yes
Personal	No

The SAS System

The CASUTIL Procedure

Table Information for Caslib CSVFILES											
Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Source Caslib	Compressed
COUNTY_POPULATION	3193	97	utf-8	08Apr2016:10:41:18	08Apr2016:10:41:18	No	No	No	County_Population.csv	CSVFILES	No

Example 2: Append Rows to an In-Memory Table

Program

```
proc casutil;
   load data=sashelp.cars(where=(make="Buick"))                    /* 1 */
        casout="some_cars"
        label="Some makes from the Sashelp.Cars sample data."
        promote;

   /* add rows for a few more makes */
   load data=sashelp.cars(where=(make in ("Ford", "Chrysler")))
        casout="some_cars"
        append;                                                    /* 2 */

   list tables;
quit;

libname mycas cas;

proc cardinality data=mycas.some_cars outcard=mycas.cars_cardinality;
   vars enginesize mpg_highway mpg_city;                           /* 3 */
run;

proc casutil;
  contents casdata="cars_cardinality";                             /* 4 */
run;

proc print data=mycas.cars_cardinality;
  var _varname_ _cardinality_ _nobs_ _nmiss_ _min_--_kurtosis_;
run;
```

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

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

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

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

Results: LIST TABLES Statement for the CASUSER Caslib

The CASUTIL Procedure

Table Information for Caslib [redacted]

Table Name	Label	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
SOME_CARS	Some makes from the Sashelp.Cars sample data.	47	15	utf-8	08Apr2016:13:27:37	08Apr2016:13:27:37	Yes	No	No	No

Results: CONTENTS Statement for the Cars_Cardinality Table

The CASUTIL Procedure

Table Information for Caslib [redacted]

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
CARS_CARDINALITY	3	26	utf-8	08Apr2016:13:29:19	08Apr2016:13:29:19	No	Yes	No	No

Detail Information for cars_cardinality in Caslib [redacted].

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated
ALL	7	7	21	7560	0	0	0	7	8792	0	0

Column Information for CARS_CARDINALITY in Caslib [redacted]

Column	Label	Type	Length	Formatted Length	Format
VARNAME	Variable name	char	32	32	$
FMTWIDTH	Width of the variable formatted value	double	8	12	BEST
TYPE	Type of the raw values	char	1	1	$

Results: Select Columns from the Cars_Cardinality Table

Obs	_VARNAME_	_CARDINALITY_	_NOBS_	_NMISS_	_MIN_	_MAX_	_MEAN_	_STDDEV_	_SKEWNESS_	_KURTOSIS_
1	EngineSize	17	47	0	2	6.8	3.3957446809	1.0133795046	0.777871628	1.3478938265
2	MPG_Highway	15	47	0	13	36	26.638297872	4.3410354	-0.725567937	1.3662855198
3	MPG_City	12	47	0	10	27	19.382978723	3.4550101711	0.1929445364	0.6283786957

Example 3: Promote a Table

Program

```
caslib hps datasource=(srctype="path") path="/hps" global;

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

libname mycas cas;

proc casutil;
   load data=sashelp.iris casout="irisraw";
quit;

data mycas.irisout;                                    /* 1 */
   set mycas.irisraw;
   sepalratio = sepalwidth / sepallength;
   petalratio = petalwidth / petallength;
run;

/*
 * The purpose for outcaslib= is to show how to
 * work with more than one caslib.
 */
proc casutil outcaslib="hps";                          /* 2 */
   promote casdata="irisout";
quit;

proc casutil incaslib="hps";
   contents casdata="irisout";
quit;
```

1 The DATA step creates an output table that is named Irisout from an input table named Irisraw. Two columns are added to the table.

2 The OUTCASLIB= option is used to demonstrate how to work with more than one caslib. If you specify OUTCASLIB= when you promote or load a table, then you need to use INCASLIB= with the same name to access the table again.Notice that in the CASDATA= option in the PROMOTE statement on page 67 that follows, the table name is specified without the libref.

Results: CONTENTS Statement for the Irisout Table

The following graphic shows the results of the CONTENTS statement. The Table Information results show that the caslib is HPS and that the table is promoted to global

scope. The Column Information results show the two columns that were added with the DATA step.

The CASUTIL Procedure

Table Information for Caslib HPS

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
IRISOUT	150	7	utf-8	08Apr2016:13:47:40	08Apr2016:13:47:40	Yes	No	No	No

Detail Information for irisout in Caslib HPS.

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated
ALL	14	7	150	9600	0	0	0	14	19200	0	0

Column Information for IRISOUT in Caslib HPS

Column	Label	Type	Length	Formatted Length
Species	Iris Species	char	10	10
SepalLength	Sepal Length (mm)	double	8	12
SepalWidth	Sepal Width (mm)	double	8	12
PetalLength	Petal Length (mm)	double	8	12
PetalWidth	Petal Width (mm)	double	8	12
sepalratio		double	8	12
petalratio		double	8	12

Example 4: Saving and Loading Encrypted SASHDAT Files

Program

The CSV file that is used in this example was downloaded on 10FEB2016 from https://www.hokoukukan.go.jp/download/jta_free_wifi.csv. Your results for the count of WiFi access points by category could be different.

```
options validmemname=extend validvarname=any;                    /* 1 */

options caslib="casuser";
libname mycas cas;

proc casutil incaslib="casuser" outcaslib="casuser";
   load casdata="jta_free_wifi.csv"
        importoptions=(filetype="csv" getnames="yes"
           encoding="sjis")                                      /* 2 */
        casout="jta_free_wifi";

   contents casdata="jta_free_wifi";                             /* 3 */

   save casdata="jta_free_wifi"
        datasourceoptions=(encryptionPassword='changeit');

   droptable casdata="jta_free_wifi";                            /* 4 */
```

```
   load casdata="jta_free_wifi.sashdat"                          /* 5 */
        importoptions=(filetype="hdat" encryptionPassword='changeit');
quit;

proc mdsummary data=mycas.jta_free_wifi;                         /* 6  */
  var スポット ID;
  groupby カテゴリー / out=mycas.category;
run;

proc print data=mycas.category;                                  /* 7  */
  var カテゴリー _nobs_;
run;
```

1 This example uses data with column names that do not follow SAS naming conventions. These options provide greater flexibility with table names and column names.

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

3 The CONTENTS statement on page 60 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 81.

4 The DROPTABLE statement is not necessary in most programs. It is included in this example so that the subsequent LOAD CASDATA= statement succeeds without the REPLACE option.

5 The LOAD CASDATA= statement includes the encryption password. Notice also that you do not need to specify an ENCODING= option. The SAVE statement stored the data as UTF-8 when it created the SASHDAT file.

6 The MDSUMMARY procedure is included to show that after the table is loaded into memory, then you can use a CAS engine libref to access the in-memory table. The goal for this example is to identify the different categories of hotspots and the counts. Only one variable is summarized, the WiFi hotspot identifier, and the variable is grouped by values of the hotspot category. The summary is output to an in-memory table on the server that is named Category.

7 The PRINT procedure is used to read the summarized results from the in-memory table on the server. The VAR statement limits the display to the different hotspot categories and the count for each category. See Output 4.1 on page 81.

Results: CONTENTS Statement for the Jta_free_wifi Table

The CASUTIL Procedure

Table Information for Caslib CASUSER([redacted])

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Source Caslib	Compressed
JTA_FREE_WIFI	42259	21	utf-8	08Apr2016:13:56:01	08Apr2016:13:56:01	No	No	No	jta_free_wifi.csv	CASUSER([redacted])	No

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

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated
ALL	84	42	42259	32730247	19883511	0	0	84	65471024	0	0

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

Column	Type	Length	Formatted Length
スポットID	double	8	12
スポット名（日本語）	varchar	115	55
スポット名（英語）	varchar	255	255
スポットステータス	varchar	20	8

Results: WiFi Hotspot Categories and Counts

Output 4.1 *WiFi Hotspot Categories and Counts*

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

Chapter 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

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 with the OUT= option specified. You cannot specify both the OUTPUT= statement and a GROUPBY statement with the OUT= option specified.

Syntax

OUTPUT <OUT=*libref.table name*>;

Optional 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*

Column Name	Statistic	Description
CSS	CSS	Corrected sum of squares
_CV_	CV	Coefficient of variation
MAX	MAX	Maximum value
MEAN	MEAN	Arithmetic mean
MIN	MIN	Minimum value
NMISS	NMISS	Number of values that are missing
NOBS	NOBS	Total number of observations

Column Name	Statistic	Description
PRT	PRT	p-Value for Student's t statistics
STD	STD	Standard deviation
STDERR	STDERR	Standard error of the mean
SUM	SUM	Sum
_T_	T	Student's t statistic
USS	USS	Uncorrected sum of squares
VAR	VAR	Variance

Examples: MDSUMMARY Procedure

Example 1: Compute Descriptive Statistics

Program

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

```
caslib MyCasLib datasource=(srctype="path") path='your-file-path';/*1*/

libname mycas cas;/*2*/

proc casutil;/*3*/
   load data=sashelp.cars outcaslib="MyCasLib ";
   contents casdata="cars";
quit;

proc mdsummary data=mycas.cars;/*4*/
   var MSRP MPG_City;
   output out=mycas.mdsumstat;
run;

proc print data=mycas.mdsumstat;/*5*/
   var _Column_ _NObs_ _Mean_ _Max_ _Min_ _Std_  ;
   title 'Summary of MSRP and City Miles Per Gallon';
run;

proc casutil;/*6*/
   save casdata="mdsumstat" incaslib="MyCasLib" outcaslib="MyCasLib";
```

```
   list files incaslib="MyCasLib";
quit;
```

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

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

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

4 The MDSUMMARY procedure produces summary statistics for MPG_CITY and MSRP. The OUTPUT statement creates an in-memory table named Mycas.MdsumStat. The table includes a row of summary statistics for MSRP and MPG_City.

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

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

Results

Output 5.1 *Column Information for the Cars Data Set*

The CASUTIL Procedure

Table Information for Caslib MYCASLIB

Table Name	Label	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
CARS	2004 Car Data	428	15	utf-8	05Apr2016:15:23:33	05Apr2016:15:23:33	No	No	No	No

Detail Information for cars in Caslib MYCASLIB.

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated
ALL	1	1	428	68480	0	0	0	0	0	1	68480

Column Information for CARS in Caslib MYCASLIB

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

Results: MDSUMMARY Procedure

Output 5.2 *Summary of MSRP and City Miles per Gallon*

Summary of MSRP and City Miles Per Gallon

Obs	_Column_	_NObs_	_Mean_	_Max_	_Min_	_Std_
1	MSRP	428	32774.85514	192465	10280	19431.716674
2	MPG_City	428	20.060747664	60	10	5.2382176386

Additional Information

- Caslibs provide a way to organize in-memory tables and an associated data source. They also provide a way to apply access controls to data. After a caslib is dropped, the in-memory tables are dropped too. Files in the caslib's data source are not removed or modified in any way. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- Tables that are saved from a caslib are saved in SASHDAT format by default.
- For documentation about the CASUTIL procedure syntax, see Chapter 4, "CASUTIL Procedure," on page 57.
- For more examples of using the CASUTIL procedure to access and save data, see "Accessing Data" in *SAS Cloud Analytics 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;/*1*/

proc casutil incaslib="casuser" outcaslib="casuser" ;/*2*/
   load data=sashelp.cars;
   contents casdata="cars";
quit;

proc mdsummary data=mycas.cars;/*3*/
   var MPG_City;
   groupby / out=mycas.carsmpgcityall; /*4*/
   groupby make type / out=mycas.carsmaketype; /*5*/
run;

proc print data=mycas.carsmpgcityall;/*6*/
   var _Column_ _NObs_ _Mean_ _Std_ _Min_ _Max_;
   title 'Overall City Mileage';
run;

proc print data=mycas.carsmaketype;/**/
```

```
   var make Type  _Column_ _NObs_ _Mean_ _Std_ _Min_ _Max_;
   title 'City Mileage by Make and Type';
run;
```

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

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

3 The MDSUMMARY procedure produces summary statistics for MPG_CITY.

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

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

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

Results: MDSUMMARY Procedure

Output 5.3 *Overall Summary of City Mileage*

Overall City Mileage

Obs	_Column_	_NObs_	_Mean_	_Std_	_Min_	_Max_
1	MPG_City	428	20.060747664	5.2382176386	10	60

Output 5.4 *Summary of City Mileage by Make and Type*

City Mileage by Make and Type

Obs	Make	Type	_Column_	_NObs_	_Mean_	_Std_	_Min_	_Max_
1	Acura	SUV	MPG_City	1	17	.	17	17
2	Acura	Sedan	MPG_City	5	20.4	2.6076809621	18	24
3	Acura	Sports	MPG_City	1	17	.	17	17
4	Audi	Sedan	MPG_City	13	18.615384615	2.3642638579	14	23
5	Audi	Sports	MPG_City	4	19	2.7080128015	15	21
6	Audi	Wagon	MPG_City	2	16.5	2.1213203436	15	18
7	BMW	SUV	MPG_City	2	16	0	16	16
8	BMW	Sedan	MPG_City	13	19.230769231	0.8320502943	18	20
9	BMW	Sports	MPG_City	4	18.25	2.6299556397	16	21
10	BMW	Wagon	MPG_City	1	19	.	19	19
11	Buick	SUV	MPG_City	2	17	2.8284271247	15	19
12	Buick	Sedan	MPG_City	7	19.428571429	0.9759000729	18	20
13	Cadillac	SUV	MPG_City	2	15	1.4142135624	14	16
14	Cadillac	Sedan	MPG_City	4	18	0	18	18
15	Cadillac	Sports	MPG_City	1	17	.	17	17
16	Cadillac	Truck	MPG_City	1	13	.	13	13
17	Chevrolet	SUV	MPG_City	4	15.75	2.3629078131	14	19
18	Chevrolet	Sedan	MPG_City	15	22.266666667	4.0964560758	14	28
19	Chevrolet	Sports	MPG_City	2	18	0	18	18
20	Chevrolet	Truck	MPG_City	5	15.2	1.9235384062	13	18
21	Chevrolet	Wagon	MPG_City	1	22	.	22	22
22	Chrysler	Sedan	MPG_City	13	20.307692308	1.6525039276	18	22
23	Chrysler	Sports	MPG_City	1	17	.	17	17
24	Chrysler	Wagon	MPG_City	1	17	.	17	17
25	Dodge	SUV	MPG_City	1	15	.	15	15

Results: PROC CASUTIL CONTENTS Statement

Output 5.5 *Metadata for the Cars Table*

The CASUTIL Procedure

Table Information for Caslib CASUSER([redacted])

Table Name	Label	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Compressed
CARS	2004 Car Data	428	15	utf-8	04Apr2016:17:05:10	04Apr2016:17:05:10	No	No	No	No

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

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated
ALL	1	1	428	68480	0	0	0	0	0	1	68480

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

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

Additional Information

- Caslibs provide a way to access in-memory tables and an associated data source. They also provide a way to apply access controls to data. In this example, the personal caslib Casuser is being used, so no CASLIB statement is needed.
- Tables that are saved from a caslib are saved in SASHDAT format by default.
- For documentation about the CASUTIL procedure syntax, see Chapter 4, "CASUTIL Procedure," on page 57.
- 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

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

libname mycas cas;/*2*/
```

```
proc mdsummary data=mycas.cars ;/*3*/
   var MPG_City;
   groupby make model type / out=mycas.carsmiles;
run;

proc print data=mycas.carsmiles;/*4*/
   var Make Model Type _Column_ _NObs_ _Min_ _Max_;
   title 'City Mileage for Make, Model, and Type';
run;
```

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

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

3 The MDSUMMARY procedure produces summary statistics for MPG_CITY, grouped by Make, Model, and Type. The OUT= option creates an in-memory table named Mycas.CarsMiles. The table includes a row of summary statistics for each unique combination of MPG_City, Make, Model, and Type.

4 The PRINT procedure prints the table Mycas.CarsMiles.

Results

Output 5.6 *PROC MDSUMMARY Output: City Mileage for Make, Model, and Type*

City Mileage for Make, Model, and Type

Obs	Make	Model	Type	_Column_	_NObs_	_Min_	_Max_
1	Audi	A4 3.0 Quattro 4dr manual	Sedan	MPG_City	1	17	17
2	Audi	A41.8T convertible 2dr	Sedan	MPG_City	1	23	23
3	Audi	A6 2.7 Turbo Quattro 4dr	Sedan	MPG_City	1	18	18
4	Audi	A6 3.0 Quattro 4dr	Sedan	MPG_City	1	18	18
5	Audi	TT 1.8 convertible 2dr (coupe)	Sports	MPG_City	1	20	20
6	Audi	TT 3.2 coupe 2dr (convertible)	Sports	MPG_City	1	21	21
7	BMW	325i 4dr	Sedan	MPG_City	1	20	20
8	BMW	330Ci 2dr	Sedan	MPG_City	1	20	20
9	BMW	330xi 4dr	Sedan	MPG_City	1	20	20
10	BMW	525i 4dr	Sedan	MPG_City	1	19	19
11	BMW	M3 convertible 2dr	Sports	MPG_City	1	16	16
12	BMW	M3 coupe 2dr	Sports	MPG_City	1	16	16
13	BMW	Z4 convertible 2.5i 2dr	Sports	MPG_City	1	20	20
14	Buick	LeSabre Custom 4dr	Sedan	MPG_City	1	20	20
15	Buick	LeSabre Limited 4dr	Sedan	MPG_City	1	20	20
16	Buick	Regal GS 4dr	Sedan	MPG_City	1	18	18
17	Buick	Rendezvous CX	SUV	MPG_City	1	19	19
18	Cadillac	SRX V8	SUV	MPG_City	1	16	16
19	Chevrolet	Cavalier 4dr	Sedan	MPG_City	1	26	26
20	Chevrolet	Malibu LT 4dr	Sedan	MPG_City	1	23	23
21	Chevrolet	Silverado SS	Truck	MPG_City	1	13	13
22	Chevrolet	Tahoe LT	SUV	MPG_City	1	14	14
23	Chrysler	Concorde LX 4dr	Sedan	MPG_City	1	21	21
24	Chrysler	Concorde LXi 4dr	Sedan	MPG_City	1	19	19
25	Chrysler	PT Cruiser 4dr	Sedan	MPG_City	1	22	22

Additional Information

- Caslibs provide a way to access in-memory tables and an associated data source. They also provide a way to apply access controls to data. In this example, the personal caslib Casuser is being used, so no CASLIB statement is needed. The in-memory table Mycas.CarsMiles is temporary, and is dropped when the session is ended. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- For documentation about the CASUTIL procedure syntax, see Chapter 4, "CASUTIL Procedure," on page 57.
- 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.

```
libname mycas cas;/*1*/

proc format casfmtlib='fmtlib' ;/*2*/
   value $flvrfmt
      'Chocolate'='Chocolate'
      'Vanilla'='Vanilla'
      'Rum','Spice'='Other Flavor';
   value agefmt (multilabel)
      15 - 29='below 30 years'
      30 - 50='between 30 and 50'
      51 - high='over 50 years';
run;

data mycas.cake;/*3*/
input LastName $ 1-12 Age 13-14 PresentScore 16-17
TasteScore 19-20 Flavor $ 23-32 Layers 34 ;
format age agefmt. flavor $flvrfmt.;
datalines;
Orlando     27 93 80  Vanilla    1
Ramey       32 84 72  Rum        2
Goldston    46 68 75  Vanilla    1
Roe         38 79 73  Vanilla    2
Larsen      23 77 84  Chocolate  .
Davis       51 86 91  Spice      3
Strickland  19 82 79  Chocolate  1
Nguyen      57 77 84  Vanilla    .
Hildenbrand 33 81 83  Chocolate  1
Byron       62 72 87  Vanilla    2
Sanders     26 56 79  Chocolate  1
Jaeger      43 66 74             1
Davis       28 69 75  Chocolate  2
Conrad      69 85 94  Vanilla    1
Walters     55 67 72  Chocolate  2
Rossburger  28 78 81  Spice      2
Matthew     42 81 92  Chocolate  2
Becker      36 62 83  Spice      2
Anderson    27 87 85  Chocolate  1
Merritt     62 73 84  Chocolate  1
;

proc mdsummary data=mycas.cake; /*4*/
   var TasteScore;
   groupby flavor / out=mycas.flav; /*5*/
   groupby flavor age / out=mycas.flag; /*6*/
run;
```

```
proc print data=mycas.flav; /*7*/
   var flavor _Column_ _NObs_ _Min_ _Max_ _Mean_;
title 'Taste Score for Cake Flavors and Participant''s Age';
title2 'GROUPBY Flavor';
run;

proc print data=mycas.flag;
   var flavor age _Column_ _NObs_ _Min_ _Max_ _Mean_;
title 'Taste Score for Cake Flavors and Participant''s Age';
title2 'GROUPBY Flavor and Age';
run;
```

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

2 The FORMAT procedure creates the formats Flvrfmt and Agefmt. The CASFMTLIB= option adds the format library to the CAS session. It associates the format library with the CAS tables.

3 The DATA step creates the input data set. This DATA step runs in the SAS client session and not in CAS. However, the DATA step sends the results to CAS in the form of an in-memory CAS table. The CAS engine libref "Mycas" enables CAS processes to run on the data set.

4 The MDSUMMARY procedure computes summary statistics for cake tasting scores.

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

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

7 The PRINT procedure prints the output data sets.

Results

Output 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

Obs	Flavor	_Column_	_NObs_	_Min_	_Max_	_Mean_
1		TasteScore	1	74	74	74
2	Chocolate	TasteScore	9	72	92	81.444444444
3	Other Flavor	TasteScore	4	72	91	81.75
4	Vanilla	TasteScore	6	73	94	82.166666667

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

Taste Score for Cake Flavors and Participant's Age
GROUPBY Flavor and Age

Obs	Flavor	Age	_Column_	_NObs_	_Min_	_Max_	_Mean_
1	Chocolate	below 30 years	TasteScore	5	75	85	80.4
2	Chocolate	between 30 and 50	TasteScore	2	83	92	87.5
3	Other Flavor	below 30 years	TasteScore	1	81	81	81
4	Other Flavor	over 50 years	TasteScore	1	91	91	91
5	Vanilla	below 30 years	TasteScore	1	80	80	80
6	Chocolate	over 50 years	TasteScore	2	72	84	78
7	Vanilla	between 30 and 50	TasteScore	2	73	75	74
8	Vanilla	over 50 years	TasteScore	3	84	94	88.333333333
9		between 30 and 50	TasteScore	1	74	74	74
10	Other Flavor	between 30 and 50	TasteScore	2	72	83	77.5

Additional Information

- Caslibs provide a way to access in-memory tables and an associated data source. They also provide a way to apply access controls to data. In this example, the personal caslib Casuser is being used, so no CASLIB statement is needed. The in-memory tables Mycas.Flav and Mycas.Flag are temporary, and are dropped when the session ends. To add tables to a data source permanently, use the SAVE statement in PROC CASUTIL.
- For documentation about the CASUTIL procedure syntax, see Chapter 4, "CASUTIL Procedure," on page 57.
- 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: Graph Summary Statistics Results Obtained from the MDSUMMARY Procedure

Program

The following example loads data into CAS, and creates a plot from the summarized results of the MDSUMMARY procedure.

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

libname mycas cas;/* 2 */

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

ods graphics / width=4in;
```

```
title "Summarized Highway MPG";
proc sgpanel data=mycas.mpghw_sum;                    /* 4 */
   where origin in ("Asia" "USA");
   panelby origin / uniscale=row;
   format _mean_ 2.;
   vbar type / response=_mean_;
   rowaxis label="Summary MPG Values";
run;
title;
ods graphics / reset=all;
```

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

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

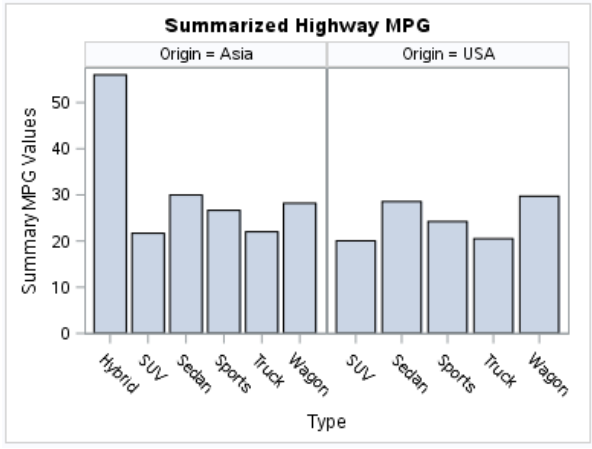
3 The MDSUMMARY procedure produces summary statistics for highway miles-per-gallon. The OUT= option in the GROUPBY statement creates a table in CAS. The table includes a row of summary statistics for each unique combination of origin and type.

4 The SGPANEL procedure plots the summarized results from the MDSUMMARY procedure. The procedure creates a parameterized vertical bar chart that shows the mean statistic for highway miles-per-gallon. The procedure subsets the data, comparing only the cars made in Asia and the U.S.A. The graph is paneled by country of origin.

Results

The SGPANEL procedure generates the following graph:

Output 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 57.
- 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" in *SAS Viya ODS Graphics: Procedures Guide*.

Chapter 6

Platform Data Sources

Data Redundancy

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

Table 6.1 *Data Redundancy by Data Access Method, Data Source, and File Type*

Data Access Method	Caslib Data Source	Redundancy		File Formats
		SASHDAT files	Other files	

DATA step and
`PROC CASUTIL;`
`LOAD DATA=`
	The caslib data source isn't a factor for these data access methods.	Not applicable.	Based on COPIES= when the file is loaded.	SAS Data Sets

Data Access Method	Caslib Data Source	Redundancy		File Formats
		SASHDAT files	Other files	
PROC CASUTIL; LOAD CASDATA=	DNFS	COPIES= is not needed, surviving nodes reread the file.	Based on COPIES= when the file is loaded.	SASHDAT and CSV files
	HDFS	Based on COPIES= when the file was saved.	Based on COPIES= when the file is loaded.	SASHDAT and CSV files
	PATH	Based on COPIES= when the file is loaded.	Based on COPIES= when the file is loaded.	SASHDAT, CSV, and PC Files
	Data Connectors	Not applicable.	Based on COPIES= when the file is loaded.	Based on the data connector.

Note: Tables in DNFS caslibs that have been appended to or updated do not support worker failover. If you have a file to load that is subsequently updated (including appending records) you can use a PATH caslib rather than DNFS. PATH caslibs provide failover support for appends and updates.

Dictionary

HDFS Data Source

Specifies a Hadoop Distributed File System directory for loading and saving files that the SAS Cloud Analytic Servicescontroller can access. SAS Cloud Analytic Services must be co-located with the Hadoop cluster to use this data source type.

Applies to: Distributed servers only, CASLIB statement

Example: Add a caslib to access files from the **/vapublic** directory in HDFS.

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

proc casutil incaslib="public" outcaslib="public";
   list files;
quit;
```

Syntax

Data Source Arguments

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

PATH="*/directory-path*"
specifies the fully qualified path to the directory to use as a data source. Notice that the PATH= argument is specified outside of the parenthesis for the DATASOURCE= argument.

SRCTYPE="HDFS"
specifies that the data source is an HDFS directory that is co-located with SAS Cloud Analytic Services.

Requirement The SRCTYPE= argument is required.

DNFS Data Source

Specifies a server-side directory for loading and saving files that the SAS Cloud Analytic Services controller can access. The directory must be mounted by every machine that is used by the server.

Applies to: Distributed servers only, CASLIB statement

Example: Add a caslib to access files from the `/data01` directory on the controller.

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

proc casutil incaslib="dnfsds" outcaslib="dnfsds";
   list files;
quit;
```

Syntax

Data Source Arguments

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

PATH="*/directory-path*"
specifies the fully qualified path to the directory to use as a data source. Notice that the PATH= argument is specified outside of the parenthesis for the DATASOURCE= argument.

SRCTYPE="DNFS"
specifies that the data source is a directory that is mounted by every machine that is used for SAS Cloud Analytic Services.

Requirement The SRCTYPE= argument is required.

Details

DNFS is an abbreviation for distributed network file system. This data source type provides support for distributed data access to NFS directories. Several systems such as MapR-FS, EMC Isilon, and others provide high-availability, replicated, high-performance, stand-alone storage clusters with an NFS interface. These systems offer

popular alternatives to Hadoop. DNFS provides a good alternative for deployments where the server is not co-located with Hadoop and yet must provide similar capabilities.

DNFS can also be used to access NFS-mounted directories from standard UNIX or Linux file systems.

The design principle is that NFS-mounted directories are accessed concurrently by each controller node and worker node in a distributed server. This is why the directory path for a DNFS caslib must be mounted on every machine. DNFS performs parallel read and write for SASHDAT and CSV files that are stored in the directory path specified for the caslib.

SAS LASR Analytic Server

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

Applies to: CASUTIL procedure, CASLIB statement

Examples: Add a caslib to access data from a SAS LASR Analytic Server.

```
caslib publiclasr datasource=(
  srctype="lasr"
  server="gridhost.example.com"
  port=10050
  signer="https://webserver.example.com/SASLASRAuthorization"
  username="sasdemo"
  password="secret"
);
```

Load a table from a SAS LASR Analytic Server into SAS Cloud Analytic Services.

```
proc casutil incaslib="publiclasr";
   load casdata="epa_cars"
        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.

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

GRIDRSHCOMMAND Environment Variable

If you do not specify the SIGNER= option when you add the caslib, then you must specify the GRIDRSHCOMMAND environment variable in the casconfig.lua configuration file:

```
env.GRIDRSHCOMMAND="/usr/bin/ssh"
```

Make sure that you specify the path to the ssh command for your Linux deployment. The path that is shown in the example might not be correct for your distribution or release.

Arguments Summary

Table 6.2 *Summary of Arguments for SAS LASR Analytic Server*

Argument	Valid Data Source Options in the CASLIB Statement	Valid Import Options in the CASUTIL Procedure
COMPPGM=		•
COMPVARS=		•
FILETYPE=		Required
METALIB=	•	
PARALLELMODE=		•
PASSWORD=	•	•
PORT=	Required	•
PRESERVEORDER=		•
SRCTYPE=	Required	
SERVER=	Required	
SIGNER=	•	
USERNAME=	•	•
VARCHARS=		•
VARS=		•
WHERE=		•

Example

This example shows how to load a table from a SAS LASR Analytic Server and use the VARS= options. The sample Epa_cars data is available from the SAS Visual Analytics 7.3 documentation page at http://support.sas.com/documentation/onlinedoc/va/index.html.

```
caslib valasr datasource=(
  srctype="lasr"
  /* ...data source options... */
);

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


proc casutil;
   list files;                                                   /* 2 */


   load casdata="epa_cars" casout="epa_cars"
        importoptions=(
           filetype="lasr"
           varchars="true"
           vars=("model_year" "vehicle_manufacturer_name")       /* 3 */
        );

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

1 The CAS statement is used to set the active caslib for the Casauto session to Valasr explicitly.

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

3 The VARS= option is used to subset the columns that are read from SAS LASR Analytic Server and loaded into SAS Cloud Analytic Services.

Path Data Source

Specifies a server-side directory for loading and saving files that the SAS Cloud Analytic Services controller can access.

Applies to: CASLIB statement

Example: Add a caslib to access files from the **/data01** directory on the controller.

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

proc casutil incaslib="pathds" outcaslib="pathds";
   list files;
quit;
```

Syntax

Data Source Arguments

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

PATH="*/directory-path*"
specifies the fully qualified path to the directory to use as a data source. Notice that the PATH= argument is specified outside of the parenthesis for the DATASOURCE= argument.

SRCTYPE="PATH"
specifies that the data source is a directory that is accessible to the SAS Cloud Analytic Services controller.

Requirement The SRCTYPE= argument is required.

Chapter 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.

```
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*

```
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.

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( )			
Column	**Type**	**Length**	**Formatted Length**
sexe	varchar	8	8
nom	varchar	10	10
âge	double	8	12
la taille	double	8	12
poids	double	8	12

Output 7.2 *Three Rows from the Class_Fr Table*

Obs	sexe	nom	âge	la taille	poids
1	Masculin	André	14	69.0	112.5
2	Masculin	Benoît	14	63.5	102.5
3	Masculin	Kévin	12	57.3	83.0

SASHDAT Files

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

Applies to: CASUTIL procedure

Example: Load an encrypted SASHDAT file into SAS Cloud Analytic Services.

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

Syntax

File Type Arguments

FILETYPE="HDAT"
specifies the file type.

Requirement The FILETYPE= argument is required.

ENCRYPTIONPASSWORD="*string*"
specifies a password for encrypting or decrypting stored data. You can use this option to override an ENCRYPTIONPASSWORD= value that is set with the CASLIB statement.

Details

SASHDAT files can be read from caslibs with the following data source types:

- DNFS
- HDFS
- PATH

Instead of specifying the ENCRYPTIONPASSWORD= option on a file-by-file basis, you can use it as a data source option when you add a caslib. For an example, see Encrypt Tables in a Caslib on page 53.

Chapter 8

Data Connectors

Working with SAS Data Connectors

To access and load data, the administrator must create and configure data connectors for SAS Cloud Analytic Services (CAS). Data connectors contain connection information and data-source specifics to connect with data sources, such as Oracle or SAS data sets.

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. 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. Authorizations that are defined on a third-party database overrule permissions and privileges that are set in CAS.

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 "DATASOURCE=(SRCTYPE="*type*", *data-source-options*, ENCRYPTIONPASSWORD="*string*")" on page 40.

Table 8.1 *Data Source Types and Options*

srcType= Type	Option Syntax	Example

hadoop	"SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop" on page 120	`caslib hvlib desc="Hadoop Caslib"`
		`datasource=(srctype="hadoop",`
		`dataTransferMode="parallel",`
		`hadoopjarpath="/data/cdh54/sdm/lib",`
		`hadoopconfigdir="/data/cdh54/sdm/conf",`
		`username="hadoopuser",`
		`server="hive01.example.com",`
		`schema="default");`

impala	"SAS Data Connector to Impala" on page 129	`caslib imlib desc="Impala Caslib"`
		`datasource=(srctype="impala",`
		`username="impalauser",`
		`server="impala01.example.com");`

odbc	"SAS Data Connector to ODBC" on page 136	`caslib odbccaslib desc="ODBC caslib"`
		`datasource=(srctype="odbc"`
		`username="user1"`
		`password="password1"`
		`database="dbodbc"`
		`catalog="*");`

oracle	"SAS Data Connector to Oracle" on page 143	`caslib oraclecaslib desc="Oracle Caslib"`
		`datasource=(srctype="oracle",`
		`username="user1",`
		`password="password1",`
		`path="//machine.lnx.com:5570/exadat");`

postgres	"SAS Data Connector to PostgreSQL" on page 151	`caslib postgrescaslib desc='PostgreSQL Caslib'`
		`datasource=(srctype='postgres'`
		`username='user1'`
		`password='password1'`
		`server="postgresServer"`
		`database="postgresDatabase");`

srcType= Type	Option Syntax	Example

teradata	"SAS Data Connector to Teradata and SAS Data Connect Accelerator for Teradata" on page 160	`caslib TDcaslib desc='Teradata Caslib'`
		`datasource=(srctype='teradata'`
		`username='user1'`
		`password='password1'`
		`server="teradataServer");`

Loading a Data Source

When you load a data source, you specify the type of data source that CAS is connecting to. However, a connection is not made until you access data in the data source.

There are two ways to load a data source. To load the data source only, call the loadDataSource action. To load a data source and add a caslib that accesses that data source, use the CASLIB statement. If you use the CASLIB statement, you do not also need to call the loadDataSource action.

Where to Specify Data Connector Parameters

The parameters that you specify for your data connector are specific to your data source. In general, you specify parameters for data connectors in the following places in your CAS code.

When you add a caslib, specify data connector parameters within the dataSource= parameter. Use the dataSource= parameter in the CASLIB statement or in calls to the addCaslib CAS action. 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:

```
/* CASLIB statememt */
caslib simple
  sessref=mysess
  dataSource=(srcType="teradata",
              dataTransferMode="parallel",
              server="teraserver",
              database="teradatabase",
              schema="mySchema");
```

If the user name is myUser and the password is myPwd, then the caslib can be added as follows using Lua code:

```
/* Call to addCaslib in Lua code */
s:addCaslib{lib="tdlib",
   dataSource={srcType="teradata",
      dataTransferMode="parallel",
      username="myUser",
      password="myPwd",
      server="teraserver",
      database="teradatabase"}}
```

For PROC CASUTIL, specify data connector parameters within the dataSourceOptions= parameter for a CONTENTS, LIST, or LOAD statement.

```
proc casutil;
   list files incaslib="tdlib" dataSourceOptions=(schema="tdSchema");
   contents casdata="cars" vars((name="make"), (name="model"))
      dataSourceOptions=(schema="tdSchema2");
   load casdata="cars" incaslib="tdlib" casout="cars_CAS"
      vars=((name="make"), (name="model"))
      dataSourceOptions=(dbmswhere="cylinders=8");
quit;
```

For PROC CAS, specify data connector parameters within the dataSourceOptions= parameter for a table actions. The table actions that receive data connector parameters are columnInfo, fileInfo, and loadTable. Specify data connector parameters in the dataSource option for the addCaslib action. (See above.)

```
proc cas;
   action fileInfo / casLib="tdlib"
             dataSourceOptions={catalog="Study1", schema="mySchema"};
   action columnInfo /
             dataSourceOptions={schema="mySchema"};
   action loadTable /
      casLib="tdlib"
      path="cars"
      casout="cars_CAS"
      vars={{name="make"},{name="model"}}
      dataSourceOptions={dbmsWhere="cylinders=4",
                         dataTransferMode="parallel"};
run;
quit;
```

Note: It is not common to specify data connector options in all statements in PROC CAS. These are shown to demonstrate where you can specify data connector options in the code.

See Also

Syntax

- "CASLIB Statement" on page 39
- "Add caslib" in *SAS Cloud Analytic Services: System Programming Guide*
- Chapter 4, "CASUTIL Procedure," on page 58
- "CAS" in *SAS Cloud Analytic Services: CAS Procedure Programming Guide and Reference*

Data Connectors

- "SAS Data Connector to Hadoop and SAS Data Connect Accelerator for Hadoop" on page 120
- "SAS Data Connector to Impala" on page 129
- "SAS Data Connector to ODBC" on page 136
- "SAS Data Connector to Oracle" on page 143
- "SAS Data Connector to PC Files" on page 148

- "SAS Data Connector to PostgreSQL" on page 151
- "SAS Data Connector to SAS Data Sets" on page 157
- "SAS Data Connector to Teradata and SAS Data Connect Accelerator for Teradata" on page 160

Using Wildcards for Pattern Matching

Overview of Wildcards

There are instances when you might use wildcards to facilitate pattern matching. For example, you might use wildcards to see a listing of all tables that begin with the string "aBc". To do this, you might specify the following call to fileInfo:

```
s.fileInfo / 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_':

```
s.fileInfo / 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'`**.

Additional Parameter That Affects Pattern Matching

By default, pattern matching is case-specific. That is, the pattern 'aBc' matches only 'aBc' and not 'ABC' or 'abc'. However, you can use the wildsensitive= parameter to specify whether matches are case sensitive or case insensitive. By default, wildsensitive=true, so that matching is case sensitive. If you specify wildsensitive=false, then matching is not case sensitive. When wildsensitive=false, the pattern 'aBc' matches with 'abc', 'aBc', and 'ABC'.

To execute a case-insensitive search, you might execute this code:

```
table.fileInfo / path='abc%' wildsensitive=false;
```

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 to use depends on the type of source data that you are accessing.

If you are loading data from a SAS table or an external file, such as an Excel spreadsheet, then you can use the WHERE option for the LOAD statement in PROC CASUTIL, or you can use the where= parameter for the loadTable action.

```
action loadTable / caslib="mydata",
   path="cars",
   vars={{name="make"},{name="model"}},
   where="price > 20000";
```

If you want to load a subset of data from an external database table, such as PostgreSQL or Teradata, then use the dbmsWhere= data connector parameter. With the dbmsWhere= parameter, the WHERE clause that you specify is passed directly to the external database for use while loading the data.

```
action loadTable / caslib="tdlib", path="myTDtab",
   options={dbmsWhere="salary > 50000 and
   sales < 75000"};
```

Note: Using the WHERE option in a LOAD statement or using the where= parameter for a loadTable action results in an error for an external database.

See Also

- "Load table" in *SAS Cloud Analytic Services: System Programming Guide*
- "LOAD Statement" on page 63

Dictionary

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 parameters for details)

Restriction: The ARRAY, MAP, STRUCT, UNION, or BOOLEAN data types are not supported.

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.

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

See: For configuration instructions, see the *SAS Hadoop Configuration Guide for Base SAS and SAS/ACCESS*.

Chapter 9, “Data Types,”

Examples: Load a Hadoop data source and define a casref to Hadoop.

```
caslib hvlib sessref=mysess
  datasource=(srctype='hadoop',
              dataTransferMode='parallel',
              server='HiveServer',
              username='user1'
              hadoopJarPath="<Hadoop jar path directory>",
              hadoopConfigDir="<Hadoop configuration directory>",
              schema="<Hive schema name>");
```

Load a Hadoop data source using PROC CASUTIL.

```
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 parameter described, the applicable statements and CAS actions where you can use that parameter are indicated. For information about where to specify these parameters within statements and action calls, see “Where to Specify Data Connector Parameters” on page 117.

bufferSize=*bytes*

specifies in bytes the buffer size length that is used to receive data from SAS embedded processes. This value overrides the bufferSize= value that was set in the CASLIB statement or with the addCaslib action. Increasing the size might result in better performance with a trade-off of increased memory usage.

Valid in	CASLIB statement
	CAS action: addCaslib, loadTable
	PROC CASUTIL: LOAD OPTIONS= statement
Default	1048576
Restriction	This parameter applies only when dataTransferMode="parallel" for SAS Data Connect Accelerator for Hadoop.

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 fileInfo action or from the CONTENTS statement in PROC CASUTIL.

Valid in	CASLIB statement
	CAS action: addCaslib
Default	active caslib
Restriction	This option applies only when dataTransferMode="serial" for SAS Data Connector to Hadoop.

charMultiplier=*double*
specifies a common option for the Multiplier for the character column length for nonwide character types.

Default	1
Restriction	This does not apply to Hadoop because all Hive tables are UTF-8.

client_encoding="*encoding*"
specifies the DBMS encoding type.

Valid in	CASLIB statement
	CAS action: addCaslib
Default	UTF-8
Restriction	This option applies only with a SAS Data Connector (dataTransferMode="serial").

database="*Hive database name*"
specifies the name of the database.

Valid in	CASLIB statement
	CAS actions: addCaslib, loadTable
	PROC CASUTIL: LOAD statement
Alias	db=
Default	"" (zero-length 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 or in the addCaslib action. Here are the valid values.

auto	specifies to first try to load the data in parallel (using embedded processing). If this fails, an error is issued and serial processing is then attempted.
parallel	specifies to always use only parallel (using embedded processing).
serial	specifies to load the data serially by using the SAS Data Connector to your database.

Valid in CASLIB statement

CAS actions: addCaslib, loadTable

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.

dbmsWhere="*WHERE-clause*"

specifies a database-specific SQL WHERE clause to submit to the database. The WHERE clause that you specify is passed to the database exactly as you enter it. This option is used to filter the rows that are read into a CAS table.

Valid in CAS actions: loadTable

PROC CASUTIL: LOAD statement

Default "" (zero-length string)

Example

```
action loadTable / caslib="tdlib", path="myTDtab",
  dataSourceOptions={dbmsWhere="make='Chevrolet' and
  MSRP < 18000"};
```

hadoopConfigDir="*configuration files directory*"

specifies the Hadoop configuration files directory, which is the one that is obtained by running the Hadoop tracer tool on the target Hadoop cluster.

Valid in CASLIB statement [required]

CAS actions: addCaslib [required]

PROC CASUTIL: LOAD statement

Default "" (zero-length string)

hadoopJarPath="*jar files path*"

specifies one or more paths to the Hadoop JAR files. These are the JAR files that are obtained by running the Hadoop tracer tool on the target Hadoop cluster. These files are delimited by colons for Linux.

Valid in CASLIB statement [required]

CAS actions: addCaslib [required]

PROC CASUTIL: LOAD statement

Default "" (zero-length string)

hdfsTempDir="*Hadoop HDFS temporary directory*"
specifies the Hadoop HDFS directory to use to store temporary data.

Valid in CASLIB statement [required]

CAS actions: addCaslib [required]

PROC CASUTIL: LOAD statement

Default "/tmp"

name="*source name*"
specifies the type of data source.

Valid in CAS actions: loadDataSource [required]

Default none

Examples

```
s:loadDatasource{name='hadoop'}
```

```
proc cas;
   session mysess;
   action loadDatasource / name='hadoop';
run;
```

password="*password*"
specifies the DBMS password for a user. You typically specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply a username= and password= value in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in CASLIB statement [see Requirement]

CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Aliases pass=

pwd=

Default "" (zero-length string)

Requirement If your database requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

path="*Hive table path and name*"
specifies the name of a Hive source table. When used with the fileInfo action, you can use wildcards to match a specific set of file names. For more information, see "Using Wildcard Characters for Filename Matching" on page 71.

Valid in CAS actions: fileInfo [required], loadTable [required]

Default none

port=*port value*
specifies the Hive JDBC port number.

Default 10000

Range 1–65535

properties="*Hive JDBC properties value*"
specifies a free-form value for Hive JDBC properties. The value is appended to the JDBC connection URI. You can use it to override default Hive behaviors.

Default none

readBuff="*integer*"
specifies the number of rows to fetch per block of data when fetching in serial data transfer mode.

Valid in CASLIB statement

CAS actions: addCaslib, loadTable

PROC CASUTIL: LOAD statement

Aliases RAS=

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.

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 or for an action other than the addCaslib action, then this value overrides any value of schema= that was set when the caslib was defined.

Valid in CASLIB statement

CAS actions: addCaslib, fileInfo, columnInfo, loadTable

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Default "" (zero-length string)

server="*Hive server identifier*"
specifies the Hive server name that runs the Hive service.

Valid in CASLIB statement [required]

CAS actions: addCaslib [required]

Default "" (zero-length string)

srcType="*hadoop*"
specifies that the data source is Hadoop (Hive database).

Valid in CASLIB statement [required]

CAS actions: addCaslib [required]

Default "" (zero-length string)

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 or in the addCaslib action.

Valid in CASLIB statement

CAS actions: addCaslib, loadTable

PROC CASUTIL: LOAD statement

Default 0 (no message)

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.

Default none

Interaction If you use this option, options that alter the JDBC URI (such as properties=) are ignored.

username="*user-name*"
specifies the database user name.

Typically, you specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply username= and password= values in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in CASLIB statement [see Requirement]

CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Aliases	uid=
	user=
Default	none
Requirement	If it is required by your database, then you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

Details

Use Options

Data connector options are used in the context of different statements and CAS actions that connect your data in Hadoop with SAS Cloud Analytic Services. For each described option, the applicable statements and CAS actions 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 Parameters" on page 117.

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.

Load Hadoop Data in Parallel

If the SAS Data Connect Accelerator for Hadoop is installed on your system, you can use it to load data in parallel, using dataTransferMode=. When you load data in parallel, the data connect accelerator uses the Hive database's hash distribution of the data to spread the data across multiple connections for parallel loading into SAS Cloud Analytic Services. The data distribution in the database is determined by the Hive Primary Index (PI) for tables or by a hash that is calculated by the database for Hive views. Talk to your Hadoop administrator or review the Hadoop user documentation to get more information about how Hadoop distributes data across its units of parallelism (AMPs). The more evenly the data is divided, the more efficiently the data can be loaded into SAS Cloud Analytic Services.

Add a SAS Cloud Analytic Services Library for the Hadoop Data Source

The addCasLib action adds a new SAS Cloud Analytic Services library to the SAS Cloud Analytic Services session. Using a Hadoop data source requires a datasource option on the addCaslib action that describes the connection information for that data source. The datasource option consists of a list of name/value pairs.

Load a Connection to a Hadoop Data Source

When you load a connection to a data source, you specify the type of data source that SAS Cloud Analytic Services is connecting to. However, a connection is not made until you load data from the data source.

You can load a connection to a data source in two ways. To load only the connection, call the loadDataSource action. To load the connection to the data source and to define a caslib that accesses that data source, use the CASLIB statement. If you use the CASLIB statement, you do not also need to call the loadDataSource action.

Supported Hive Data Types

This table shows the data types that can be read into SAS Cloud Analytic Services from a Hive 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 8.2 *Supported Hive Data Types*

Hive Data Type	SAS Cloud Analytic Services Data Type	Default Format in SAS Cloud Analytic Services
CHAR	CHAR	*w.*
VARCHAR	VARCHAR	*w.*
DOUBLE	DOUBLE	*w.d*

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 Hive Database as a Data Source for a Caslib 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 database until an action or statement that accesses the data is called. The data is read serially into the caslib Hadoopcaslib.

```
caslib hvlib desc='Hadoop Caslib'
   dataSource=(srctype='Hadoop',
               dataTransferMode='serial',
               server='hiveServer',
               username='user1',
               password='******',);
```

Example 2: Load a Hive Table into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="Hadoopcaslib";                    1
   load casdata="myHDdata" incaslib="Hadoopcaslib" outcaslib="casuser"
        casout="HDdata_from_Hadoopcaslib";                2
   list tables incaslib="casuser";                        3
   contents casdata="HDdata_from_Hadoopcaslib" incaslib="casuser";
                                                          4
quit;
```

1 List the tables in Hadoopcaslib before loading your data.

2 Load the table myHDdata from Hive into caslib Casuser. Call the new table HDdata_from_Hadoopcaslib.

3 List the tables in caslib 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 3: Using Lua to Connect to Hive, Examine the Data, and Load a Table

```
-- Load the Hadoop data source
r=s:loadDatasource{name='hadoop'}

-- Add the Caslib for Hadoop
r = s:addCaslib{lib='hvlib',
   datasource={srcType='hadoop',
                        dataTransferMode='parallel',
                        server='Hive server name',
                        username='user1',
                        password='******',
                        schema='Hive schema name'
                       }
           }

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 4: Add a Caslib with the addCaslib Action

```
/* Using PROC CAS */
proc cas;
   session mysess;
   action addCaslib / caslib="hvlib"
                      session=false
                      dataSource={srctype="hadoop",
                                  username="Hive username",
                                  password="Hive password",
                                  server="Hive server name",
                                  schema="Hive schema name"};
run;
quit;
```

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 define a casref to Impala.

```
caslib imlib sessref=mysess
  datasource=(srctype='impala',
              server='impala01.example.com',
              username='user1'
              password='myPwd'
              path="//machine.lnx.com:1521/exadat");

proc cas;
session mysess;
action loadDatasource / name="impala";
run;
quit;
```

Load an Impala data source using PROC CASUTIL.

```
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 parameter described, the applicable statements and CAS actions where you can use that parameter are indicated. For information about where to specify these parameters within statements and action calls, see “Where to Specify Data Connector Parameters” on page 117.

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 fileInfo action or from the CONTENTS statement in PROC CASUTIL.

Valid in	CASLIB statement
	CAS action: addCaslib
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 or in the addCaslib action.

Valid in	CASLIB statement
	CAS actions: addCaslib, loadTable
	PROC CASUTIL: LOAD statement
Default	1.0
Range	1.0–5.0

client_encoding="*encoding*"
specifies the DBMS encoding type.

Valid in	CASLIB statement
	CAS action: addCaslib
Default	UTF-8
Restriction	This option applies only with a SAS Data Connector (dataTransferMode="serial").

dm_unicode="*unicode setting*"
specifies the unicode encoding for the driver manager. Possible values include UTF-8, UCS-2, and so on. This setting applies to Linux platforms when using third-party ODBC driver managers, such as unixODBC.

Valid in	CASLIB statement
	CAS actions: addCaslib, loadTable
	PROC CASUTIL: LOAD statements
Default	UTF-8

driver_vendor="*driver vendor name*"
specifies the name of the specific third-party vendor that supports the Impala driver.

Valid in	CASLIB statement
	CAS actions: addCaslib

impala_dsn="*Impala datasource name*
specifies the Impala data source name. With this option, you can configure an Impala data source in an odbc.ini file.

Valid in	CASLIB statement
	CAS actions: addCaslib
Default	"" (zero-length string)
Requirement	Set the ODBCSYSINI environment variable to point to the directory that contains the configured odbc.ini file.

name="*source name*"
specifies name of the data source.

Valid in	CAS actions: loadDataSource [required]
Default	none
Examples	`s:loadDatasource{name='impala'}`

```
proc cas;
   session mysess;
   action loadDatasource / name='impala';
run;
```

password="*password*"
specifies the DBMS password for a user. You typically specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply a username= and password= value in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in	CASLIB statement [see Requirement]
	CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Aliases	pass=
	pwd=
Default	"" (zero-length string)
Requirement	If your database requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

path="*Impala table path and name*"
specifies the name of a Impala source table. When used with the fileInfo action, you can use wildcards to match a specific set of file names. For more information, see "Using Wildcard Characters for Filename Matching" on page 71.

Valid in	CAS actions: fileInfo [required], loadTable [required]
Default	none

port="*port value*"
specifies the port to use to connect to the Impala database.

Valid in	CASLIB statement
	CAS action: addCaslib
Default	21050
Range	1–21050

readBuff="*integer*"
specifies the number of rows to fetch per block of data.

Valid in	CASLIB statement
	CAS actions: addCaslib, loadTable
	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.

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 or for an action other than the addCaslib action, then this value overrides any value of schema= that was set when the caslib was defined.

Valid in	CASLIB statement
	CAS actions: addCaslib, fileInfo, columnInfo, loadTable
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Default	"" (zero-length string)

server="*Impala server identifier*"
specifies the server identifier for the Impala server.

Valid in	CASLIB statement [required]
	CAS actions: addCaslib [required]
Default	"" (zero-length string)

srcType="*impala*"
specifies that the data source is an Impala database.

Valid in	CASLIB statement [required]
	CAS actions: addCaslib [required]
Default	"" (zero-length string)

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 or in the addCaslib action.

Valid in	CASLIB statement
	CAS actions: addCaslib, loadTable
	PROC CASUTIL: LOAD statement

Default	0 (no message)

username="*user-name*"
specifies the database user name.

Typically, you specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply username= and password= values in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in	CASLIB statement [see Requirement]
	CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Aliases	uid=
	user=
Default	none
Requirement	If it is required by your database, then you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

Details

Use Options

Data connector options are used in the context of different statements and CAS actions that connect your data in Impala with SAS Cloud Analytic Services. For each described option, the applicable statements and CAS actions 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 Parameters" on page 117.

Impala Naming

The data connector and data connect accelerator can load Impala tables with names up to 128 characters or with column names that are up to 128 characters.

Add a SAS Cloud Analytic Services Library for the Impala Data Source

The addCasLib action adds a new SAS Cloud Analytic Services library to the SAS Cloud Analytic Services session. Using an Impala data source requires a datasource option on the addCaslib action that describes the connection information for that data source. The datasource option consists of a list of name/value pairs.

Load a Connection to an Impala Data Source

When you load a connection to a data source, you specify the type of data source that SAS Cloud Analytic Services is connecting to. However, a connection is not made until you load data from the data source.

You can load a connection to a data source in two ways. To load only the connection, call the loadDataSource action. To load the connection to the data source and to define a caslib that accesses that data source, use the CASLIB statement. If you use the CASLIB statement, you do not also need to call the loadDataSource action.

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 8.3 *Supported Impala Data Types*

Impala Data Type	SAS Cloud Analytic ServicesData Type	Default Format in SAS Cloud Analytic Services
CHAR	CHAR	*w.*
VARCHAR	VARCHAR	*w.*
DOUBLE	DOUBLE	*w.d*

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

```
caslib imlib desc='Impala Caslib'
   dataSource=(srctype='Impala',
               server='impalaServer',
               username='user1',
               password='******',);
```

Example 2: Load an Impala Table into SAS Cloud Analytic Services Using PROC CASUTIL

```
proc casutil;
   list files incaslib="Impalacaslib";                    1
   load casdata="myIMdata" incaslib="Impalacaslib" outcaslib="casuser"
        casout="IMdata_from_Impalacaslib";                2
   list tables incaslib="casuser";                        3
   contents casdata="IMdata_from_Impalacaslib" incaslib="casuser";
                                                          4
quit;
```

1 List the tables in Impalacaslib before loading your data.

2 Load the table myIMdata from 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

```
-- 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='user1',
                        password='******',
                        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)
```

Example 4: Add a Caslib with the addCaslib Action

```
/* Using PROC CAS */
proc cas;
   session mysess;
   action addCaslib / caslib="imlib"
                      session=false
                      dataSource={srctype="impala",
                                  username="Impala username",
                                  password="Impala password",
                                  server="Impala server name",
                                  schema="Impala schema name"};
run;
quit;
```

SAS Data Connector to ODBC

SAS Data Connector to ODBC specifies the arguments to use when loading data from an ODBC data source into SAS Cloud Analytic Services. Data connector parameters are used in the context of different statements and CAS actions that connect your data in ODBC with CAS.

Valid in: CASLIB statement

PROC CASUTIL: LOAD statement

Example: Establish a connection to ODBC data. (Note: The GLOBAL option is restricted to administrators.)

```
caslib odbccaslib desc="SQLviaODBCtoCaslib"
     datasource=(srctype="odbc"
                      username="user1"
                      password="password1"
                      odbc_dsn="dbodbc") global;
```

Syntax

Data Connector Parameters for ODBC

For each parameter described, the applicable statements and CAS actions where you can use that parameter are indicated. For information about where to specify these parameters within statements and action calls, see "Where to Specify Data Connector Parameters" on page 117.

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 fileInfo action or 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

CAS action: addCaslib

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 or in the addCaslib action.

Valid in CASLIB statement

CAS actions: addCaslib, loadTable

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

CAS action: addCaslib, loadTable

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

CAS action: addCaslib

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-specific SQL WHERE clause to submit to the database. The WHERE clause that you specify is passed to the database exactly as you enter it. This option is used to filter the rows that are read into a CAS table.

Valid in CAS actions: loadTable

PROC CASUTIL: LOAD statement

Default "" (zero-length string)

Example

```
action loadTable / caslib="tdlib", path="myTDtab",
  dataSourceOptions={dbmsWhere="make='Chevrolet' and
  MSRP < 18000"};
```

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

CAS actions: addCaslib, loadTable

PROC CASUTIL: LOAD statement

Default UTF-8

name="*odbc*"
specifies the name of a DBMS type.

Valid in	CAS action: loadDataSource [Required]
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]
	CAS action: addCaslib [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 when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply a username= and password= value in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in	CASLIB statement [see Requirement]
	CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Aliases	pass=
	pwd=
Default	"" (zero-length string)
Requirement	If your database requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

readBuff="*number-of-rows*"
specifies the number of rows to fetch per block of data retrieved.

Valid in	CASLIB statement
	CAS actions: addCaslib, loadTable
	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 or for an action other than the addCaslib action, then this value overrides any value of schema= that was set when the caslib was added.

Valid in CASLIB statement

CAS actions: addCaslib, fileInfo, columnInfo, loadTable

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]

CAS action: addCaslib [Required]

Default none

username="*user-name*"
specifies the database user name.

Typically, you specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply username= and password= values in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in CASLIB statement [see Requirement]

CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Aliases uid=

user=

Default none

Requirement If it is required by your database, then you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

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 8.4 *Data Type Conversions When Loading Data from ODBC into CAS*

ODBC Data Type	CAS Data Type
SQL_CHAR	CHAR
SQL_VARCHAR	VARCHAR
SQL_LONGVARCHAR	VARCHAR
SQL_WCHAR	CHAR
SQL_WVARCHAR	VARCHAR
SQL_WLONGVARCHAR	VARCHAR
SQL_BINARY	CHAR
SQL_VARBINARY	VARCHAR
SQL_LONGVARBINARY	VARCHAR
SQL_NUMERIC	DOUBLE
SQL_DECIMAL	DOUBLE
SQL_INTEGER	DOUBLE
SQL_SMALLINT	DOUBLE
SQL_FLOAT	DOUBLE
SQL_REAL	DOUBLE
SQL_DOUBLE	DOUBLE
SQL_BIGINT	DOUBLE
SQL_TINYINT	DOUBLE

ODBC Data Type	CAS Data Type
SQL_BIT	DOUBLE
SQL_TYPE_DATE	DOUBLE (formatted as DATE)
SQL_TYPE_TIME	DOUBLE (formatted as TIME)
SQL_TYPE_TIMESTAMP	DOUBLE (formatted as DATETIME)

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 through 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 39.

```
caslib odbccaslib desc="SQLviaODBCtoCaslib"
     datasource=(srctype="odbc"
                      username="user1"
                      password="password1"
                      odbc_dsn="dbodbc"
                      catalog="*") global;
```

Example 2: PROC CASUTIL: Load Data from an External Database Into SAS Cloud Analytic Services

```
proc casutil;
  list files incaslib="odbccaslib";       /* 1 */
  load casdata="myDBdata" incaslib="odbccaslib" outcaslib="casuser"
       casout="class_from_odbccaslib";    /* 2 */
  list tables incaslib="casuser";         /* 3 */
  contents casdata="class_from_odbccaslib" incaslib="casuser";
                                          /* 4 */
quit;
```

1 List the files in odbccaslib before loading your data.

2 Load the table myDBdata from an external database into memory in caslib Casuser. Call the new table class_from_oraclecaslib.

3 List the tables in casuser to see the newly created table, class_from_odbccaslib, that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

Example 3: Lua: Load Data from an External Database Into SAS Cloud Analytic Services

The following Lua code loads data from an ODBC database into CAS. Assume that the ODBC data source name is myODBC, the user name is myUserID, the password is myPwd, and the table name is myODBCtab.

```
r = s.loadDataSource{name="odbc"}                    1

r = s:addCaslib{lib="odbclib",                       2
                datasource={srcType="odbc",
                            odbc_dsn="myODBC",
                            username="myUserID",
                            password="myPwd"}
                }

print("Calling fileInfo")
r = s:fileInfo{caslib="odbclib"}                     3
print(r)

print("Calling columnInfo")                          4
r = s:columnInfo{table={caslib="odbclib", name="myODBCtab"}}
print(r)

print("Calling loadTable")
r = s:loadTable{caslib="odbclib", path="myODBCtab"} 5
print(r)
```

1 Call the loadDataSource action to load the data source and specify the data source type.

2 Define the connection parameters for a caslib, Odbclib.

3 List the tables in the caslib Odbclib.

4 List the column information in table myODBCtab.

5 Load the data from table myODBCtab into memory in the caslib Odbclib.

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 parameters are used in the context of different statements and CAS actions that connect your data in Oracle with CAS.

Valid in: CASLIB statement
CAS table actions
PROC CASUTIL statements (see parameters for details)

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

```
caslib oraclecaslib desc='Oracle Caslib'
     datasource=(srctype='oracle'
                 username='user1'
                 password='******'
                 path="//machine.lnx.com:1521/exadat");
```

Overriding the user and password values.

```
proc casutil;
   load casdata="%upcase(mycas.orexamp)" dataSourceOptions=(
         username='user5'
         password='******');
```

Syntax

Data Connector Parameters for Oracle

For each parameter described, the applicable statements and CAS actions where you can use that parameter are indicated. For information about where to specify these parameters within statements and action calls, see "Where to Specify Data Connector Parameters" on page 117.

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 or in the addCaslib action.

Valid in CASLIB statement

CAS actions: addCaslib, loadTable

PROC CASUTIL: LOAD statement

Default 1.0

Range 1.0–5.0

dbmsWhere="*WHERE-clause*"

specifies a database-specific SQL WHERE clause to submit to the database. The WHERE clause that you specify is passed to the database exactly as you enter it. This option is used to filter the rows that are read into a CAS table.

Valid in CAS actions: loadTable

PROC CASUTIL: LOAD statement

Default "" (zero-length string)

Example

```
action loadTable / caslib="tdlib", path="myTDtab",
  dataSourceOptions={dbmsWhere="make='Chevrolet' and
  MSRP < 18000"};
```

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

CAS actions: addCaslib, loadTable

PROC CASUTIL: LOAD statement

Default UNICODE

password="*password*"
specifies the DBMS password for a user. You typically specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply a username= and password= value in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in CASLIB statement [see Requirement]

CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Aliases pass=

pwd=

Default "" (zero-length string)

Requirement If your database requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

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

CAS actions: addCaslib, fileInfo, loadTable [Required]

PROC CASUTIL: CONTENTS, LIST, LOAD [Required] statements

Default none

Example

```
caslib oraclecaslib desc='Oracle Caslib'
                    datasource=(srcType='oracle'
                                username='user1'
                                password='myPwd1'
                                path="//machine.lnx.com:1521/exadat");
```

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 or for an action other than the addCaslib action, then this value overrides any value of schema= that was set when the caslib was defined.

Valid in	CASLIB statement
	CAS actions: addCaslib, fileInfo, columnInfo, loadTable
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Default	"" (zero-length string)

srcType="*ORACLE*"
specifies that the data source is an Oracle database.

Valid in	CASLIB statement [Required]
	CAS action: addCaslib [Required]
Default	none

username="*user-name*"
specifies the database user name.

Typically, you specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply username= and password= values in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in	CASLIB statement [see Requirement]
	CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Aliases	uid=
	user=
Default	none
Requirement	If it is required by your database, then you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

Details

Restrictions on Manipulating Oracle Data

When you load data from Oracle into SAS Cloud Analytic Services, the following 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 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 Oracle database.

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, the Oracle data is stored in the location that is designated by the path= argument.

```
caslib oraclelib
        datasource=(srctype='oracle',
                    username='user1',
                    password='myPwd',
                    path="myORAdata"
                    );
```

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

```
proc casutil;
   list tables incaslib="casuser";                    1
   load casdata="MYORADATA" incaslib="oraclecaslib" outcaslib="casuser"
        casout="ORAdata_from_oraclecaslib";           2
   list tables incaslib="casuser";                    3
   contents casdata="%upcase(class_from_oraclecaslib)" incaslib="casuser";
                                                      4
quit;
```

1 List the tables in casuser before loading your data.

2 Load the table myORAdata from Oracle into caslib Casuser. Call the new table ORAdata_from_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.

Example 3: Load Oracle Data into SAS Cloud Analytic Services Using Lua Code

This example shows how to connect to an Oracle database. Assume the user ID of user1 with a password of myPwd1. The data is located in path **//machine.lnx.com:1521/exadat**.

```
r = s:addCaslib{lib="orlib"                                    1
                    datasource={srtType="oracle",
                                username="user1",
                                password="myPwd1",
                                path="//machine.lnx.com:1521/exadat"}
                    };

r = s:fileInfo{caslib="orlib"}                                 2
print(r)

r = s:columnInfo{table={caslib="orlib",name="ORACARS"}}        3
print(r.columnInfo)

r = s:loadTable{caslib="orlib", path="ORACARS"}                4
print(r)
```

1 Add the caslib, Orlib.

2 List the tables in your Oracle database.

3 List the column information about table ORACARS.

4 Load the table ORACARS into memory in the caslib Orlib.

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.

```
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" | "stata" | "xls"
specifies the file type. Specify fileType=”excel” to work with XLSX files (.xlsx filename suffix).

Default xls

Restriction Use XLSX for national language characters. These do not display correctly for XLS files.

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. When used with the fileInfo action, you can use wildcards to match a specific set of file names. For more information, see "Using Wildcard Characters for Filename Matching" on page 71.

Valid in CAS actions: fileInfo [required], loadTable [required]

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. The getnames="false" option indicates 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 PosgreSQL enables you to load data from PostgreSQL into SAS Cloud Analytic Services. Data connector parameters are used in the context of different statements and CAS actions that connect your data in PostgreSQL with CAS.

Valid in: CASLIB statement
CAS table actions
PROC CASUTIL statements (see parameters for details)

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

```
caslib postgrescaslib desc='PostgreSQL Caslib'
     dataSource=(srctype='postgres'
                 server='PGserver'
                 username='user1'
                 password='myPwd'
                 database="PGdatabase-name");
```

Overriding the user and password values.

```
proc casutil;
   load casdata="mycas.pgexamp" casout="myPGdata" casuser dataSourceOptions=(
         username='user5'
         password='myPwd');
quit;
```

Syntax

Data Connector Parameters for PostgreSQL

For each parameter described, the applicable statements and CAS actions where you can use that parameter are indicated. For information about where to specify these parameters within statements and action calls, see "Where to Specify Data Connector Parameters" on page 117.

catalog="*catalog-name*"
specifies the name of the catalog to use for a connection.

Valid in CASLIB statement
CAS action: addCaslib

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 or in the addCaslib action.

Valid in CASLIB statement

CAS actions: addCaslib, loadTable

PROC CASUTIL: LOAD statement

Default 1.0

Range 1.0–5.0

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

CAS actions: addCaslib, loadTable

PROC CASUTIL: LOAD statement

conopts="*connection-options*"
specifies DBMS-specific connection options.

Valid in CASLIB statement

CAS actions: addCaslib, columnInfo, fileInfo, and loadTable

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]

CAS action: addCaslib [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-specific SQL WHERE clause to submit to the database. The WHERE clause that you specify is passed to the database exactly as you enter it. This option is used to filter the rows that are read into a CAS table.

Valid in CAS actions: loadTable

PROC CASUTIL: LOAD statement

Default "" (zero-length string)

Example
```
action loadTable / caslib="tdlib", path="myTDtab",
  dataSourceOptions={dbmsWhere="make='Chevrolet' and
```

```
MSRP < 18000"};
```

password="*password*"
specifies the DBMS password for a user. You typically specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply a username= and password= value in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in	CASLIB statement [see Requirement]
	CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Aliases	pass=
	pwd=
Default	"" (zero-length string)
Requirement	If your database requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

ptg_dsn="*PostgreSQL-datasource-name*"
specifies the PostgreSQL data source name. This option enables you to use a configured PostgreSQL data source in an odbc.ini file.

Valid in	CASLIB statement
	CAS action: addCaslib
Default	"" (empty string)
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.

readBuff="*number-of-rows*"
specifies the number of rows to fetch per block of data.

Valid in	CASLIB statement
	CAS action: addCaslib, loadTable
	PROC CASUTIL: LOAD statement
Default	calculated automatically based on row size

schema="*schema-name*"
specifies the schema name to use for the connection to the database.

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 or for an action other than the addCaslib action, then this value overrides any value of schema= that was set when the caslib was defined.

Valid in	CASLIB statement
	CAS actions: addCaslib, fileInfo, columnInfo, loadTable
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Default	"" (zero-length string)
Interaction	If you supply 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]
	CAS action: addCaslib [Required]

srcType="postgres"
specifies that the data source is a PostgreSQL database.

Valid in	CASLIB statement [Required]
	CAS action: addCaslib [Required]
Default	none

username="*user-name*"
specifies the database user name.

Typically, you specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply username= and password= values in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in	CASLIB statement [see Requirement]
	CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Aliases	uid=
	user=
Default	none

Requirement If it is required by your database, then you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

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 8.5 *Supported PostgreSQL Data Types*

PostgreSQL Data Type	CAS Data Type
BIT(*n*)	VARCHAR
CHARACTER(*n*)	VARCHAR
TEXT	VARCHAR
INTEGER	DOUBLE
BIGINT	DOUBLE
SMALLINT	DOUBLE
DOUBLE PRECISION	DOUBLE
DECIMAL(*p*, *s*)	DOUBLE
NUMERIC	DOUBLE
REAL	DOUBLE
SERIAL	DOUBLE
BIGSERIAL	DOUBLE
DATE	DOUBLE
TIME(*p*) (without time zone)	DOUBLE

PostgreSQL Data Type	CAS Data Type
TIMESTAMP(*p*) (without time zone)	DOUBLE
JSON, JSONB	VARCHAR

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= parameters.

```
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

```
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.

Example 3: Add a Caslib with the addCaslib Action by Using PROC CAS

```
/* Using PROC CAS */
proc cas;
```

```
  session mysess;
  action addCaslib / caslib="pglib"
                     session=false
                     dataSource={srctype="postgres",
                                 username="<PostgreSQL-username>",
                                 password="<PostgreSQL-password>",
                                 server="<PostgreSQL-server>",
                                 database="<PostgreSQL-database>"};
run;
quit;
```

Example 4: Add a Caslib with the addCaslib Action by Using Lua

```
/* Using Lua Code */
s:addCaslib{lib="pglib"
            dataSource={srctype="postgres",
                        username="<PostgreSQL-username>",
                        password="<PostgreSQL-password>",
                        server="<PostgreSQL-server>",
                        database="<PostgreSQL-database>"}}
```

SAS Data Connector to SAS Data Sets

Specifies the settings to use when loading data from SAS tables (data sets) into SAS Cloud Analytic Services.

Valid in: CASLIB statement

Applies to: CASUTIL procedure

See: Chapter 9, “Data Types,”

Example: Load a password-protected SAS table into SAS Cloud Analytic Services.

```
proc casutil;
   load casdata="salary.sas7bdat" casout="salary"
      importoptions=(filetype="basesas" password="secret");
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.

Default 1.0

Range 1.0–5.0

dataTransferMode="auto | parallel | serial"

specifies how SAS Cloud Analytic Services reads the SAS table when the server is a distributed server.

auto specifies that the worker nodes should access and read data from the SAS table.

parallel specifies that the worker nodes should access and read data from the SAS table.

serial specifies that the SAS Cloud Analytic Services controller node should access and read the SAS table.

Aliases dataTransfer=

dtm

Default auto

Restriction Data is loaded serially if only the controller has access to the table.

Requirements Worker nodes can load data serially only if the table is on a shared file system to which all worker nodes have access.

Worker nodes can load data in parallel only if they can all access the table. Otherwise, the SAS Cloud Analytic Services controller node loads the data serially.

Tip Specify parallel if you know that all nodes have access to the table. Specify serial if you know that only the controller has access to the table. If you aren't sure what type of access the nodes have to the table, specify auto or do not specify a value for this option. In this case auto is used if all nodes can access the table. Otherwise, serial is used.

encryptionPassword="*string*"
specifies a passphrase to use for a table that is encrypted using the Advanced Encryption Standard (AES).

Restriction You can specify this option only with the datasource option.

Requirement This option is required only if the table uses AES encryption.

fileType="basesas"
specifies the file type.

Default This is based on the file extention that is specified in the loadTable path option.

Requirement This option is required for sas7bdat files only if you do not include the sas7bdat extension in the file name.

password="*string*"
specifies the password for a password-protected table.

Aliases pass=

pwd=

Default none

Requirement Use this option only if the table is password-protected or uses SAS proprietary encryption.

path="*table-path-and-name*"
specifies the name of a SAS source table. When used 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 71.

Valid in CAS actions: addCaslib, loadTable

CAS action:

Default none

Details

Access SAS Tables

To access SAS tables requires a datasource option on the addCaslib action with **srctype="*path*"** as the specified value. Except for engine-based SAS tables that a metadata-bound library controls, you should be able to use the SAS Data Connector to access any SAS tables.

For large tables, if SAS Cloud Analytic Services can access the .sas7bdat file, you can use the CASUTIL procedure with the LOAD CASDATA= syntax. In this case, you can use the fileType option.

If you can access the SAS table 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, look for sashelp.iris in load a client-side file.

Example: Access a SAS Table Using the CASLIB Statement and the CAS Procedure

```
/* Find an engine-based SAS table. */
libname abc '/tstgen/wky/tst-v9cas/supio/testsio/lax';
proc contents data=abc.customer;run;

/* Start the SAS Cloud Analytic Services server. */
proc casoperate start=(term=yes)
   host="cashost001"
   install="/opt/v9cas/laxnd/TKGrid"
   port=0;
   quit;
run;

/* Create a SAS Cloud Analytic Services session. */
cas mysess user=&SYSUSERID

proc cas;
/* Define a PATH type CASLIB called "foodoo"   */
/* and point it to where the table is located. */
addcaslib /
   caslib="foodoo"
   datasource={srctype="path"}
   path="/tstgen/wky/tst-v9cas/supio/tests001/linus";
```

```
run;

/* call the loadTable action to load the Base data set */
/* "customer" into a CAS table called "xyz"            */
loadtable /
    caslib="foodoo"
    path="customer"
    importoptions={filetype="basesas", dtm="parallel"}
    casout={name="xyz" replace=true};
run;

/* Use the SAS Cloud Analytic Services engine to print out */
/* the newly created XYZ SAS Cloud Analytic Services.      */
libname mycas2 cas sessref=mysess;

proc print data=mycas2.xyz;run;
```

SAS Data Connector to Teradata and SAS Data Connect Accelerator for Teradata

SAS Data Connector to Teradata enables you to load data serially from Teradata into SAS Cloud Analytic Services. All users can use SAS Data Connector to Teradata. SAS Data Connect Accelerator for Teradata is a separately licensed product that enables you to load data in parallel using the SAS Embedded Process. Data connector parameters are used in the context of different statements and CAS actions that connect your data in Teradata with CAS.

Valid in: CASLIB statement
CAS table actions
PROC CASUTIL statements (see parameters for details)

Examples: Load a Teradata data source and add a casref to it.

```
caslib tdlib
     datasource=(srctype='teradata'
                 dataTransferMode='parallel'
                 server='TDserver'
                 username='user1'
                 password='myPwd'
                 database='TDdatabase');
```

Load Teradata source data using PROC CASUTIL.

```
proc casutil;
   load incaslib='tdlib' casdata='cars'
        casout='cars_CAS'
        options=(dbmsWhere='cylinders=8',
                 dataTransferMode='parallel');
run;
quit;
```

Syntax

Data Connector Parameters for Teradata

For each parameter described, the applicable statements and CAS actions where you can use that parameter are indicated. For information about where to specify these parameters within statements and action calls, see "Where to Specify Data Connector Parameters" on page 117.

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 was set in the CASLIB statement or with the addCaslib action.

Valid in	CASLIB statement
	CAS actions: addCaslib, loadTable
	PROC CASUTIL: LOAD statement
Default	1048576
Restriction	This parameter applies only with SAS Data Connect Accelerator for Teradata (dataTransferMode="parallel").

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 fileInfo action or from the CONTENTS statement in PROC CASUTIL.

Valid in	CASLIB statement
	CAS action: addCaslib
Default	active caslib
Restriction	This option applies only with SAS Data Connector to Teradata (dataTransferMode="serial").

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 or in the addCaslib action.

Valid in	CASLIB statement
	CAS actions: addCaslib, loadTable
	PROC CASUTIL: LOAD statement
Default	1.0
Range	1.0–5.0

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
	CAS actions: addCaslib, loadTable
	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
	CAS actions: addCaslib, loadTable
	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 or in the addCaslib action. Here are the valid values.

auto	specifies to first try to load the data in parallel (using embedded processing). If this fails, an error is issued and serial processing is then attempted.
parallel	specifies to always use only parallel (using embedded processing).
serial	specifies to load the data serially by using the SAS Data Connector to your database.

Valid in	CASLIB statement
	CAS actions: addCaslib, loadTable
	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.

dbmsWhere="*WHERE-clause*"
specifies a database-specific SQL WHERE clause to submit to the database. The WHERE clause that you specify is passed to the database exactly as you enter it. This option is used to filter the rows that are read into a CAS table.

Valid in CAS actions: loadTable

PROC CASUTIL: LOAD statement

Default "" (zero-length string)

Example
```
action loadTable / caslib="tdlib", path="myTDtab",
  dataSourceOptions={dbmsWhere="make='Chevrolet' and
  MSRP < 18000"};
```

name="*source-name*"
specifies the type of data source.

Valid in CAS action: loadDataSource [Required]

Examples
```
s:loadDatasource{name='teradata'}
```

```
proc cas;
   session mysess;
   action loadDatasource / name='teradata';
run;
```

password="*password*"
specifies the DBMS password for a user. You typically specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply a username= and password= value in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in CASLIB statement [see Requirement]

CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Aliases pass=

pwd=

Default "" (zero-length string)

Requirement If your database requires authentication, you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

readBuff="*integer*"
specifies the number of rows to fetch per block of data.

Valid in CASLIB statement

CAS actions: addCaslib, loadTable

PROC CASUTIL: LOAD statement

Aliases RAS=

row_array_size=

Default calculated automatically based on row size

Restriction This option is not valid with the SAS Data Connect Accelerator (dataTransferMode="parallel"). Specify bufferSize= when loading data in parallel.

role="*name*"
specifies the Teradata role name.

Valid in CASLIB statement

CAS actions: addCaslib

Default "" (empty string)

Restriction This option is not valid with the SAS Data Connect Accelerator (dataTransferMode="parallel").

schema="*schema-name*"
specifies the schema name to use for the connection to the database.

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 or for an action other than the addCaslib action, then this value overrides any value of schema= that was set when the caslib was defined.

Valid in CASLIB statement

CAS actions: addCaslib, fileInfo, columnInfo, loadTable

PROC CASUTIL: CONTENTS, LIST, LOAD statements

Default "" (zero-length string)

Interaction If you supply values for schema= and database=, then the value for schema= is used to qualify the table name.

server="*Teradata-server-identifier*"
specifies the server identifier for the Teradata server.

Valid in CASLIB statement [Required]

CAS actions: addCaslib [Required]

srcType="*teradata*"
specifies that the data source is a Teradata database.

Valid in CASLIB statement [Required]

CAS actions: addCaslib [Required]

Default	none

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 parameter.

This value overrides a value of statusInterval= that was set in the CASLIB statement or in the addCaslib action.

Valid in	CASLIB statement
	CAS actions: addCaslib, loadTable
	PROC CASUTIL: LOAD statement
Default	0 (no message)

username="*user-name*"
specifies the database user name.

Typically, you specify username= and password= values when you define a caslib. These credentials are then used for any statement or action that accesses the data using that caslib. If you supply username= and password= values in a statement other than the CASLIB statement or for an action other than the addCaslib action, then these credentials override any credentials that were specified when a caslib was defined.

Valid in	CASLIB statement [see Requirement]
	CAS actions: addCaslib [see Requirement], columnInfo, fileInfo, loadTable
	PROC CASUTIL: CONTENTS, LIST, LOAD statements
Aliases	uid=
	user=
Default	none
Requirement	If it is required by your database, then you must specify valid credentials to access data. You can provide these credentials by specifying username= and password= values.

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 or the addCaslib action and specify username= and password= values. These credentials are then used for any statement or action that accesses the data using that caslib.

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 when you add a caslib, and then you supply a password= value when you call an action, like loadTable, that accesses the data. In this situation, you must supply the password= value for each action or 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 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 8.6 *Supported Teradata Data Types*

Teradata Data Type	CAS Data Type	Default Format in CAS
CHAR	CHAR	$CHAR*w.*
VARCHAR	VARCHAR	$CHAR*w.*
DOUBLE	DOUBLE	*w.d*
DATE (DA)	DOUBLE	DATE*w.*
TIME (AT)	DOUBLE	TIME*w.d*
TIMESTAMP (TS)	DOUBLE	DATETIME*w.d*
INTEGER	DOUBLE	*w.d*
BYTEINT (I1)	DOUBLE	*w.d*
SMALLINT (I2)	DOUBLE	*w.d*
BIGINT (I8)	DOUBLE	*w.d*

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 an action or 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 call an action that accesses the data.

```
caslib TDlib desc='Teradata Caslib'
     dataSource=(srctype='Teradata',
                 dataTransferMode='serial',
                 server='teradataServer',
                 username='user1',
                 password='******',
                 );
```

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

```
proc casutil;
   list files incaslib="Teradatacaslib";                    1
   load casdata="myTDdata" incaslib="Teradatacaslib" outcaslib="casuser"
        casout="TDdata_from_Teradatacaslib";                2
   list tables incaslib="casuser";                          3
   contents casdata="TDdata_from_Teradatacaslib" incaslib="casuser";
                                                            4
quit;
```

1 List the tables in Teradatacaslib before loading your data.

2 Load the table myTDdata from Teradata into caslib Casuser. Call the new table TDdata_from_Teradatacaslib.

3 List the tables in caslib to see the newly created table, TDdata_from_Teradatacaslib, that you loaded.

4 List information about the newly loaded table, including column names, data types, and so on.

Example 3: Using Lua to Connect to Teradata, Examine the Data, and Load a Table

```
-- Load the Teradata data source
r=s:loadDatasource{name='teradata'}

-- Add the Caslib for Teradata
r = s:addCaslib{lib='tdlib',
             datasource={srcType='teradata',
                         dataTransferMode='parallel',
                         server='<Teradata-server-name>',
                         database='<Teradata-database-name>',
                         username='user1',
                         password='******',
                         schema='<Teradata-schema-name>'
```

```
                                }
                        }

print("Calling fileInfo")
r = s:fileInfo{caslib='tdlib'}
print(r)

print("Calling columnInfo")
r = s:columnInfo{table={caslib='tdlib', name='<Teradata-table-name>'}}
print(r)

print("Calling loadTable")
r = s:loadTable{caslib='tdlib', path='<Teradata-table-name>'}
print(r)
```

Example 4: Add a Caslib with the addCaslib Action

```
/* Using PROC CAS */
proc cas;
  session mysess;
  action addCaslib / caslib="tdlib"
                     session=false
                     dataSource={srctype="teradata",
                                 username="<Teradata-username>",
                                 password="<Teradata-password>",
                                 server="<Teradata-server>",
                                 database="<Teradata-database>"};
run;
quit;
```

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.

VARCHAR Benefits and Considerations

In most cases you can take advantage of the benefits of using VARCHAR instead of CHAR. Here are some examples.

- the lengths of the character data vary significantly.
- the longest strings are infrequent and would require a fixed length of 64 bytes.

In other cases, however, it is better to use a fixed-width column when data is consistently short—namely, less than 16 bytes, such as an ID column of airport codes—because it uses less memory and runs faster.

In addition, keep these considerations in mind for variables with an undefined maximum length:

- VARCHAR(*) indicates that no maximum length on the column is being defined.
- Using VARCHAR(*) can be helpful if the maximum length of data for a column is not known when the column is being defined.

There is another consideration to keep in mind when you use VARCHAR(*). If you copy a table that is defined with a VARCHAR(*) to an engine library that does not support VARCHAR, a CHAR data type is created instead and is defined with the maximum length of 32767 bytes. If you instead provide an explicit length, such as VARCHAR(10), a CHAR column is created in the new table with a byte length of 40. A maximum length of 40 bytes is required to hold 10 characters in a 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 9.1 *Data Types for SAS Cloud Analytic Services Table Columns*

Data Type Definition Keyword	SAS Cloud Analytic Services Table Column Data Type	Data Type Returned	Missing Values
CHAR(*n*)	CHAR(*n*)	CHAR(*n*)	all blanks (the same as in SAS)
VARCHAR(*n*)	VARCHAR(*n*)	all blanks or a zero length	
DOUBLE	DOUBLE	DOUBLE	

Chapter 10

Functions

Dictionary

GETCASURL Function

Returns the value for a URL for connecting to the CAS Server Monitor.

Requirement:

- The server name identified by the SAS CASHOST= option is used when constructing the URL.
- The value provided is a valid session name for the SAS CASHOST= option or, if the value is not provided, the SAS SESSREF= option is a valid session on the server. This connection obtains additional URL information.

Syntax

GETCASURL(<session>)

Optional Argument

session
if 0 parameters are specified, then the SAS SESSREF= option value is used.

Example: GETCASURL Example

```
%put httpaddr= %sysfunc(getcasurl());
httpaddr=http://host and port value
```

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 177 .

TIP You can list the session option names with this code:

```
CAS mysess LISTSESSOPTS;
```

Example: Listing the Active CASLIB

This example returns the CASLIB option value.

Statements	Results

`%put caslib = %sysfunc(GETSESSOPT(mysess, caslib)) ;`
 	
`data work.one ;`
`  x = GETSESSOPT("mysess", "caslib") ;`
`  put x= ;`
`run;`
	`x=CASUSERHDFS(userid)`

SESSFOUND Function

Returns a 0 when a CAS session is not connected to a server and a 1 when the session is connected to a server.

Syntax

SESSFOUND (*session-name*)

Required Argument

session-name
returns a value to indicate the status of the session.

0
The session is not found.

1
The session is found.

Details

This function has access to sessions that you started in your SAS session only. You can use the CAS statement with the LISTSESSIONS option to identify all your CAS sessions on a server.

Example: Determining CAS Sessions

This example shows whether the CAS session is found.

Statements	Results
`%put doIExist= %sysfunc(sessfound(mysess));`	`doIExist= 0`
`%put doIExist= %sysfunc(sessfound(existingSession));` `run;`	`doIExist= 1`

Chapter 11

Macro Variables

Dictionary

CASHOST Macro Variable

Specifies the name of the SAS Cloud Analytic Services server.

Default: Not defined

Range: 256 characters

Interactions: When set, this macro variable overrides SAS system option CASHOST.
The CAS statement HOST= option overrides this macro variable.

Tip: If you want to delete macro variable _CASHOST_ and allow SAS system option CASHOST to prevail, use the following statement:

```
%symdel _CASHOST_;
```

Syntax

%let _CASHOST_=cloud.example.com

CASPORT Macro Variable

Specifies the SAS Cloud Analytic Services 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 CASNAME (alias 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 9. To see the setting for a specific property, use the GETSESSOPT function. See “GETSESSOPT Function” on page 171.

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 server 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 SAS Cloud Analytic Services Server Monitor. For each option, this page shows the current value and the source of the value. The values for the session options are used as defaults for the session options, unless they are overridden. For information about the Server Monitor, see *SAS Viya Administration: Using CAS Server Monitor*.

Overriding the Default Session Option Values

To override the default session option values for a new session, use the option shown in the following table to complete the desired task.

Task	Option to use:
Override one or more session options for a new session only.	CAS statement SESSOPTS= option
Override the TIMEOUT option value for all subsequently created sessions.	SAS system option CASTIMEOUT=****
Override the NWORKERS option value for all subsequently created sessions.	SAS system option CASNWORKERS=*****

* TIMEOUT in the CAS statement SESSOPTS= option overrides this option.

** NWORKERS in the CAS statement SESSOPTS= option overrides this option.

*** Setting this option does not affect existing sessions.

Setting Options for Existing Sessions

For an existing session, use the option shown in the following table to complete the desired task.

Task	Option to use:
Change one or more session options for a specific session.	CAS statement SESSOPTS option
Change one or more session options for the active session.*	SAS system option CASSESSOPTS= (alias SESSOPTS=)
Specify a caslib for the active session.*	SAS system option CASLIB=

* SAS system option CASNAME (alias SESSREF) stores the name of the currently active session.

Session Options by Category

Category	Language Elements	Description
Action	APPTAG= Session Option (p. 180)	specifies the string to prefix to log messages.
Caslib	CASLIB= Session Option (p. 180)	specifies the caslib name to set as the active caslib.
	MAXTABLEMEM= Session Option (p. 185)	specifies the maximum amount of physical memory, in bytes, to allocate for a table.
DATA step	DATASTEPMSGSUMLEVEL= Session Option (p. 183)	specifies the DATA step message summary level.
	DATASTEPREPLACETABLE= Session Option (p. 183)	specifies whether a DATA step can replace an existing table.
Formats	FMTCASLIB Session Option (p. 184)	specifies the caslib where persisted format libraries are retained.
Localization	LOCALE= Session Option (p. 184)	specifies the locale to use for sorting and formatting.
Log	LOGFLUSHTIME= Session Option (p. 184)	specifies the log flush time, in milliseconds.
	MESSAGELEVEL= Session Option (p. 185)	specifies the log message level.
	METRICS= Session Option (p. 185)	specifies whether to include default detailed performance metrics reports in the SAS log.

Category	Language Elements	Description
Session	NWORKERS= Session Option (p. 187)	specifies the number of worker nodes for a new session.
	TIMEOUT= Session Option (p. 187)	specifies the SAS Cloud Analytic Services session time-out in seconds for a new or existing session.
	TIMEZONE= Session Option (p. 188)	specifies the time zone offset, in hours, from UTC.
Sort	COLLATE= Session Option (p. 182)	specifies the collating sequence for sorting.
System Administration: Performance	CMPOPT= Session Option (p. 181)	specifies the type of code generation optimizations to use in the SAS language compiler.

Dictionary

APPTAG= Session Option

specifies the string to prefix to log messages.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Action

Default: No prefix

Syntax

APPTAG="*tag-string*"

CASLIB= Session Option

specifies the caslib name to set as the active caslib.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Caslib

See: “CASLIB Statement” on page 39

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

See: Option CMPOPT= in *SAS Viya System Options: Reference*

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:

```
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.

DATASTEPMSGSUMLEVEL= Session Option

specifies the DATA step message summary level.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: DATA step

Default: ALL

Tip: When the DATA step runs on multiple threads, the same message can be generated on each thread. In that case, use this option to control the summary level of the duplicate messages to help reduce the client log output.

Syntax

DATASTEPMSGSUMLEVEL=ALL | PUT | NONE

Parameter Values

In a DATA step, messages are received from each thread, which can result in a large number of duplicate messages when multiple threads are used. The first occurrence of all messages, including PUT statement messages, are sent to the client when they occur. By default, duplicate messages are summarized, and then sent to the client to reduce client log output. Specify one of the following values to control the level of summarization for duplicate messages:

ALL
summarizes all duplicate messages, including PUT statement messages, and sends them to the client when the DATA step exits. This is the default.

PUT
summarizes all duplicate messages, except PUT statement messages, and sends them to the client when the DATA step exits. All PUT statement messages are not summarized and are sent to the client as they occur.

NONE
does not summarize duplicate messages. All messages, including PUT statement messages, are sent to the client as they occur.

DATASTEPREPLACETABLE= Session Option

specifies whether a DATA step can replace an existing table.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: DATA step

Default: TRUE

Syntax

DATASTEPREPLACETABLE=TRUE | FALSE

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

Note: This option is set by the system administrator.

Default: FORMATS

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*"

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 physical 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*

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 12.1 *Default Metrics*

Metric	Note Written to the SAS Log	
The number of bytes moved.	`NOTE: bytes moved`	*number<units>*
The CPU time in seconds and as a percentage of cluster utilization. Cluster utilization is the sum of the utilization for each core in the cluster and can exceed 100%. For example, cluster utilization for a 96-core cluster where each core is 100% used is 9600%.	`NOTE: cpu time`	*number* `seconds (`*number*`%)`
The data movement time in seconds.	`NOTE: data movement time`	*number* `seconds`
The amount of memory used in bytes and as a percentage of the total available memory.	`NOTE: memory`	*number<units>* `(`*number*`%)`
The amount of time it took to run the action start to finish (real time) in seconds.	`NOTE: real time`	*number* `seconds`
The total available memory.	`NOTE: total memory`	*number<units>*
The total number of nodes and cores in the cluster.	`NOTE: total nodes`	*number* `(`*number* `cores)`

Here is an example of the default metrics that are written to the SAS log when the MDSUMMARY procedure is executed.

```
NOTE: real time               0.126269 seconds
NOTE: cpu time                1.036837 seconds (821.13%)
NOTE: data movement time      0.013819 seconds
NOTE: total nodes             27 (1296 cores)
NOTE: total memory            6.65T
NOTE: memory                  57.25M (0.00%)
NOTE: bytes moved             4.80K
NOTE: The SAS Cloud Analytic Server processed the request in 0.126269 seconds.
NOTE: The data set MYCAS.MPGHW_SUM has 15 observations and 19 variables.
NOTE: PROCEDURE MDSUMMARY used (Total process time):
      real time           0.38 seconds
      cpu time            0.01 seconds
```

NWORKERS= Session Option

specifies the number of worker nodes for a new session.

Valid in: CAS statement SESSOPTS option
OPTIONS statement CASSESSOPTS option
GETSESSOPT function

Category: Session

Default: In order of descending precedence:

1. SAS system option CASNWORKERS, if you explicitly set it in SAS
2. 0 (all)

Range: 0–5000

Restriction: The number of workers can be set for new sessions only.

See: “CASNWORKERS= System Option” on page 193

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 194

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*

Chapter 13

System Options

Dictionary

AUTHINFO= System Option

specifies a file where user ID and passwords are kept for authentication.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

PROC OPTIONS GROUP= CAS

Alias: CASAUTHINFO=

Interaction: SAS Studio user credentials are used to authenticate the connection to CAS. The Authinfo file credentials are not used. The most frequent use of this system option is to submit code to CAS from the command line, in batch mode.

Note: AUTHINFO is also an environment variable.

See: For more information about the Authinfo file, see "Create an Authinfo File" in *SAS Viya Administration: Authentication*.

Syntax

AUTHINFO='*authinfo_file_path*';

Session Options

authinfo_file_path

specifies the path where an Authinfo file is located. The Authinfo file provides an alternative to including passwords in programs. The most frequent use of this option is to submit code to CAS from the command line, in batch mode. For more information, see "Authinfo File" in *SAS Viya Administration: Authentication* and "Create an Authinfo File" in *SAS Viya Administration: Authentication*.

AUTHINFO can also be used as an environment variable. 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.

End users can store an encoded password in an Authinfo file. For more information, see "Encode the Authinfo Password Using PROC PWENCODE" in *SAS Viya Administration: Authentication*.

Examples Set AUTHINFO= SAS system option. This option overrides the Authinfo file pointed to by the AUTHINFO environment variable.

```
Options AUTHINFO='$HOME/authInfo-file';
```

AUTHINFO can also be set as an environment variable. This option overrides the .authinfo file.

```
Options insert=(set=AUTHINFO='$HOME/authInfo-file');
```

Details

There is an order of precedence to using AUTHINFO options. The following order applies.

1. The AUTHINFO environment variable overrides the .authinfo file.
2. The AUTHINFO= SAS system option overrides the AUTHINFO environment variable.
3. The CAS statement AUTHINFO option overrides the AUTHINFO SAS System option and the environment variable.

See Also

"CAS Statement" on page 1

CASSESSOPTS= System Option

Specifies one or more session options for the active CAS session.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

PROC OPTIONS GROUP= CAS

Alias: SESSOPTS=

Default: None

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS Viya 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 10.

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)
```

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 177 for a list of the options that you can specify for *session-option(s)*.

See Also

Functions:

- "GETSESSOPT Function" on page 171

Macro Statement and Functions:

- "%PUT Statement" in *SAS Viya Macro Language: Reference*
- "%SYSFUNC and %QSYSFUNC Functions" in *SAS Viya Macro Language: Reference*

Statements:

- "CAS Statement" on page 1

CASHOST= System Option

Specifies the CAS host name that is associated with a CAS session.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

PROC OPTIONS GROUP= CAS

Default: None

Range: 256 characters

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS Viya System Options: Reference*.

Example:
```
options cashost="cloud.example.com";
```

Syntax

CASHOST= "*host-name*"

CASLIB= System Option

Specifies the caslib name for the session that is identified by the SESSREF= option.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

PROC OPTIONS GROUP= CAS

Default: None

Range: 128 characters

Interaction: The CAS statement session option CASLIB= overrides this option. For more information, see "CASLIB= Session Option" on page 180.

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS Viya 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 10.

Example: Set the default caslib:

```
options caslib="casuser";
```

Syntax

CASLIB="*name*"

See Also

Statements:

- "CAS Statement" on page 1
- "CASLIB Statement" on page 39

CASNAME= System Option

Specifies the name to associate with a generated CAS session.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

PROC OPTIONS GROUP= CAS

Alias: SESSREF

Default: CASAUTO

Range: 256 characters

Interactions: When you create a session using the CAS statement, the value of the CASNAME= 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 CASNAME= option, the value of the _SESSREF_ macro variable is set to the same name.

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS Viya System Options: Reference*.

Example:
```
options casname=mysessref;
```

Syntax

CASNAME= *session-name*

SESSREF= *session-name*

See Also

Statements:

- "CAS Statement" on page 1

CASNWORKERS= System Option

Specifies the number of worker nodes to use for a CAS session.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

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 Viya 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 187.

Example:
```
options casnworkers=10;
```

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

CASPORT= System Option

Specifies the port to use when connecting to CAS.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

PROC OPTIONS GROUP= CAS

Default: 0

Range: 0-65535

Note: This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS Viya System Options: Reference*.

Example:
```
options casport=12345;
```

Syntax

CASPORT=*port-number*

Syntax Description

port-number
specifies the CAS server port number.

When *port-number* is set to 0, CAS selects a port number.

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: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

PROC OPTIONS GROUP= CAS

Default: 60

Range: 0–31536000

Interaction: The session option TIMEOUT= overrides this option. For more information, see "TIMEOUT= Session Option" on page 187.

Notes: 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.

This option is ignored when the value is set to 0. In that case, the default for the TIMEOUT= session option applies.

This option can be restricted by a site administrator. For more information, see "Restricted Options" in *SAS Viya System Options: Reference*.

Tip: To change the time-out for an existing session, use the TIMEOUT= session option.

Example:

```
options castimeout=28800;
```

Syntax

CASTIMEOUT= *n* | *n*K | *n*M | *hex*X | MAX | MIN

Syntax Description

***n* | *n*K | *n*M**
specifies the session time out in seconds that are processed in multiples of 1, 1,024 (K) or 1,048,576 (M). For example, a value of 432008 specifies 43,200 seconds, and a value of 43k specifies 44,032 seconds.

***hex*X**
specifies the session time out seconds as a hexadecimal value. You must specify the value beginning with a number (0–9), followed by an X. For example, the value 0a8c0x sets the number of seconds to 43200 seconds.

MAX
sets the time out value to 31536000.

MIN
sets the time out value to 0.

CASUSER= System Option

Specifies the user ID to use when connecting to CAS.

Valid in: Configuration file, SAS invocation, OPTIONS statement, SASV9_OPTIONS environment variable

PROC OPTIONS GROUP= CAS

Alias: CASUSERID=

Default: None

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 "Create an Authinfo File" in *SAS Viya Administration: Authentication*.

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 10.

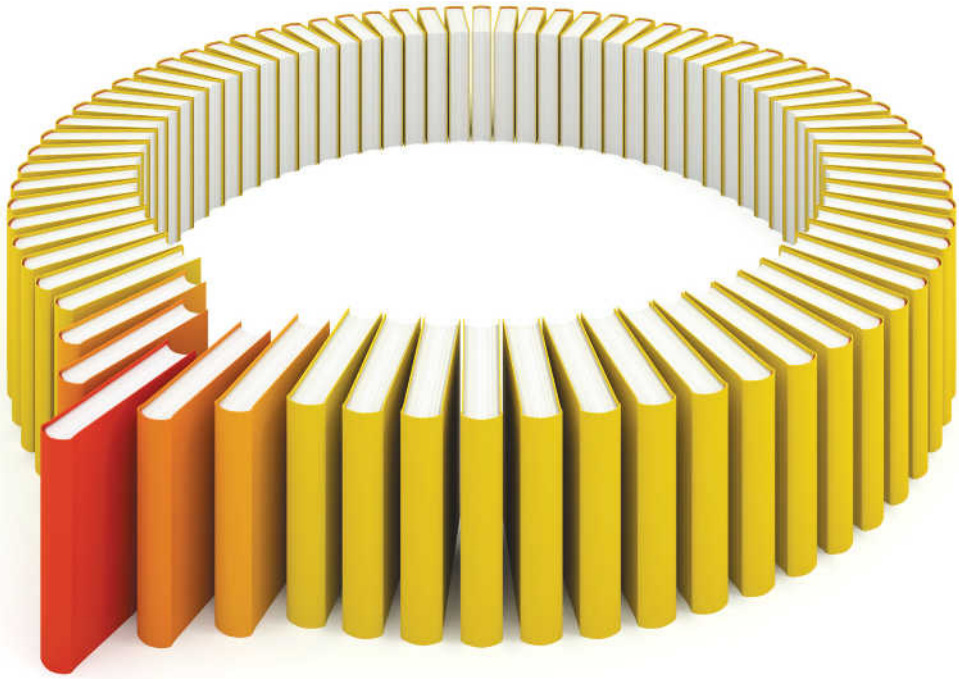
Note: This option cannot be restricted by a site administrator. For more information, see "Restricted Options" in *SAS Viya System Options: Reference*.

Example:
```
options casuser=myid;
```

Syntax

CASUSER= *user-ID*

CASUSERID= *user-ID*



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.