## Contents

**Chapter 1 • Accessing Data** ................................................................. 1
- Early Adopter Software ........................................................................ 2
- Introduction ......................................................................................... 2
- Terms to Be Familiar With .................................................................... 2
- Common Tasks for Accessing and Manipulating Data .......................... 3
- Load a Client-Side File ....................................................................... 4
- Load a Server-Side File ....................................................................... 5
- Load a Database Table ........................................................................ 12
- Save an In-Memory Table ................................................................... 13
- Drop an In-Memory Table ................................................................... 14
- Delete a File from a Caslib’s Data Source .......................................... 15
- Data Compression ............................................................................... 15

**Chapter 2 • DATA Step Feature Engineering** ..................................... 19
- Common Tasks for Running the DATA Step ........................................ 20
- DATA Step Basics .............................................................................. 21
- Run a Simple DATA Step in CAS ....................................................... 41
- Convert a CAS Table to a SAS Data Set ............................................ 43
- Save a SAS Data Set to CAS ............................................................. 44
- Create a VARCHAR Variable Using the LENGTH Statement ............ 45
- Check for Missing Values ................................................................. 46
- Index CHAR and VARCHAR Character Strings ............................... 47
- Keep Variables in a CAS DATA Step ................................................ 49
- Order BY Variables in CAS ............................................................. 50
- Format BY Variables ......................................................................... 52
- Order Rows in a Table Using ORDERBY= ......................................... 53
- Remove Rows from a Table Using the WHERE Statement ............... 54
- Merge Tables Using the SET Statement ............................................ 55
- Read a SAS 9 Data Set into SAS Viya .............................................. 56
- Control Thread Count Using SINGLE= ............................................ 58
- View DATA Step Processing Information Using Automatic Variables 59
- Use the Macro Facility to Generate CAS DATA Step Code ............... 60
- Score an In-Memory CAS Table ....................................................... 62

**Chapter 3 • Working with User-Defined Formats** ............................... 63
- Common Tasks for User-Defined Formats ........................................ 64
- User-Defined Format Basics ............................................................ 65
- Permanent Formats Supplied by SAS ............................................... 66
- How Are Format Libraries Different than Format Catalogs? .............. 66
- Add and Save a User-Defined Format ............................................... 66
- Add a Format Library from a File ...................................................... 69
- List Format Libraries, Formats, and Search Path .............................. 71
- Add Locale-Specific Format Libraries .............................................. 73
- Add Multiple Format Catalogs to a CAS Session .............................. 75
- Promote a Format Library ............................................................... 75
# Chapter 1
## Accessing Data

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Early Adopter Software</td>
<td>2</td>
</tr>
<tr>
<td>Introduction</td>
<td>2</td>
</tr>
<tr>
<td>Terms to Be Familiar With</td>
<td>2</td>
</tr>
<tr>
<td>Common Tasks for Accessing and Manipulating Data</td>
<td>3</td>
</tr>
<tr>
<td>Load a Client-Side File</td>
<td>4</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>4</td>
</tr>
<tr>
<td>Example</td>
<td>4</td>
</tr>
<tr>
<td>Key Ideas</td>
<td>5</td>
</tr>
<tr>
<td>See Also</td>
<td>5</td>
</tr>
<tr>
<td>Load a Server-Side File</td>
<td>5</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>5</td>
</tr>
<tr>
<td>Example</td>
<td>5</td>
</tr>
<tr>
<td>Results</td>
<td>8</td>
</tr>
<tr>
<td>Key Ideas</td>
<td>11</td>
</tr>
<tr>
<td>See Also</td>
<td>12</td>
</tr>
<tr>
<td>Load a Database Table</td>
<td>12</td>
</tr>
<tr>
<td>Prerequisites</td>
<td>12</td>
</tr>
<tr>
<td>Example</td>
<td>12</td>
</tr>
<tr>
<td>Key Ideas</td>
<td>13</td>
</tr>
<tr>
<td>Save an In-Memory Table</td>
<td>13</td>
</tr>
<tr>
<td>Example</td>
<td>13</td>
</tr>
<tr>
<td>Key Ideas</td>
<td>14</td>
</tr>
<tr>
<td>Drop an In-Memory Table</td>
<td>14</td>
</tr>
<tr>
<td>Example</td>
<td>14</td>
</tr>
<tr>
<td>Key Ideas</td>
<td>15</td>
</tr>
<tr>
<td>Delete a File from a Caslib’s Data Source</td>
<td>15</td>
</tr>
<tr>
<td>Example</td>
<td>15</td>
</tr>
<tr>
<td>Key Ideas</td>
<td>15</td>
</tr>
<tr>
<td>Data Compression</td>
<td>15</td>
</tr>
<tr>
<td>Overview of Data Compression</td>
<td>15</td>
</tr>
<tr>
<td>Compressed Tables and the DATA Step</td>
<td>16</td>
</tr>
<tr>
<td>Performance Considerations</td>
<td>17</td>
</tr>
<tr>
<td>Interactions</td>
<td>17</td>
</tr>
</tbody>
</table>
Early Adopter Software

THIS DOCUMENTATION FOR AN EARLY ADOPTER PRODUCT IS A PRELIMINARY DRAFT AND IS PROVIDED BY SAS INSTITUTE INC. ("SAS") ON AN "AS IS" BASIS WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, IMPLIED WARRANTIES OF MERCHANTABILITY AND/OR FITNESS FOR A PARTICULAR PURPOSE. SAS does not warrant that this documentation is complete, accurate, or similar to that which may be released to the general public, or that any such documentation will be released. The company shall not be liable whatsoever for any damages arising out of the use of this documentation, including any direct, indirect, or consequential damages. SAS reserves the right to alter or abandon use of this documentation at any time.

Introduction

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

- caslibs
- CASUTIL procedure
- DATA step

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

Terms to Be Familiar With

Before we begin, here are a few important terms:

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

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

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

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

session

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

table

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

Common Tasks for Accessing and Manipulating Data

Table 1.1  Common Tasks for Accessing Data

<table>
<thead>
<tr>
<th>Task</th>
<th>Sample Syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>Load a SAS data set.</td>
<td>proc casutil; load data=libref.member-name casout=&quot;table-name&quot;; run;</td>
</tr>
<tr>
<td>Tip: This is a good learning step if you</td>
<td></td>
</tr>
<tr>
<td>are familiar with SAS and want to learn</td>
<td></td>
</tr>
<tr>
<td>how SAS libraries, data sets, and SAS</td>
<td></td>
</tr>
<tr>
<td>Cloud Analytic Services work together.</td>
<td></td>
</tr>
<tr>
<td>Load a client-side data file.</td>
<td>proc casutil; load file=&quot;/path/to/file.suffix&quot; casout=&quot;table-name&quot;; run;</td>
</tr>
<tr>
<td>List caslibs. This shows you the server-side</td>
<td>caslib <em>all</em> list;</td>
</tr>
<tr>
<td>data sources that SAS Cloud Analytic</td>
<td></td>
</tr>
<tr>
<td>Services can access.</td>
<td></td>
</tr>
<tr>
<td>Add a file-based caslib.</td>
<td>caslib data datasource=(srctype=&quot;path&quot;) path=&quot;/data01&quot;;</td>
</tr>
<tr>
<td>Tip: Remember that the specified PATH=</td>
<td></td>
</tr>
<tr>
<td>must be accessible from the host for the</td>
<td></td>
</tr>
<tr>
<td>SAS Cloud Analytic Services controller.</td>
<td></td>
</tr>
<tr>
<td>Determine the data files in a caslib that</td>
<td>proc casutil; list files incaslib=&quot;name&quot;; run;</td>
</tr>
<tr>
<td>the server can access.</td>
<td></td>
</tr>
<tr>
<td>Load a server-side data file.</td>
<td>proc casutil; load casdata=&quot;file-name.suffix&quot; casout=&quot;table-name&quot;; run;</td>
</tr>
</tbody>
</table>
### Load a Client-Side File

#### Prerequisites

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

#### Example

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

```sas
   caslib hps datasource=(srctype="hdfs") path="/hps"; /* 1 */
   proc casutil incaslib="hps" outcaslib="hps"; /* 2 */
     load data=sashelp.iris promote; /* 3 */
     contents casdata="iris";
     load file="/data/WorldData.xlsx" casout="worlddata"; /* 4 */
     contents casdata="worlddata";
   run;
```

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

2. The CASUTIL procedure statement includes the INCASLIB= and OUTCASLIB= options. This is a best practice to ensure that tables are read from the caslib that you expect and are saved to the caslib that you expect.
The DATA= argument indicates that the table is transferred from the SAS client host to SAS Cloud Analytic Services. Replace the Sashelp.Iris value with a libref and table that you want to use. The PROMOTE option makes the Iris table a global-scope table and available to other sessions that use the Hps caslib. Use the CONTENTS statement to ensure that the table includes the column names and data types that you expect.

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

**Key Ideas**

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

**See Also**

- “CASUTIL” in *SAS Cloud Analytic Services: Language Reference*
- “CASLIB Statement” in *SAS Cloud Analytic Services: Language Reference*

**Load a Server-Side File**

**Prerequisites**

The following example assumes the following:

- You can create a small CSV file in the file system that is associated with the directory for your personal caslib.

**Example**

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

- a CSV file that describes the performance of a toy catapult. The first line of the file does not contain column names. The example shows how to specify names.
- a Microsoft Excel file.
For the CSV file, a description for the data is shown in the example. The values are as follows:

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

The program is as follows:

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

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

/* first, load the data from the CSV file */
proc casutil incaslib="casuser" outcaslib="casuser";
    contents casdata="catapult.csv";
    load casdata="catapult.csv" casout="catapultraw"
        importoptions=(filetype="csv"
            encoding="latin1"
            getnames="false"
            vars=(
                (name="turns", label="Number of turns", type="double"),
                (name="first_ft", label="Feet for first try", type="double"),
                (name="first_in", label="Inches for first try", type="double"),
                (name="second_ft", label="Feet for second try", type="double"),
                (name="second_in", label="Inches for second try", type="double"),
                (name="third_ft", label="Feet for third try", type="double"),
                (name="third_in", label="Inches for third try", type="double")
            )
        ) replace;
    save casdata="catapultraw" replace;
    contents casdata="catapultraw.sashdat";
quit;

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

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

/* 9 */
/* simple scatter plot */
proc sgplot data=mycas.catapult;
    scatter x=turns y=first;
    scatter x=turns y=second;
    scatter x=turns y=third;
```
The SESSOPTS= option is used with the CASLIB= session option to ensure that the Casuser personal caslib is set as the active caslib.

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

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

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

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

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

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

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

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

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

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

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

Figure 1.1 CONTENTS Statement Results for the CSV File

The CASUTIL Procedure

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

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

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

Figure 1.2 CONTENTS Statement Results for the SASHDAT File

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

<table>
<thead>
<tr>
<th>Column Information for catapultraw.sashdat in Caslib CASUSER( ).</th>
</tr>
</thead>
<tbody>
<tr>
<td>Column</td>
</tr>
<tr>
<td>--------</td>
</tr>
<tr>
<td>turns</td>
</tr>
<tr>
<td>first_ft</td>
</tr>
<tr>
<td>first_in</td>
</tr>
<tr>
<td>second_ft</td>
</tr>
<tr>
<td>second_in</td>
</tr>
<tr>
<td>third_ft</td>
</tr>
<tr>
<td>third_in</td>
</tr>
</tbody>
</table>
Figure 1.3  CONTENTS Statement Results for the In-Memory Table

<table>
<thead>
<tr>
<th>Table Name</th>
<th>Number of Rows</th>
<th>Number of Columns</th>
<th>NLS encoding</th>
<th>Created</th>
<th>Last Modified</th>
<th>Promoted Table</th>
<th>Dupliacted Rows</th>
<th>View</th>
<th>Source Name</th>
<th>Compressed</th>
</tr>
</thead>
<tbody>
<tr>
<td>CATAPULT</td>
<td>5</td>
<td>10</td>
<td>utf-8</td>
<td>01Feb2016:19:29:25</td>
<td>01Feb2016:19:27:56</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
<td>_T_Saad1500_7FF6E42BBA9</td>
<td>No</td>
</tr>
</tbody>
</table>

Detail Information for catapult in Caslib CASUSER:

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

Column Information for CATAPULT in Caslib CASUSER:

<table>
<thead>
<tr>
<th>Column</th>
<th>Label</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>turns</td>
<td>Number of turns</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>first_t</td>
<td>Fast for first try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>first_in</td>
<td>Inches for first try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>second_t</td>
<td>Fast for second try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>second_in</td>
<td>Inches for second try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>third_t</td>
<td>Fast for third try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>third_in</td>
<td>Inches for third try</td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>first</td>
<td></td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>second</td>
<td></td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
<tr>
<td>third</td>
<td></td>
<td>double</td>
<td>8</td>
<td>12</td>
</tr>
</tbody>
</table>

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

The CASUTIL Procedure

<table>
<thead>
<tr>
<th>Name</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>File Size</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>catapult.csv</td>
<td>-rw-r--r--</td>
<td></td>
<td></td>
<td>0.1KB</td>
<td>01Feb2016:14:50:35</td>
</tr>
<tr>
<td>historicalcpt.xls</td>
<td>-rwXr-xr-X</td>
<td></td>
<td></td>
<td>56.5KB</td>
<td>02Feb2016:19:08:19</td>
</tr>
<tr>
<td>mycas.distinct.shdat</td>
<td>-rwXr-Xr-X</td>
<td></td>
<td></td>
<td>7.2KB</td>
<td>12Nov2015:16:48:12</td>
</tr>
<tr>
<td>cars_large_part.shdat</td>
<td>-rwXr-Xr-X</td>
<td></td>
<td></td>
<td>20.4MB</td>
<td>26Jan2016:12:44:45</td>
</tr>
</tbody>
</table>
Figure 1.5  Scatter Plot Results for the Catapult Table
The following display shows the results of the CONTENTS statement for the Historicalcpi.xls file. By default, the column names are read from a file.

**Figure 1.6 CONTENTS Statement Results for the XLS File**

### The CASUTIL Procedure

#### File Information for historicalcpi.xls in caslib CASUSER(

<table>
<thead>
<tr>
<th>Name</th>
<th>Permission</th>
<th>Owner</th>
<th>Group</th>
<th>File Size</th>
<th>Last Modified</th>
</tr>
</thead>
<tbody>
<tr>
<td>historicalcpi.xls</td>
<td>rw-xr-xr-x</td>
<td></td>
<td></td>
<td>56.5KB</td>
<td>02Feb2016:19:08:19</td>
</tr>
</tbody>
</table>

#### Column Information for historicalcpi.xls in Caslib CASUSER(

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Consumer price indexes historical data, 1974 through 2014</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>B</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>C</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>D</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>E</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>F</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AO</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AP</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AQ</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AR</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AS</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
<tr>
<td>AT</td>
<td>varchar</td>
<td>0</td>
<td>0</td>
<td>$</td>
</tr>
</tbody>
</table>

**Key Ideas**

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

- “CASUTIL” in *SAS Cloud Analytic Services: Language Reference*
- “CASLIB Statement” in *SAS Cloud Analytic Services: Language Reference*
- Chapter 2, “DATA Step Feature Engineering,” on page 19

Load a Database Table

**Prerequisites**

The following example assumes the following:

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

**Example**

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

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

proc casutil;
   list files; /* 2 */
   droptable casdata="sales" quiet;
   contents casdata="sales"; /* 3 */
   load casdata="sales" casout="sales" promote /* 4 */
      label="Fact table for User-to-Item Analysis"
      varlist={
         (name="USERID" label="User ID"),
         (name="ITEMID" label="Item ID")
      };
   contents casdata="sales"; /* 5 */
```
Add a caslib that uses Oracle as the data source. Oralib becomes the active caslib for the session and the subsequent programming statements use it for input and output.

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

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

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

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

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

Key Ideas

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

Save an In-Memory Table

Example

This example demonstrates the following:

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

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

/* From some other application, or a DATA step, the */
/* Customers table is modified with a change that */
/* is important to save. */
proc casutil incaslib="hps" outcaslib="hps";
```
The SAVE statement uses the Hps caslib from the OUTCASLIB= procedure statement option. The next time a LOAD statement is used with the Customers table, the table includes the changes. You can use the CASOUT= option to specify an alternative filename other than the default, customers.sashdat.

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

**Key Ideas**

- You can save in-memory tables as SASHDAT files or CSV files in path-based caslibs only. If you need to save a table from a caslib that is not path-based, then you can specify the OUTCASLIB= option.
- For information about data source connection parameters to use in the CASLIB statement, see “Data Connectors” in *SAS Cloud Analytic Services: Language Reference*.

---

Drop an In-Memory Table

**Example**

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

```sas
proc casutil incaslib="hps" outcaslib="hps";
    load casdata="sales.sashdat" casout="sales";  /* 2 */
run;

proc casutil incaslib="hps";
    droptable casdata="sales";  /* 2 */
run
```
1. The LOAD statement loads a file named Sales.sashdat from the Hps caslib.
2. The DROPTABLE statement drops the in-memory table.

**Key Ideas**

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

---

**Delete a File from a Caslib’s Data Source**

**Example**

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

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

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

**Key Ideas**

- You can delete files from path-based caslibs only.
- Include the filename suffix in the CASDATA= argument.

---

**Data Compression**

**Overview of Data Compression**

SAS Cloud Analytic Services supports and performs all compression for in-memory tables. When you transfer a table to the server in a DATA step and specify the COMPRESS= data set option, rows are sent to the server as is and the server compresses them.

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

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

**Compressed Tables and the DATA Step**

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

**Example Code 1.1  Creating a Compressed Table with a DATA Step**

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

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

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

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

**Example Code 1.2  Appending Rows to a Compressed Table**

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

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

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

**Example Code 1.3  Creating a Partitioned and Compressed Table**

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

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

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

**Performance Considerations**

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

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

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

**Interactions**

Here are the interactions for compressed tables and SAS programs.

- You can use a compressed table in programs like any other table.
- You can define calculated columns for compressed tables with the TEMPNAMES= data set option.
- You can append to compressed tables with the APPEND= data set option. This is also supported for compressed tables that have partitioning. However, you cannot append to a compressed table that is partitioned and has an ORDERBY= data set option specification.

**See Also**

- “APPEND= Data Set Option” in *SAS Cloud Analytic Services: Language Reference*
- “COMPRESS= Data Set Option” in *SAS Cloud Analytic Services: Language Reference*
- “ORDERBY= Data Set Option” in *SAS Cloud Analytic Services: Language Reference*
- “TEMPNAMES= Data Set Option” in *SAS Cloud Analytic Services: Language Reference*
Chapter 2
DATA Step Feature Engineering

Common Tasks for Running the DATA Step ........................................ 20
DATA Step Basics ................................................................. 21
  Distributed Data ............................................................. 21
  Distributed Processing ...................................................... 22
  About Multithreaded Execution ............................................ 22
  DATA Step Processing Modes ............................................. 23
  Controlling DATA Step Processing ...................................... 23
  Role of the DATA Step in SAS Viya ....................................... 26
  SAS Language Element Support for Processing in SAS and in CAS .. 28
  Restrictions for DATA Step Processing in CAS ....................... 29
  BY Processing in SAS Viya .................................................. 31
  DATA Types in SAS Viya ..................................................... 32
  Automatic Variables .......................................................... 37
  UTF-8 Session Encoding and Migrating SAS Data Sets .............. 39
  Examples ........................................................................... 39

Run a Simple DATA Step in CAS .................................................. 41
  Example ............................................................................ 41
  Key Ideas .......................................................................... 42
  See Also ............................................................................ 42

Convert a CAS Table to a SAS Data Set ....................................... 43
  Example ............................................................................ 43
  Key Ideas .......................................................................... 44
  See Also ............................................................................ 44

Save a SAS Data Set to CAS ...................................................... 44
  Example ............................................................................ 44
  Key Ideas .......................................................................... 45
  See Also ............................................................................ 45

Create a VARCHAR Variable Using the LENGTH Statement .......... 45
  Example ............................................................................ 45
  Key Ideas .......................................................................... 46
  See Also ............................................................................ 46

Check for Missing Values .......................................................... 46
  Example ............................................................................ 46
  Key Ideas .......................................................................... 47
  See Also ............................................................................ 47

Index CHAR and VARCHAR Character Strings ............................. 47
  Example ............................................................................ 47
### Common Tasks for Running the DATA Step

<table>
<thead>
<tr>
<th>Task</th>
<th>Example Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Run the DATA step in CAS</td>
<td><code>libname mycas cas;</code></td>
</tr>
<tr>
<td></td>
<td><code>data mycas.hello;</code></td>
</tr>
<tr>
<td></td>
<td><code>   put 'Hello from ' _hostname_</code></td>
</tr>
<tr>
<td></td>
<td><code>   'thread # ' _threadid_;</code></td>
</tr>
<tr>
<td></td>
<td><code>   x=1;</code></td>
</tr>
<tr>
<td></td>
<td><code>   run;</code></td>
</tr>
<tr>
<td>View DATA Step Processing Information Using Automatic Variables</td>
<td><code>libname mycas cas;</code></td>
</tr>
<tr>
<td></td>
<td><code>data mycas.hello;</code></td>
</tr>
<tr>
<td></td>
<td><code>   put 'Hello from ' _hostname_</code></td>
</tr>
<tr>
<td></td>
<td><code>   'thread # ' _threadid_;</code></td>
</tr>
<tr>
<td></td>
<td><code>   x=1;</code></td>
</tr>
<tr>
<td></td>
<td><code>   run;</code></td>
</tr>
</tbody>
</table>
Run the DATA step on a table in CAS

```sas
libname mycas cas;
proc casutil;
    load data=sashelp.cars;
run;
data mycas.cars2;
    set mycas.cars;
    KPL_City = 0.425 * MPG_City;
    KPL_Highway = 0.425 * MPG_Highway;
run;
```

Order rows in a table

```sas
libname mycas cas;
data mycas.baseball;
    set mycas.baseball (orderby=(nHome));
run;
```

Create CHAR and VARCHAR Variables

```sas
libname mycas cas;
data mycas.string;
    length x varchar(20);
    length y $10;
    x = 'abc'; y = 'def';
run;
```

Merge CAS tables

```sas
libname mycas cas;
data mycas.table3;
    set mycas.table1 mycas.table2;
    by var1 var2;
run;
```

For a complete list of examples, see “Examples” on page 39.

---

**DATA Step Basics**

**Distributed Data**

In SAS Viya, the SAS DATA step can be used to prepare data for analysis and predictive modeling. The DATA step runs in both a traditional SAS client session as well as in SAS Cloud Analytic Services (CAS) on in-memory tables. CAS provides the high-performance environment for running the DATA step in parallel and in multiple threads on distributed data. To get started quickly using the DATA step in SAS Viya, see “Run a Simple DATA Step in CAS” on page 41.

The CAS environment can consist of a single-machine server in which the DATA step runs in multiple threads over partitioned data or, it can consist of a cluster of homogenous servers over which data is distributed and processing workload is shared. In either case, the DATA step in CAS runs in multiple threads to enhance performance. For information about CAS architecture and CAS Data, see “Architecture” in *SAS Cloud Analytic Services: Fundamentals* and “Data” in *SAS Cloud Analytic Services: Fundamentals*. 
Distributed Processing

The SAS DATA step and a select set of SAS language elements are supported for processing in CAS. This means that the processing is happening in a CAS server session across multiple computers in parallel. Or, in the case of a single-machine deployment, the DATA step is processing on a single computer in multiple threads across partitioned data. When the DATA step runs in either of these scenarios, it is “running in CAS” or running in a CAS server session.

When the DATA step runs in CAS, the same DATA step program is replicated across the CAS cluster (of computers), where each DATA step thread runs on a portion of the data. By default, the DATA step runs in all available threads on every computer node in the cluster. When the DATA step executes on a node, the processing is done only on the rows that are allocated to that node.

The following image shows how the DATA step runs in multiple threads on multiple computers in a CAS cluster.

Not all SAS language elements are appropriate for distributed processing in the CAS environment. These language elements are supported only for a DATA step that is processing in a SAS client session (and not in CAS). See “Language Elements Supported in a SAS DATA Step Only” on page 29 for more information about these language elements.

The DATA step automatically runs in CAS when you specify a CAS engine libref on both the input and output tables. There are some restrictions on what language elements are supported in a DATA step running in CAS. See Table 2.1 on page 29 for more information about these restrictions.

For information about determining where the DATA step is running and how to control where it runs, see “Controlling Where Your DATA Step Is Running” on page 23.

About Multithreaded Execution

When dividing data among several threads of a DATA step, the results might not be the same as when one thread is used. For example, when a RETAIN statement is used, a value is retained from one row to the next. Often this is used to create a sum from all rows. Because each thread operates on part of the data, each thread holds and stores a
partial sum. A second step, that runs in one thread across the entire cluster, is used to sum the partial sums. When RETAIN, LAG, and DIF are used, loop-carried dependencies can occur.

**DATA Step Processing Modes**

In the SAS Viya environment, the DATA step can run either locally in a SAS client session or it can run in CAS. When programming with the SAS DATA step in SAS Viya, it is important to understand where the computation is taking place because performance can become an issue with very large data sets. Also, because not all SAS language elements run in both environments, you need to be aware of where the DATA step is processing so that you know what features are available to you for writing programs in SAS.

**SAS client session processing**

executes outside the CAS environment in a local SAS session. This is the traditional SAS DATA step processing mode in which the DATA step runs in a single thread on a traditional SAS Studio server. Even though DATA step processing is external to CAS, there can be an interaction with CAS if the DATA step is being used to transform SAS data sets to CAS tables. When the DATA step is being used like this (as a SAS to CAS data loader), the output data is sent to CAS, where it exists as an in-memory CAS table. The output data is loaded to CAS, but the statements within the DATA step are processed locally, in the client session, prior to the data loading.

For information about using the DATA step to transform data sets into CAS tables, see Role #2 on page 26 in “Role of the DATA Step in SAS Viya” on page 26.

**CAS server session processing**

executes in the CAS server in multiple threads over distributed data in a multi-machine CAS server or over partitioned data in a single-machine CAS server. In both scenarios, the DATA step code is executed in each thread on each node. Both the input data and the output tables must be specified using a libref that you have created using the CAS engine.

For information about using the DATA step to manipulate CAS tables, see Role #3 on page 26 in “Role of the DATA Step in SAS Viya” on page 26).

**Controlling DATA Step Processing**

**Controlling Where Your DATA Step Is Running**

This section describes how to control where the DATA step is running after you have logged in to SAS Studio and started a CAS session.

The DSACCEL= system option controls whether the DATA step automatically runs in CAS in a CAS server session. The DSACCEL= option can be set to one of two values: ANY or NONE. In SAS Viya, DSACCEL is automatically set to ANY, which means that, by default, a valid DATA step automatically runs in CAS when the following conditions are true:

1. All librefs in the DATA step are CAS engine librefs to the same CAS session.
2. All statements in the DATA step are supported by the CAS DATA step. See “SAS Language Element Support for Processing in SAS and in CAS” on page 28 for a list of SAS language elements that are supported in CAS.

**CAUTION:**
If you try to run a DATA step in CAS, but you use a language element that is not supported in CAS (such as the INPUT or DATALINES statement), then the DATA step will automatically run in SAS rather than generate an error.

Note: If you explicitly set the DSACCEL= option to NONE, then the DATA step automatically runs in SAS and you must specify the SESSREF= data set option to enable it to run in CAS.

In the following examples, both DATA steps run in CAS, but in Example 1, the SESSREF= option is required because the DSACCEL option is set to NONE. In Example 2, you do not have to explicitly set the DSACCEL option to ANY since it is already on by default in SAS Viya.

Example 1:
```plaintext
options dsaccel=none msglevel=i;
libname mycas cas;
data mycas.class2 / sessref="casauto";
set mycas.class;
run;
```

Example 2:
```plaintext
options dsaccel=any;
libname mycas cas;
data mycas.class2;
set mycas.class;
run;
```

In the examples above, the DATA step is running on tables that were already loaded into CAS. For an example of loading a SAS data set into CAS using PROC CASUTIL, see “Load a Client-Side File” on page 4.

Determining Where Your DATA Step Is Running

- Look at your log output. If you see the note Running DATA step in Cloud Analytic Services, then your DATA step ran in CAS.

Figure 2.1 Log output showing DATA step running in CAS and in SAS

- Look at the librefs specified on your input and output table names. Both the input and output tables must use the same CAS libref and all language elements in the DATA step must be supported in CAS. See “SAS Language Element Support for
Processing in SAS and in CAS on page 28 for a list of supported and unsupported language elements.

**Figure 2.2** Libraries Pane in SAS Studio showing CAS and SAS libraries and how they are used in DATA step code

- Set the MSGLEVEL= system option to i in addition to setting the DSACCEL option to ANY, and check the log for detailed information about where the DATA step processed and why it processed there.

```sas
options dsaccel=any msglevel=i;
data mycas.air(where=(air>480));
set mycas.air;
run;
```

**Figure 2.3** Log output showing DATA step execution in SAS when it encounters an unsupported language element

**: Note:** When the DATA step runs in CAS, the SINGLE= option is set to NOINPUT instead of NO. This causes a DATA step without input tables to run in one thread instead of in all available threads in CAS.

### Controlling the Number of Threads Your DATA Step Is Running In

When the DATA step runs in CAS, it automatically runs in all available threads on all nodes in the CAS environment. For example, if the CAS environment consists of a cluster of 4 worker nodes and each node supports 32 threads, then the DATA step will run in \(4 \times 32 = 128\) threads.

This default behavior is true only for a DATA step that has input data. A DATA step that has no input data runs in a single thread by default.

You can control whether the DATA step runs in a single thread or in all available threads by specifying the SINGLE= option with the SESSREF= option in the DATA statement.

```sas
libname mycas cas;
data mycas.class / sessref="casauto" single=yes;
set mycas.class;
run;
```
When the SINGLE= option is set to YES, the DATA step runs in a single thread. When the SINGLE= option is set to NO (which is the default for DATA steps with input tables), the DATA step runs in all available threads. When the SINGLE= option is set to NOINPUT, the DATA step runs in a single thread only if there is no input data, and it runs in multiple threads if there is input data. See “SINGLE=NO | YES | NOINPUT” in SAS Viya Statements: Reference for syntax information related to this option.

Note: In the example above, "CASAUTO" is the default CAS session that starts automatically when you log on to SAS Studio. See “Sessions” in SAS Cloud Analytic Services: Fundamentals for more information about CAS sessions.

Role of the DATA Step in SAS Viya

If you are familiar with the SAS DATA step, then you know that its primary function is to enable you to create and manipulate SAS data sets and prepare them for analysis. In SAS Viya, the SAS DATA step serves 3 distinct roles:

1. **traditional** SAS DATA step - for manipulating SAS client-session (local) data and performing various data preparation tasks on local data in the SAS Studio client session. There is no interaction with CAS and SAS uses only the V9 engine for data access. Here is a code example:
   ```sas
   data work.mycars;
      set sashelp.cars(where=(weight>6000));
      keep make model type weight MPG_City;
   run;
   ```

2. a **data loader** that uses the CAS engine to transform SAS data sets into CAS tables.
   
   Note: It is recommended that you use the CASUTIL procedure rather than the DATA step for loading SAS data sets into CAS. See the example, “Load a Client-Side File” on page 4.

   The DATA step processing is done in the local SAS client session and not in CAS. Even though there is interaction with CAS in that the DATA step and the CAS engine send the output to CAS as an in-memory table, the processing is done locally. You must use a CAS engine libref to name the output data set so that SAS knows to output the data set to CAS. Once the data set is loaded into CAS, you can use a separate DATA step to execute processing in CAS on CAS tables, as shown in Role 3. CAS table manipulator on page 26. Here is a code example that shows loading a SAS data set into CAS.
   ```sas
   libname mycas cas;
   data mycas.cars(where=(weight>6000));
      set sashelp.cars;
      keep make model type weight MPG_City;
   run;
   ```

   For an example showing how to load and save a SAS data sets as a CAS table, see “Load a Client-Side File” on page 4.

3. a **CAS table manipulator** for creating and manipulating in-memory CAS tables in parallel in a distributed CAS server environment. Both the input and output data must use a CAS engine libref in the same session. Here is a code example that shows the DATA step processing in CAS. The DATA step reads in the in-memory CAS table and outputs a new in-memory CAS table in the same session. All data is in CAS and all processing is done in CAS.
   ```sas
   libname mycas cas;
   data mycas.cars2;
   ```
The following table lists summary information about the three ways that you can use the DATA step in SAS Viya.

<table>
<thead>
<tr>
<th>Role: Manipulate CAS Tables in a CAS server session</th>
<th>Load SAS Data Set into CAS</th>
<th>Manipulate a SAS Data Set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Sample:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>data mycas.class;</td>
<td>data mycas.cars;</td>
<td>data cars;</td>
</tr>
<tr>
<td>set mycas.class;</td>
<td>set sashelp.cars;</td>
<td>set sashelp.cars;</td>
</tr>
<tr>
<td>if age&lt;15 then delete;</td>
<td>if age&lt;15 then delete;</td>
<td>if age&lt;15 then delete;</td>
</tr>
<tr>
<td>run;</td>
<td>run;</td>
<td>run;</td>
</tr>
<tr>
<td>Requirements:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• CAS engine libref for both input and output table names.</td>
<td>• CAS engine libref on the output table name</td>
<td>• CAS engine libref on the output table name</td>
</tr>
<tr>
<td>Properties:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>• processes CAS tables in parallel in CAS.</td>
<td>• processes in the SAS client session but loads a SAS data set to CAS as an in-memory CAS table using the CAS engine.</td>
<td>• runs independently of CAS in a SAS client session.</td>
</tr>
<tr>
<td>• CAS engine librefs on both the input and output data sets.</td>
<td>• manipulates SAS client session data before loading it into CAS as an in-memory table.</td>
<td>• processes in the SAS client session (not in CAS) as a traditional SAS DATA step. For information about what is supported in SAS only, see the “for SAS Only” documents here: “Language Elements Supported in a CAS DATA Step” on page 28</td>
</tr>
<tr>
<td>• processes CAS supported language elements. For a list of these language elements, see “Language Elements Supported in a CAS DATA Step” on page 28</td>
<td>• runs independently of CAS in a SAS client session.</td>
<td>• processes in the SAS client session (not in CAS) as a traditional SAS DATA step. For information about what is supported in SAS only, see the “for SAS Only” documents here: “Language Elements Supported in a CAS DATA Step” on page 28</td>
</tr>
</tbody>
</table>

Data Access Method (Engine): CAS engine V9 and CAS engine V9 engine

Input: in-memory CAS table SAS data set SAS data set (and other V9 engine supported data)

Output: in-memory CAS table in-memory CAS table

Output File Format: .sashdat .sashdat .sas7bdat

DATA types: DOUBLE, CHAR, VARCHAR DOUBLE, CHAR VARCHAR (only with the CAS engine) DOUBLE, CHAR
Manipulate CAS Tables in a CAS server session | Load SAS Data Set into CAS | Manipulate a SAS Data Set
--- | --- | ---
Session encoding: The default session encoding in SAS Viya (in both SAS and CAS sessions) is UTF-8
See “UTF-8 Session Encoding and Migrating SAS Data Sets” on page 39 for more information.
Language elements support: See “SAS Language Element Support for Processing in SAS and in CAS” on page 28

- You can load a SAS data set directly into CAS using a caslib. This is known as “server-side data loading.” For information about caslibs and server-side data loading see “What is a caslib?” in SAS Cloud Analytic Services: Fundamentals, “Caslibs” in SAS Cloud Analytic Services: Fundamentals, and “CASLIB Statement” in SAS Cloud Analytic Services: Language Reference.
- .sashdat files are not the only types of files that the DATA step can manage in CAS. See examples for loading and saving data in “Save an In-Memory Table” on page 13.

### SAS Language Element Support for Processing in SAS and in CAS

#### About SAS Language Element Support in SAS Viya

The following sections provide links to the reference documentation for each type of SAS language element that is supported in SAS Viya. SAS language elements in SAS Viya are categorized into two groups: those that can be used in a DATA step that is processing in CAS and those that are supported only in a DATA step that is processing in the SAS client session (and not in CAS).

#### Language Elements Supported in a CAS DATA Step

Language elements that are supported in the CAS environment are a subset of the traditional SAS Language. Functionality offered by some SAS language elements, like reading text files, does not translate well to parallel execution and is therefore not supported in distributed or multithreaded CAS environment. In order for a DATA step to be processed in CAS, it must have a CAS table as both its input and output (using a CAS engine libref).

The following list shows the syntax documents for the SAS language elements that are supported for CAS DATA step processing:

**Component Objects**

“Dictionary of Hash and Hash Iterator Object Language Elements for SAS and the CAS Server” in SAS Viya Component Objects: Reference

**Data Set Options**

“Dictionary of Data Set Options for SAS and the CAS Server” in SAS Viya Data Set Options: Reference

**Formats and Informats**

“Dictionary of Formats for SAS and the CAS Server” in SAS Viya Formats and Informats: Reference
Functions and CALL Routines
“Dictionary of Functions and Call Routines for SAS and the CAS Server” in SAS Viya Functions and CALL Routines: Reference

Statements
“Dictionary of Statements for SAS and the CAS Server” in SAS Viya Statements: Reference

Language Elements Supported in a SAS DATA Step Only
A DATA step that runs in SAS only is one that processes in the local SAS client session but not in CAS. The following language elements are supported only for SAS client-session processing.

Note: A step that runs in the SAS client session but that uses the CAS engine to output data to CAS is supported by the CAS engine but is not necessarily processing in CAS. The DATA step can only process in CAS if both the input and output tables are CAS tables.

Component Objects

Data Set Options
“Dictionary of Data Set Options for SAS Only” in SAS Viya Data Set Options: Reference

Formats and Informats
“Dictionary of Formats for SAS Only” in SAS Viya Formats and Informats: Reference

Functions and CALL Routines
“Dictionary of Functions and CALL Routines for SAS Only” in SAS Viya Functions and CALL Routines: Reference

Statements
“Dictionary of Statements for SAS Only” in SAS Viya Statements: Reference

Note: Language elements labeled as “for SAS Only” might run in a DATA step that interacts with CAS in some way, such as a DATA step that outputs to CAS, but this does not mean that the DATA step is processing in CAS. For example, you can use the PROC PRINT or PROC CONTENTS procedures to view CAS tables and you can use the DATALINES and INPUT statements to create raw data in a DATA step that outputs to CAS, but these are not valid for a DATA step that is processing in CAS. For more information see “Role of the DATA Step in SAS Viya” on page 26 and “Determining Where Your DATA Step Is Running” on page 24.

Restrictions for DATA Step Processing in CAS
The following table contains restrictions on the use of some DATA step language elements in CAS. See “SAS Language Element Support for Processing in SAS and in CAS” on page 28 for a list of reference documentation for supported SAS language elements.

<table>
<thead>
<tr>
<th>Name of Feature</th>
<th>Description</th>
</tr>
</thead>
</table>

Table 2.1 Restrictions on the DATA Step in SAS Viya and in CAS
<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BY group processing</td>
<td>BY variables are supported in CAS but there are a few differences in behavior that should be noted. See the following sections for details: “Using Formats with BY Variables” on page 31, “Ordering BY Variables” on page 32, and the example “Order BY Variables in CAS” on page 50.</td>
</tr>
<tr>
<td>BY statement, DESCENDING option</td>
<td>The DESCENDING option in the BY statement is not supported in a DATA step that is running in CAS.</td>
</tr>
<tr>
<td>Functions and Byte-Length Semantics</td>
<td>SAS functions use byte-length semantics when operating on SAS character strings. In other words, SAS functions assume that one character is always equal to one byte and they return results that are based on this assumption. However, in a UTF-8 environment, your data might contain multi-byte characters, in which one character does not equal one byte. So, string functions that parse character strings to calculate length and positional values might return unexpected results. For example, the LENGTH function returns the number of bytes in the specified string, not the number of characters. See “Index CHAR and VARCHAR Character Strings” on page 47 for an example of this behavior in functions.</td>
</tr>
<tr>
<td>INFILE, INPUT and DATALINES statements</td>
<td>These language elements are supported in a DATA step that is running in the SAS client session. This includes the DATA step that is being used to load data into CAS. For example, you can specify the following valid DATA step because the processing is being done in SAS in the client (not in CAS): The DATA step uses the CAS engine to load data to CAS, but the processing is being done in SAS.</td>
</tr>
<tr>
<td>Macros and the Macro Facility</td>
<td>Macros are supported in SAS Viya, but only in the SAS client session. Macros can be useful in generating the code that runs in CAS, but the macro language itself is not supported in CAS.</td>
</tr>
<tr>
<td>MODIFY, REMOVE, and REPLACE statements</td>
<td>These statements are not supported in a DATA step that is running in CAS and you cannot use them to modify an in-memory CAS table. You can use these statements in a SAS client-session DATA step and then load the results to CAS.</td>
</tr>
<tr>
<td>SUBSTR(left of =) Function</td>
<td>The SUBSTR(left of =) function is not supported for VARCHAR data types.</td>
</tr>
<tr>
<td>VARCHAR Data Type</td>
<td>There are some restrictions on uses of the VARCHAR data type. See “VARCHAR Data Type” on page 33.</td>
</tr>
</tbody>
</table>
WHERE= The WHERE= data set option is not supported in the DATA statement in a DATA step that is processing in CAS. Specifying WHERE= on the output data set of a CAS DATA step causes the DATA step to run locally in the SAS client session.

For example, the following DATA step is processed in the SAS client session since the WHERE= statement is not supported in the DATA statement in a DATA step running in CAS:

```sas
libname mycas cas;
data mycas.air(where=(air>450));
  set mycas.air;
run;
```

However, you can use the WHERE= option on the input data set and still process the DATA step in CAS:

```sas
libname mycas cas;
data mycas.air;
  set mycas.air(where=(air>450));
run;
```

See “Controlling DATA Step Processing” on page 23.

Some restricted features will be made available in future releases of SAS Viya.

**BY Processing in SAS Viya**

**Using Formats with BY Variables**

When executing a BY statement that has multiple BY variables (BY name date; for example), CAS first partitions the rows of the input table using the first BY variable. Within each partition, it then orders the rows by all the BY variables. CAS uses the formatted value of the first BY variable rather than the raw values to partition the data on the CAS server.

An entire partition is operated on by a single DATA step thread. If there is low cardinality in the formatted values of the first BY variable, then fewer threads will be used. Therefore, choose your first BY variable wisely because it affects the performance of the job. For example, if you choose gender as your first BY variable, then only a few DATA step threads will be used to process the entire table.

See “Order BY Variables in CAS” on page 50 for an example that assigns the BY variable to a higher cardinality format. Here are some examples of low and high cardinality formats:

<table>
<thead>
<tr>
<th>Low-Cardinality formats</th>
<th>High-Cardinality formats</th>
</tr>
</thead>
<tbody>
<tr>
<td>• QTRw.</td>
<td>• DATEw.</td>
</tr>
<tr>
<td>• MONYY.</td>
<td>• DATETIMEw.d</td>
</tr>
<tr>
<td>• WEEKDAYw.</td>
<td>• DDMMYYw.</td>
</tr>
</tbody>
</table>

See *SAS Viya Formats and Informats: Reference* for syntax information about formats in SAS and “User-Defined Format Basics” on page 65 for more information about using SAS formats in SAS Viya and CAS.
Ordering BY Variables
The order that rows are added to a CAS table are not necessarily the order they are processed. In Base SAS, the Base engine guarantees the ordering of rows. CAS tables do not make this guarantee. CAS tables are split among multiple grid nodes. When partitioning occurs, BY groups are formed on a single node from rows that were on multiple worker machines. Therefore, the ordering of rows within a partition (or, within a BY group) is not stable.

If you need a stable ordering, use more BY variables. For example, if you use the BY statement to process BY ID, the rows are grouped by formatted value for ID, then ordered by raw value for ID. The order the rows were added to the table is not considered. If you use the BY statement to process BY ID DATE; then the rows are grouped by the formatted value of ID, then ordered by the raw values for ID and DATE.

See the following examples for help with BY variables and ordering: “Order BY Variables in CAS” on page 50 , “Order Rows in a Table Using ORDERBY=” on page 53 , and “Format BY Variables” on page 52.

Note: The DESCENDING option in the BY statement is not supported in a DATA step that is running in CAS.

DATA Types in SAS Viya

Overview
There are 3 data types in SAS Viya: CHAR, DOUBLE, and (with the CAS engine), VARCHAR. The following table provides information for each type.

Table 2.2  DATA Types in SAS Viya

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Description</th>
<th>Example</th>
<th>Common Methods for Creating</th>
<th>Missing Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>V9 Engine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>CHAR(n)</td>
<td>Stores a fixed-length character string, where n is the maximum number of characters to store. The maximum number of characters is required to store each value regardless of the actual size of the value.</td>
<td>data greeted;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>length greet1 $10;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>greet1=&quot;good morning&quot;;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>format greet2 $char8.;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>greet2=&quot;hi&quot;;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>attrib greet3</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>label=&quot;Greeting&quot;;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>run;</td>
<td>LENGTH Statement, FORMAT statement, ATTRIB Statement, assignment statement, SET statement, MERGE statement, MODIFY statement, and UPDATE statement</td>
<td>all blanks</td>
</tr>
<tr>
<td>DOUBLE(numeric)</td>
<td>Stores a numeric value, including dates and times, as a floating-point number.</td>
<td>data info2;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>set mycas.info2;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>length num $8;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>num = 25;</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>run;</td>
<td>LENGTH statement, FORMAT statement, ATTRIB statement, assignment statement, SET statement, MERGE statement, MODIFY statement, and UPDATE statement</td>
<td>( . )</td>
</tr>
<tr>
<td>CAS Engine</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
### CHAR Data Type

Stores a fixed-length character string, where \( n \) is the maximum number of characters to store. The maximum number of characters is required to store each value regardless of the actual size of the value.

```sas
libname mycas cas;
data mycas.greets;
length greet1 $10;
greet1="good morning";
format greet2 $char8.;
greet2="hi";
attrib greet3 label="Greeting";
run;
```

LENGTH Statement, FORMAT statement, ATTRIB Statement, assignment statement, SET statement, MERGE statement, MODIFY statement, and UPDATE statement

### DOUBLE Data Type

Stores a 64-bit (8-byte) double precision, IEEE floating-point number. Allows numbers of large magnitude and permits computations that require many digits of precision to the right of the decimal point.

A minimum length of 8 is assigned automatically if you specify a length less than 8 when running the DATA step in CAS.

```sas
data mycas.nums;
x=5;
length y 8; y=1;
format z best.;
z=y;
run;
```

LENGTH statement, assignment statement, ( . )

### VARCHAR Data Type

Stores a varying-length character string, where \( n \) is the maximum number of characters to store.

```sas
data mycas.info3;
length lastname varchar(20);
set mycas.info2;
lastname="Catelyn";
run;
```

LENGTH statement, SET statement, ARRAY statement

V ARCHAR(\( n \))

Stores a varying-length character string, where \( n \) is the maximum number of characters to store.

### See “Data Types” in SAS Cloud Analytic Services: Language Reference for more information.

**VARCHAR Data Type**

The VARCHAR type is declared with a length that represents the maximum number of characters you want to store. For example, VARCHAR(20) can store up to 20 characters. This is in contrast to the CHAR data type whose length is measured in bytes and is fixed in length.

The VARCHAR data type is a varying-length data type, which means that a VARCHAR(20) column can store up to 20 characters, but the actual storage used is determined by the lengths of the individual values in the column. For example, a column defined as a VARCHAR(10) column can store up to 10 characters but the string value “hello” uses only half of the reserved storage space. A fixed-length column, on the other hand, requires the defined number of bytes regardless of the actual size of the data. The CHAR data type is fixed-length.

The VARCHAR data type is useful when you have character columns that contain values with different lengths. If the values in a character column do not change in length, for example, in the case of a column containing 3-letter airport codes, then it is probably better to use the CHAR fixed-length data type.

VARCHAR is supported by the CAS engine but not by the SAS V9 engine. This means that the VARCHAR data type is supported in both the CAS DATA step and in the DATA step running in SAS, as long as the CAS engine is used for data access. If the DATA step is processing in SAS, then you must output data to a CAS table using a CAS engine.
libref to get full support of VARCHAR data types. The VARCHAR data type is also useful for reading in multi-byte character data.

For more information about the CAS engine, see “CAS LIBNAME Statement” in SAS Cloud Analytic Services: Language Reference.

Syntax

```
LENGTH variables(s) VARCHAR(length|*);
ARRAY arrayname[N] VARCHAR(length|*);
ARRAY arrayname[*] varchar-variables;
ARRAY arrayname[N] VARCHAR(length|*) _temporary_;
```

variable(s)

One or more variables that are assigned the type VARCHAR.

length

A numeric constant that is the maximum number of characters stored in the VARCHAR variable. This value can up to \(2^{31} - 1\).

* 

An asterisk means to use the maximum length allowed, which is currently \(2^{31} - 1\).

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

Example

```
libname mycas;

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

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

When you output a CAS table that contains VARCHAR variables to a SAS data set, all VARCHAR type variables are converted to CHAR type variables. The length of the CHAR variables depends on how the VARCHAR lengths are defined. If a VARCHAR variable is defined for its maximum length (for example, `x varchar(*)`), then the VARCHAR will be converted to a CHAR type with a length of 32767 bytes. If a VARCHAR is defined with a specific length (for example, `x varchar(20)`), then the VARCHAR will be converted to a CHAR with a length (in bytes) that is equal to 4 times the length of the VARCHAR.

```
data mycas.test;
  length x varchar(*);
  length y varchar(20);
  x="hello";
  y="goodbye";
run;
proc contents data=mycas.test; run;
/* transform the CAS table into a SAS data set */
data test;
  set mycas.test;
```
run;
proc contents data=test; run;

Log 2.1  Log Output

NOTE: One or more variables were converted because the data type is not supported by the V9 engine.
For more details, run with options MSGLEVEL=I.

Output 2.1  PROC CONTENTS Output for CAS Table "Mycas.Test"

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>Varchar</td>
<td>536870911</td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td>Varchar</td>
<td>20</td>
</tr>
</tbody>
</table>

Output 2.2  PROC CONTENTS Output for SAS Data Set “Test”

<table>
<thead>
<tr>
<th>#</th>
<th>Variable</th>
<th>Type</th>
<th>Len</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>x</td>
<td>Char</td>
<td>32767</td>
</tr>
<tr>
<td>2</td>
<td>y</td>
<td>Char</td>
<td>80</td>
</tr>
</tbody>
</table>

Table 2.3  Restrictions on the VARCHAR Data Type

<table>
<thead>
<tr>
<th>Feature or Language Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ATTRIB</td>
<td>You cannot use the ATTRIB statement to create VARCHAR variables.</td>
</tr>
<tr>
<td>BY statement</td>
<td>The BY statement uses a fixed width for VARCHAR variables. Using a VARCHAR(*) type in the BY statement might cause unexpected results.</td>
</tr>
<tr>
<td>DATA step views</td>
<td>VARCHAR variables are not supported for DATA step views.</td>
</tr>
<tr>
<td>FILE PRINT ODS with PUT statement to ODS</td>
<td>VARCHAR variables are not supported with the PUT statement when the DATA step outputs to ODS. For example, you get an error if you submit the following:</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>data <em>null</em>;</td>
</tr>
<tr>
<td></td>
<td>length firstname varchar(20) lastname varchar(20);</td>
</tr>
<tr>
<td></td>
<td>file print ods;</td>
</tr>
<tr>
<td></td>
<td>put <em>ods</em>;</td>
</tr>
<tr>
<td></td>
<td>run;</td>
</tr>
<tr>
<td>Format widths</td>
<td>Format widths with VARCHAR variables are measured in terms of bytes rather than characters.</td>
</tr>
<tr>
<td>Hash object key</td>
<td>VARCHAR variables are not supported for Hash objects.</td>
</tr>
<tr>
<td>Function</td>
<td>Description</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>INDEX function</td>
<td>The INDEX function treats VARCHAR and CHAR type variables differently. For VARCHAR data types, the INDEX function returns the character position of the substring within a string and for CHAR data types, the INDEX function returns the byte position of the substring within the string. See “Index CHAR and VARCHAR Character Strings” on page 47 for an example that shows how the INDEX function processes VARCHAR and CHAR type variables.</td>
</tr>
<tr>
<td>INPUT statement</td>
<td>The INPUT statement does not support the VARCHAR data type. It supports CHAR and NUMERIC data types only.</td>
</tr>
<tr>
<td>KEY= on SET and MODIFY statements</td>
<td>VARCHAR variables are not supported by the KEY= option in either the SET or MODIFY statements.</td>
</tr>
<tr>
<td>LAG function</td>
<td>VARCHAR variables are not supported by the LAG function.</td>
</tr>
<tr>
<td>MODIFY with BY statement</td>
<td>You cannot open a table for Update access using the MODIFY statement with the BY statement if the table contains VARCHAR variables.</td>
</tr>
<tr>
<td>PUT statement (to ODS output)</td>
<td>VARCHAR variables are not supported with the PUT statement when it outputs to ODS.</td>
</tr>
<tr>
<td>SUBSTR(left of =) function</td>
<td>The SUBSTR(left of =) function is not supported for VARCHAR variables.</td>
</tr>
<tr>
<td>TYPEOF function</td>
<td>The TYPEOF function returns “C” instead of “VC” for VARCHAR variables.</td>
</tr>
<tr>
<td>VARCHAR</td>
<td>In SAS Viya, the VARCHAR data type is supported only by the CAS engine. It is not supported by the V9 engine. This means that the VARCHAR data type is supported only for a DATA step that outputs to CAS using the CAS engine or that is processing in CAS using the CAS engine.</td>
</tr>
</tbody>
</table>
| Variable Lists    | Selecting a character variable range for character variable lists (for example, a-character-f) is not supported for VARCHAR variables because VARCHAR variables are not fixed-width character variables. For example, you cannot specify VARCHAR variables using the following shorthand forms: 
  _CHARACTER_ 
  var1–varN 
  var1–CHARACTER-varN 
  sum(of variable-name-prefix) |
| VNEXT Function    | The VNEXT function returns “C” instead of “VC” for VARCHAR variables. |

For more information about Data Types in Cloud Analytic Services, see “SAS Cloud Analytic Services Data Types” in SAS Cloud Analytic Services: Language Reference
Automatic Variables

Automatic variables are created by the DATA step and saved in memory, but they are not output to the data set being created. The values of automatic variables are retained from one iteration of the DATA step to the next, rather than set to missing. To see an example that uses automatic variables see “View DATA Step Processing Information Using Automatic Variables” on page 59.

The following automatic variables are created by every DATA step that runs in a SAS client session or in CAS.

_ERROR_
valid in the SAS client session and in the CAS server session.

_ERROR_ is 0 by default but is set to 1 whenever an error is encountered, such as an input data error, a conversion error, or a math error, as in division by 0 or a floating point overflow. You can use the value of this variable to help locate errors in data records and to print an error message to the SAS log.

For example, the following IF statements write to the SAS log during each iteration of the DATA step. The DATA step writes the contents of the input record when an error is encountered:

Example:
libname mycas cas;
data test;
  input x y;
  if _error_ = 0 then
    put "Error Code is " _error_ "for row " _n_ ";
datalines;
  1 1
  2 2
  3 3
  4 4
;run;

Figure 2.4 Partial Log Output for _ERROR_ Automatic Variable for DATA Step Running in SAS

_N_
inicially set to 1. Each time the DATA step loops past the DATA statement, the variable _N_ increments by 1. The value of _N_ represents the number of times the DATA step has iterated, which translates to the number of rows in a data set. You can use the PUT statement with the _ALL_ argument to print the values of _N_ and _ERROR_ to the SAS log.

Example 1:
libname mycas cas;
data mycas.test;
input x y;
   put "This is iteration (row) # " _n_;
datalines;
  1 1
  2 2
  3 3
  4 4
;
run;

Figure 2.5  Partial Log Output for _N_ Automatic Variable for a DATA Step Running in SAS

```
This is iteration (row) # 1
This is iteration (row) # 2
This is iteration (row) # 3
This is iteration (row) # 4
NOTE: The data set MYCAS.TEST has 4 observations and 2 variables.
```

In a CAS server session, _N_ returns the number of DATA step iterations (table rows) per node per thread. For example, if you have rows distributed across 4 worker nodes (with each node holding 1 row), then you would see one DATA step iteration (row) appear 4 times in the log output - one for each node (row 1 on node 0, row 1 on node 1, row 1 on node 2, row 1 on node 3, and so on).

Example 2:
```
data mycas.test;
   set mycas.test;
   put "Iteration # " _n_ "of DATA step running in thread # "
       threadid_ "on node # " _rankid_;
run;
```

Log output:

Figure 2.6  Log Output for _N_ Automatic Variable for a DATA Step Running in CAS

```
NOTE: Running DATA step in Cloud Analytic Services.
Iteration # 1 of DATA step running in thread # 40 on node # 0
Iteration # 1 of DATA step running in thread # 33 on node # 2
Iteration # 1 of DATA step running in thread # 17 on node # 1
Iteration # 1 of DATA step running in thread # 1 on node # 0
NOTE: There were 4 observations read from the table TEST in caslib CASUSER( ).
NOTE: The table test in caslib CASUSER( ) has 4 observations and 2 variables.
```

_NTHREADS_
_NTHREADS_ returns the number of DATA step threads running in a CAS server session.

_HOSTNAME_
_HOSTNAME_ returns the name of the worker node or host that the DATA step is running on in a CAS server session.

_RANKID_
_RANKID_ returns the 0-based number that is associated with the worker node that the DATA step is running on in a CAS server session. The _RANKID_ is listed from 0 to n.

_THREADID_
_THREADID_ returns the number that is associated with the thread that the DATA step is running in a CAS server session.
UTF-8 Session Encoding and Migrating SAS Data Sets

The default session encoding for all processing modes and environments in SAS Viya is UTF-8. UTF-8 is a character encoding that follows the Unicode standard and is capable of encoding all possible characters in the Unicode standard. In SAS 9 and other prior releases, the default session encoding is Latin1 for UNIX and Linux and wLatin1 for Windows environments. When you read data sets into SAS Viya from these earlier environments, you might see log messages indicating the differences between the two encodings.

SAS/CONNECT can convert data from a non-UTF-8 encoded SAS session to the UTF-8 format that SAS Viya requires and it is required for moving data from other SAS deployments into SAS Viya. SAS/CONNECT is not included with a standard SAS Viya Data Mining and Machine Learning order. You must order it separately. If you order SAS/CONNECT, the required commands to install it are automatically included in your purchase. Here are some resources for handling the migration of your legacy SAS data into the SAS Viya environment:

- Example: “Read a SAS 9 Data Set into SAS Viya” on page 56
- “LIBNAME Statement, CVP Engine” in SAS Viya Statements: Reference
- “COPY” in SAS Viya Utility Procedures Guide

Examples

Prerequisites
The following examples assume that you have executed the required code needed to sign on to SAS Studio and to activate the default SAS Studio session. The required code consists of the following two statements, which you must run each time you log in to SAS Studio:

```r
cas casauto;
caslib _all_ assign;
```

Or, you can add these statements to your autoexec file so that everytime you log into SAS Studio using your credentials, the statements will execute automatically and you do not have to add them manually. To do this, select the icon in the Code Editor window, select Edit Autoexec File from the drop-down menu,
and paste the statements above into the Autoexec.sas window.

```
Autoexec.sas Log

Specify the SAS statements to run when the application starts.

cas casauto;
caslib _all_ assign;
```

Once you have clicked Save and closed the Autoexec window, you can run the examples.

These statements activate the default CAS Studio session and associate it with any CAS engine librefs that you create during your session. This enables you to use the CAS engine libref, Mycas, that is included in each of these examples and gives you easy access to the CAS environment.

**By Category**

- **Basic**
  - “Run a Simple DATA Step in CAS” on page 41
- **Table and Row Management**
  - “Convert a CAS Table to a SAS Data Set” on page 43
  - “Merge Tables Using the SET Statement” on page 55
  - “Order Rows in a Table Using ORDERBY=” on page 53
  - “Convert a CAS Table to a SAS Data Set” on page 43
  - “Remove Rows from a Table Using the WHERE Statement” on page 54
  - “Score an In-Memory CAS Table” on page 62
  - “Format BY Variables” on page 52
  - “Read a SAS 9 Data Set into SAS Viya” on page 56
- **Variables Management**
  - “Keep Variables in a CAS DATA Step” on page 49
  - “Order BY Variables in CAS” on page 50
  - “Check for Missing Values” on page 46
  - “Create a VARCHAR Variable Using the LENGTH Statement” on page 45
- **Session Management**
  - “Run a Simple DATA Step in CAS” on page 41
  - “View DATA Step Processing Information Using Automatic Variables” on page 59
Run a Simple DATA Step in CAS

Example

The following example shows how to run a DATA step in CAS. This DATA step processes in CAS because there is no input data set and the output data is in the form of a CAS table. Had there been an input table or data set, the input would have to have been a CAS table for the DATA step to run in CAS. Because there is no input data, the DATA step is executed in a single thread in the CAS server session. CAS tables require at least one variable, so an assignment statement is used to create a variable.

```
libname mycas cas;
data mycas.hello;
   put 'Hello from ' _hostname_ ' thread # ' _threadid_;
x=1;
run;
```

The output table is an in-memory CAS table that contains one row and one column with a value of 1. Here is the log output:

```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step has no input data set and will run in a single thread.
Hello from server01 thread #1
NOTE: The table hello in caslib CASUSER(****user) has 1 observations and 1 variables.
NOTE: DATA statement used (Total process time):
   real time           0.81 seconds
   cpu time            0.00 seconds
```

To run the DATA step in CAS using input data, the input data must be in the form of a CAS table. If the input data is a SAS data set, you must first load the data set into CAS and then run the DATA step on it. The CAS DATA step processes only in-memory CAS tables as input and output. In the following example, the “CASUTIL” in SAS Cloud Analytic Services: Language Reference is used to load a SAS data set into CAS. The SAS data set is named using the CAS engine libref, mycas.

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

proc casutil outcaslib="casuser";
   load data=sashelp.cars; /*2*/
run;

data mycas.cars2; /*3*/
   set mycas.cars; /*4*/
   if mpg_city > 25 then eff='Y';
   else eff='N';
   put 'Thread number: ' _threadid_ ' on worker node ' _hostname_;
run;
```

1 Create a CAS engine libref named MyCas.
Use PROC CASUTIL to load the SASHELP data set to CAS. This creates an in-memory CAS table named Cars. The OUTCASLIB= option loads the CAS table to the CASUSER caslib.

Specify a CAS table as the output data using the CAS engine libref.

Specify a CAS table as the input data using the CAS engine libref.

**Key Ideas**

- Running a DATA step in CAS means that the DATA step code and all the statements within it are executing in CAS in a CAS server session. For information about how the DATA step runs in CAS, see “Distributed Processing” on page 22.

- For a DATA step to run in CAS, the following must be true:
  - For a DATA step that has both input and output data, both the input and output data must be in the form of CAS tables that are named using a CAS engine libref: `data MYCAS.hello; set MYCAS.goodbye; run;`
  - For a DATA step that has no input table, the DATA step must output to a CAS table that contains at least 1 variable: `data mycas.hello; x=1; run;`
  - A DATA step must always have output in the form of a CAS table; specifying _NULL_ in the DATA statement as output causes the DATA step to process in SAS rather than in CAS: `data _NULL_; x=1; run;`

- It is recommended that you use the CASUTIL procedure to load your SAS data sets into CAS. For more information, see “CASUTIL” in *SAS Cloud Analytic Services: Language Reference* for more information.

- The DATA step automatically runs in CAS when you use CAS engine librefs on the input and output data sets, or, when you do not specify an input data set and specify output data using a CAS engine libref. For more information, see “Controlling DATA Step Processing” on page 23.

- Not all language elements are supported in a CAS DATA step. See “SAS Language Element Support for Processing in SAS and in CAS” on page 28 for information about what language elements are supported in the CAS DATA step.

**See Also**

- Chapter 1, “Accessing Data,” on page 1
- “CASUTIL” in *SAS Cloud Analytic Services: Language Reference*
- “SESSREF=cas-session-reference-name” in *SAS Viya Statements: Reference*
- “SAS Language Element Support for Processing in SAS and in CAS” on page 28
- “DATA Step Processing Modes” on page 23
Convert a CAS Table to a SAS Data Set

Example

The following example shows how to use the DATA step to convert an in-memory CAS table to a SAS data set. The first DATA step creates an in-memory CAS table that can be used for the conversion in this example. The second DATA step reads in the CAS table and outputs it as a SAS data set to the SAS WORK library.

libname mycas cas;  /*1*/
data mycas.earnings;  /*2*/
Amount=1000;  /*3*/
Rate=.075/12;
do month=1 to 12;
   Earned +{amount+earned}*(rate);
end;
run;
proc print data=mycas.earnings;run;

libname mysas "/u/myFiles/";  /*4*/
data mysas.earnings;
set mycas.earnings;
run;

1 Specify a LIBNAME statement to create a CAS engine libref for the CAS table.
2 Specify a CAS table as the output table. This DATA step runs in CAS.
3 Create variables for the table and calculate interest earned using a DO statement.
4 Create a SAS library to save the SAS data set into.
5 Read in the CAS table and output it as a SAS data set to the SAS library that you created.

<table>
<thead>
<tr>
<th>Obs</th>
<th>Amount</th>
<th>Rate</th>
<th>month</th>
<th>Earned</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1000</td>
<td>.00525</td>
<td>13</td>
<td>77.5928</td>
</tr>
</tbody>
</table>
Key Ideas

- When you specify a CAS engine libref on the output data set in a DATA step and you generate data from SAS statements rather than using in input data set as the data source, the DATA step runs in CAS. See “CAS server session processing” on page 23 for information about DATA step processing in CAS.
- The CAS DATA step (or, the DATA step running in CAS) works only on in-memory CAS tables. Therefore, input and output data must be in-memory CAS tables.
- Tables in CAS must contain at least 1 variable.
- The DATA step automatically runs in CAS when you use CAS engine librefs on the input and output data sets, or, when you do not specify an input data set and specify output data using a CAS engine libref. For more information, see “Controlling DATA Step Processing” on page 23.
- Not all language elements are supported in a CAS DATA step. See “SAS Language Element Support for Processing in SAS and in CAS” on page 28 for information about what language elements are supported in the CAS DATA step.

See Also

- “LIBNAME Statement” in SAS Viya Statements: Reference
- Chapter 1, “Accessing Data,” on page 4.
- “DATA Statement” in SAS Viya Statements: Reference
- “CAS server session processing” on page 23

Save a SAS Data Set to CAS

Example

You cannot use the DATA step to save a CAS table to disk on the CAS server. You must use the CASUTIL procedure for this. The following example uses data from the SASHELP.HEART data set to show how to load a SAS data set into CAS using PROC CASUTIL. The table is loaded as an in-memory CAS table. It then shows how use the DATA step in CAS to keep a limited number of variables from the original data set. Finally, it shows how to use PROC CASUTIL and the default caslib to save the in-memory CAS table to disk.

```sas
libname mycas cas;  /*3*/
proc casutil outcaslib="casuser" ;  /*3*/
   load data=sashelp.heart replace;  /*3*/
run;

data mycas.heart;
   set mycas.heart;
   keep status BP_status weight_status
         smoking_status chol_status deathcause;  /*3*/
run;
```
Create a VARCHAR Variable Using the LENGTH Statement

Example

In the following example, the CAS engine is used with the LENGTH statement to create a VARCHAR variable and a CHAR variable. The VARCHAR variable, X, has a length of 10 and the CHAR variable, Y, also has a length of 10.

```sas
libname mycas cas;
data mycas.string;
  length x varchar(10);
  length y $10;
  x = 'abc'; y = 'def';
run;
```
Key Ideas

- The VARCHAR data type is not supported by the SAS V9 engine. You must use the CAS engine when creating a VARCHAR in SAS Viya. When you use the CAS engine with the DATA step, specify the CAS libref on the output data set.

- The length of VARCHAR variables is determined based on the number of characters the string contains. The length of CHAR variables is determined based on the number of bytes the characters in the string requires.

- It is usually better to declare character strings as VARCHAR types than to declare them as CHAR types.

- If the data is consistently short, such as in an ID column, or two-letter state abbreviations, consider using a VARCHAR with a fixed overhead of 16 bytes.

- With smaller variables that consist of integers with up to three digits, such as in three-letter airport codes, VARCHAR variables can increase memory use.

See Also

- “DATA Types in SAS Viya” on page 32
- “SAS Cloud Analytic Services Data Types” in *SAS Cloud Analytic Services: Language Reference*

Check for Missing Values

Example

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

data mycas.project; /* 2 */
  input M_code $ prcode $ value startdate;
  format startdate mmdy.; /* 3 */
datalines;
  m4 pr-ITC 2258 92129
  m2 pr-ADV 3543 93423
  m5 . 3394 92312
  m4 pr-JEE 2230 93422
  m3 . 3000 92232
  m8 pr-LIR 3333 93132 ;

proc print data=mycas.project; /* 4 */
run;

data mycas.project2; /* 5 */
  set mycas.project;
  if prcode=' ';
```
1 Create a CAS engine libref to access data in CAS.
2 Create some sample data using the DATALINES and INPUT statements. This DATA step runs in the SAS client session and not in CAS. However, the DATA step uses the CAS engine to send the results to CAS in the form of an in-memory CAS table.
3 Apply a SAS date format to the Startdate variable. Run the DATA step in CAS using CAS engine librefs on both the input and output data sets. Use an IF statement to identify missing values for the PRCODE variable.
4 View the results.
5 Run the DATA step in CAS to check for missing values in the Prcode column.

**Key Ideas**

- The DATA step in SAS Viya must contain at least one variable.

**See Also**

- Chapter 1, “Accessing Data,” on page 4
- “IF Statement, Subsetting” in *SAS Viya Statements: Reference*
- “IF-THEN/ELSE Statement” in *SAS Viya Statements: Reference*

### Index CHAR and VARCHAR Character Strings

**Example**

In the following example, the LENGTH statement creates a VARCHAR variable and a CHAR variable. Both variables are defined with a length of 10. The variables contain identical values (a string, ‘abc’). The INDEX function returns an integer value that represents the position of the first character in a sub-string. As you might expect, the INDEX function returns a value of 2 for both data types. This is because the variables X and Y do not contain multi-byte characters.

```r
libname mycas.cas;

data mycas.string;
  length x varchar(10);
  length y $10;
  x = 'abc'; y = 'abc';
  xi = index(x,'b');
  yi = index(y,'b');
  put "xi = " xi;
  put "yi = " yi;
run;
```
However, in the next DATA step, the INDEX function is used with multi-byte Chinese characters in which each of the 3 characters takes 3 bytes. The statement \( xi = \text{index}(x, '样') \) tells SAS to return the position of the character ‘样’ in the 3-character string ‘榜样榜’. Even though the VARCHAR and CHAR variables contain identical string values, (‘榜样榜’), the INDEX function returns different values for each type. This is because the VARCHAR data type uses character length semantics and the CHAR data type uses byte length semantics.

```sas
data mycas.string2;
  length x varchar(10);
  length y $10;
  x = '榜样榜';  y = '榜样榜';
  xi = index(x, '样');
  yi = index(y, '样');
  put "xi = " xi;
  put "yi = " yi;
run;
```

Therefore, it is important to remember that the INDEX function returns the character position when it indexes VARCHAR data types:

```
<table>
<thead>
<tr>
<th>VARCHAR string: ‘榜样榜’</th>
</tr>
</thead>
<tbody>
<tr>
<td>character position:</td>
</tr>
<tr>
<td>1    2    3 (characters)</td>
</tr>
</tbody>
</table>
```

And it returns the byte position when it indexes CHAR data types.

```
<table>
<thead>
<tr>
<th>CHAR string: ‘榜样榜’</th>
</tr>
</thead>
<tbody>
<tr>
<td>byte position:</td>
</tr>
<tr>
<td>1-3  4-6  7-9 (bytes)</td>
</tr>
</tbody>
</table>
```
**Key Ideas**

- The VARCHAR data type is not supported by the SAS V9 engine. You must use the CAS engine when creating a VARCHAR in SAS Viya. When you use the CAS engine with the DATA step, specify the CAS libref on the output data set.

- VARCHAR uses character semantics. This means that the width of a column in a table is specified in character units rather than in bytes. Some characters, like the ones in this example, require more than one byte. VARCHAR variables are also varying length variables. This means that they use as many bytes as necessary to store up to 10 characters, even if some of those characters require 3 or 4 bytes to store them.

- VARCHAR variables can store lengths up to 40 bytes.

- VARCHAR variables can increase memory use when the column contains only 2- or 3-digit integer values, such as a three-letter airport.

**Note:** The INDEX function is a string function that uses character-based positional indexing when its parameters are VARCHAR type variables and it uses byte-based indexing when its parameters are CHAR type variables. The index function is useful for finding a substring in a parent string. For example, to find the position of the string "hat" in "abhatcd" you specify h = index(‘abhatcd’, ‘hat’); and the function returns h = 3, because the first letter of the “hat” string is the third from the left starting with 1.

**See Also**

- “Migrating Data to UTF-8” in *SAS Viya National Language Support (NLS): Reference Guide*
- “INDEX Function” in *SAS Viya Functions and CALL Routines: Reference*

---

**Keep Variables in a CAS DATA Step**

**Example**

In this example, the KEEP statement is used in a CAS DATA step to keep only a select set of variables in the Sashelp.Cars data set. The WHERE= data set option is used in the SET statement to filter data from the Cars data set.

```plaintext
libname mycas cas; /* 1 */
proc casutil outcaslib="casuser"; /* 2 */
   load data=sashelp.cars replace;
run;

data mycas.cars;
   set mycas.cars(where=(weight>6000)); /* 3 */
   drop keep make model type; /* 4 */
run;
```

1. Create a CAS engine libref to access data in CAS.
2 Load the Sashelp.cars data set into CAS.
3 Filter the data on the input data set.
4 Keep only three variables.

**Key Ideas**

- Both the DROP and KEEP statements are supported in a CAS DATA step.
- When the DATA step is running in CAS, the WHERE= data set option is supported only for the input data set. You cannot specify the WHERE= data set option on a DATA step that is processing in CAS.

**See Also**

- “DROP Statement” in *SAS Viya Statements: Reference*
- “KEEP Statement” in *SAS Viya Statements: Reference*
- “WHERE Statement” in *SAS Viya Statements: Reference*

## Order BY Variables in CAS

### Example

In this example, a transaction data set is created to show the difference between BY processing in SAS and BY processing in CAS. In SAS, the Base engine guarantees the ordering of rows as they are input but CAS does not guarantee this ordering. CAS does order the data by the first BY variable, but within that BY group, the ordering of rows is not guaranteed. When running the DATA step using BY variables in CAS, you must explicitly specify the second or third BY variable to ensure that order is preserved when you load a table from SAS to CAS.

In the example, the DATA step first creates a sample transaction data set in which the rows are ordered by Customer and within the Customer group, they are further ordered by Date. You will see how the DATA step in SAS preserves the ordering but the DATA step in CAS does not.

```sas
/* 1 */
libname mycas cas;
data transactions;
  format date date.;
customer="Jane"; date = '01FEB2016'd; amt=1; output;
customer="Jane"; date = '02FEB2016'd; amt=2; output;
customer="Jane"; date = '03FEB2016'd; amt=3; output;
customer="John"; date = '01FEB2016'd; amt=4; output;
customer="John"; date = '02FEB2016'd; amt=5; output;
customer="John"; date = '03FEB2016'd; amt=6; output;
run;
```

<table>
<thead>
<tr>
<th>Obs</th>
<th>Date</th>
<th>Customer</th>
<th>amt</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>01FEB10</td>
<td>Jane</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>02FEB10</td>
<td>Jane</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>03FEB10</td>
<td>Jane</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>01FEB10</td>
<td>John</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>02FEB10</td>
<td>John</td>
<td>5</td>
</tr>
<tr>
<td>6</td>
<td>03FEB10</td>
<td>John</td>
<td>6</td>
</tr>
</tbody>
</table>
Create a Transactions data set for this example.

Process the Transactions data set in SAS and print the results. The DATA step assumes ordering by DATE because the Base SAS engine returns rows in the order in which they were entered. You do not have to explicitly specify DATE as the second BY variable because the Base engine automatically preserves the ordering of the data. Copy the Transactions data set into a CAS table.

Process the data set in CAS and once again order by Customer. Notice that the FIRST variable now equals 'yes' in the second row of the BY group. The output shows that the ordering on the second variable, DATE, is not preserved in CAS like it is in the SAS DATA step. Even though the ordering of the Customer BY group is preserved, the ordering within the CUSTOMER group is random when the data is loaded into CAS. This is because the ordering of rows within BY groups is not guaranteed in CAS.

To solve the ordering problem, you must explicitly specify the order within the BY group by specifying a second BY variable.

**Key Ideas**

- The ordering of BY variables within a data set is not supported in CAS when loading the data set to a CAS table.
- To preserve the desired ordering, you must specify more BY variables.
Format BY Variables

Example

This example shows how the format of a variable can affect the number of threads used for processing in a CAS DATA step. A duplicate variable is created with a new format so that the BY processing can be done on the higher-cardinality format.

The first DATA step in the example sets up the data set for the example.

```
libname mycas cas; /*
 1 */
data air; /*
 2 */
  set sashelp.air(obs=20);
  format date qtr4.; /*
 3 */
run; proc print data=air; run;
data mycas.air; /*
 4 */
  set air(obs=20);
  newdate=date; /*
 5 */
  format newdate date10.; /*
 6 */
run; proc print data=mycas.air; run;
data mycas.air; /*
 7 */
  set mycas.air(obs=20);
  by newdate; /*
 8 */
run; proc print data=mycas.air; run;
```

1 Create a CAS engine libref for data access to CAS.
2 Create a new SAS data set, Air, from the Sashelp.Air data set.
3 Assign a low-cardinality format to the Date variable for the purposes of the example.
4 Read the data set, AIR, into a CAS table.
5 Duplicate the low-cardinality Date variable by setting it equal to a new variable named NewDate.
6 Change the new variable, NewDate, to a higher cardinality format using the FORMAT statement, and output the data set to an in-memory CAS table.
7 Run the DATA step in CAS and set the BY processing on the new, higher cardinality variable, NewDate. By setting the BY processing on the higher cardinality format, more threads are used and performance will improve.
Order Rows in a Table Using ORDERBY=

**Example**

In this example, a CAS table is loaded from a SAS data set into a CAS table using PROC CASUTIL. The DATA step runs in CAS to order the data in the table by the number of home runs using the ORDERBY= CAS data set option.

```plaintext
libname mycas cas; /*1*/
proc casutil incaslib="casuser" outcaslib="casuser"; /*2*/
  load data=sashelp.baseball replace;
run;

data mycas.baseball;
set mycas.baseball(orderby=(nHome)); /*3*/
  keep name nHits salary team;
run;
proc print data=mycas.baseball; run;
```

1  Create a CAS engine libref for data access to CAS.
2  Load the SASHELP.BASEBALL data set to a CAS table.
3  In a CAS DATA step, read in the data set and group the rows by the nHome variable.
You can also order the rows when you use PROC CASUTIL and the ORDERBY= option to load the data set to CAS:

```
proc casutil incaslib="casuser" outcaslib="casuser";
    load data=sashelp.baseball casout="baseball" replace orderby=(team name nHome);
run; proc print data=mycas.baseball;run;
```

**See Also**

- Chapter 1, “Accessing Data,” on page 1
- “ORDERBY= Data Set Option” in *SAS Cloud Analytic Services: Language Reference*

---

**Remove Rows from a Table Using the WHERE Statement**

**Example**

In the following example, the SAS data set Class is loaded into a CAS table using PROC CASUTIL and the rows are filtered using the WHERE statement.

```
libname mycas cas;

proc casutil outcaslib="casuser";
    load data=sashelp.class replace;
run; proc print data=mycas.class; run;

data mycas.class;
    set mycas.class;
    where sex="F" and age>13;
run;
proc print data=mycas.class; run;
```

**Note:** You cannot specify the WHERE= data set option as the output data in a CAS DATA step.

**See Also**

- “WHERE Statement” in *SAS Viya Statements: Reference*
- “WHERE= Data Set Option” in *SAS Viya Data Set Options: Reference*
- “Introduction” in *SAS Cloud Analytic Services: Fundamentals*
- “Client-Side Data Access” in *SAS Cloud Analytic Services: Fundamentals*
Merge Tables Using the SET Statement

Example

The following example creates two data sets and then loads them into CAS tables using PROC CASUTIL. The first CAS DATA step merges the two data sets by the variable, Common, and the second CAS DATA step merges the data by Common and Number.

```sas
libname mycas cas;  /*1*/
data animall;  /*2*/
input Common $ Animal $ 3-7 Number 8-10;
datalines;
a Ant    1
b Bird   2
c Cat    3
d Dog    4
e Eagle  5
f Frog   6
;
run;
data plant1;  /*3*/
input Common $ Plant $ 3-10 Number 12-13;
datalines;
a Grape   1
c Hazelnut 2
e Indigo  3
g Jicama  1
i Kale    2
k Lentil  3
;
run;
data merged;  /*4*/
   set animall plant1;
run;
proc print data=merged;
title 'Animals and Plants Merged in SAS'; run;

data merged;  /*5*/
   set animall plant1;
   by common ;
run;
proc print data=merged;
title 'Animals and Plants Merged By Common in SAS'; run;

data merged;  /*6*/
   set animall plant1;
   by common Number;
run;
proc print data=merged;
title 'Animals and Plants Merged By Common and Number in SAS'; run;

proc casutil outcaslib="casuser";  /*7*/
```
load data=animal1 replace;
load data=plant1 replace;
run;

data mycas.merged; /*8*/
   set mycas.animal1 mycas.plant1;
run;
proc print data=mycas.merged;
title 'Animal and Plant Merged in CAS'; run;

data mycas.merged2; /*9*/
   set mycas.animal1 mycas.plant1;
   by common;
run;
proc print data=mycas.merged2;
title 'Animal and Plant Merged by Common in CAS'; run;

data mycas.merged3; /*10*/
   set mycas.animal1 mycas.plant1;
   by common Number;
run;
proc print data=mycas.merged3; /*11*/
title 'Animal and Plant Merged by Common and Number in CAS'; run;

1 Create a CAS engine libref.
2 Create the first data set to merge.
3 Create the second data set to merge.
4 Merge the two data sets in a SAS DATA step.
5 Merge the two data sets by one variable.
6 Merge the two data sets by two variables.
7 Load the data set to CAS.
8 Merge the two data sets in a DATA step running in CAS.
9 Merge the two data sets by one variable in a CAS DATA step.
10 Merge the two data sets by two variables in a CAS DATA step
11 View and compare the results of the data processed in SAS and the data processed in CAS

---

**Read a SAS 9 Data Set into SAS Viya**

**Example**

The following example shows how to read a legacy SAS data set and convert the variables from a Latin-1 encoding to UTF-8. This process involves using the CVP engine to pad column widths and the COPY procedure to change the encoding scheme of the data file to UTF-8. The COPY procedure converts the encoding scheme on the input data set to the data representation of the output library.
libname mysasv9 cvp 'u/username/sasv9'; /* 1 */
libname mysas 'u/username/viya;

libname mylib CVP "path to data sets";
select class;
run;

data casuser.class; /* 2 */
   set mysasv9.class;
run;

proc contents data=casuser.class; /* 3 */
run;
proc casutil outcaslib="casuser"; /* 4 */
   save casdata="class";
run;

proc copy in=mysasv9 out=mysas noclone; /* 5 */
   select class;
run;

1 Use your existing SAS library location with the LIBNAME statement to create a CVP engine libref and a V9 engine libref. The CVP engine libref is for accessing data from previous versions of SAS and the V9 engine libref is for saving a copy of the data set to your local computer. A CAS engine libref, Casuser, was created for you when you signed on.

2 Read in the data set using the CVP LIBNAME engine and write it out to the CAS engine library. CVP creates a READONLY copy of the data. If you want to save a permanent copy of the data, you need to create a new data set.

3 View information about the data set and verify that the session encoding entry in the contents table is UTF-8.

4 The file is now loaded to an in-memory CAS session.

5 You must use PROC COPY with the NOCLONE option to convert the data set to UTF-8. Using the NOCLONE option results in a copy with the data representation of the output data library.
Key Ideas

- The CVP engine works only on CHAR variables. The CVP engine removes trailing blanks in the conversion.
- Format lengths are expressed as number of bytes. The format width needs to be bigger in order to correctly display multibyte characters.
- The length of the character column is interpreted as the number of bytes. If your data contains more than English letters, numbers, and standard punctuation, for example, then you need to plan for extra space in the column.
- SAS string functions, such as SUBSTR and INDEX assume that one character equals one byte. If multi-byte characters are present, those functions could potentially return incorrect answers and could result in corrupt data. SAS has a set of string functions which use character semantics and they are called K functions. These are safe to use on any character data.
- In general, any data sets created prior to SAS Viya should be converted to UTF-8. If your data includes any characters that require more than one byte in UTF-8, more space might be needed in the character column. The CVP LIBNAME engine is a read-only engine that adds padding to all character variables in a SAS data set.

Note: The CVP engine does not transcode or save a permanent copy of the data set. You must use PROC COPY with the NOCLONE option to create a permanent copy the data set.

See Also

- “LIBNAME Statement, CVP Engine” in SAS Viya Statements: Reference
- “SAS Viya Functions and CALL Routines: Reference” in SAS Viya Functions and CALL Routines: Reference
- “COPY” in SAS Viya Utility Procedures Guide
- “Copying an Entire Library” in SAS Viya Utility Procedures Guide
- “SAS DATASETS” in SAS Viya Utility Procedures Guide

Control Thread Count Using SINGLE=

Example

```sas
libname mycas cas;
proc casutil outcaslib="casuser"; /*
load data=sashelp.class;
run;

data mycas.class2 / sessref="casauto" single=no;
    set mycas.class;
```
View DATA Step Processing Information Using Automatic Variables

Example

libname mycas cas; /*

data mycas.cars;
set mycas.cars;
put "Table Row # " _N_; /*
put "Thread # " _threadid_; 
put "The # of Threads is " _nthreads_; 
put "The # of Nodes in this session, including the controller, is " _nranks_; 
put "The node (numbered 0 - n) that the DATA step is running on is " _rankid_; 
put "Worker node # " _hostname_; 
run;

1  Create a CAS engine libref for data access.
2  Use the PUT statement with an automatic variable to return information about the 
   DATA step processing in CAS.

Log 2.5  Partial Log Output

This is thread 49 running on host:server04
This is table Row # 33
The Thread ID number is # 49
The number of Threads is 64
The number of Nodes in this session, including the controller, is 5
The node (numbered 0, 1, and 2) that the DATA step is running on is 3 0

Run another DATA step in CAS and specify the SINGLE= option to limit the number of 
threads to 1. Compare the results.

   data mycas.cars / sessref="casauto" single=yes;
   set mycas.cars;
   put "The number of Threads is " _nthreads_; 
   run;
Key Ideas

- The DATA step automatically runs in multiple threads across partitioned data. By default, the DATA step runs in all available threads on every computer node in the cluster.
- When the DATA step runs in CAS, the same DATA step program is replicated across the CAS cluster (of computers), where each DATA step thread runs on a portion of the data.
- The DATA step automatically runs in CAS when you specify a CAS engine libref on both the input and output data sets.
- There are restrictions on what language elements you can use in a DATA step running in CAS. For information about these restrictions, see “Language Elements Supported in a CAS DATA Step” on page 28.

See Also

- “Automatic Variables” on page 37
- “DATA Statement” in SAS Viya Statements: Reference
- “Controlling DATA Step Processing” on page 23
- “About SAS Cloud Analytic Services Sessions” in SAS Cloud Analytic Services: Fundamentals

Use the Macro Facility to Generate CAS DATA Step Code

Example

The following example uses a SAS macro to generate SAS code. The macro takes the tables listed in the TABLES= parameter and writes them out as a SAS data set.

```
libname mycas cas;                                         /*1*/
data mycas.class;                                            /*2*/
   set sashelp.class; run;
data mycas.cars;
   set sashelp.cars; run;
data mycas.air;
   set sashelp.air; run;
%

%macro load(tables=);                                         /*3*/
libname mysas "*/r/ge.unx.sas.com/vol/vol620/u62/lidave/cas/*; /*4*/
%let n=%sysfunc(countw(&tables,%str( )));                    /*5*/
```
%do i=1 %to &n;                                               /*8*/
   %let dsn=%scan(&tables,&i);
   data mysas.&dsn;                                           /*7*/
   set mycas.&dsn;
   run;
%end;
%mend;                                                         /*8*/
%load(tables=class cars air);                                  /*8*/

1 Create a CAS engine libref to access the CAS tables.
2 Load three SASHELP data sets to CAS.
3 Start the macro definition.
4 Create a SAS libref.
5 Set \( n \) equal to the number of tables.
6 Loop through \( n \) times (for each table).
7 Create a SAS data set from each CAS table.
8 End macro definition.

Log 2.6 Partial Log Output for Macro

NOTE: Libref MYSAS was successfully assigned as follows:
   Engine:        V9
   Physical Name: /r/ge.unx.sas.com/vol/vol620/u62/lidave/cas

NOTE: There were 19 observations read from the data set MYCAS.CLASS.
NOTE: The data set MYCAS.CLASS has 19 observations and 5 variables.
NOTE: DATA statement used (Total process time):
   real time 0.02 seconds
   cpu time 0.01 seconds

NOTE: There were 428 observations read from the data set MYCAS.CARS.
NOTE: The data set MYCAS.CARS has 428 observations and 15 variables.
NOTE: DATA statement used (Total process time):
   real time 0.02 seconds
   cpu time 0.00 seconds

NOTE: There were 144 observations read from the data set MYCAS.AIR.
NOTE: The data set MYCAS.AIR has 144 observations and 2 variables.
NOTE: DATA statement used (Total process time):
   real time 0.02 seconds
   cpu time 0.01 seconds

See Also

- “Using the Macro Facility in SAS Viya” in SAS Viya Macro Language: Reference
Score an In-Memory CAS Table

**Example**

The following example loads a SASHELP data set into CAS and then scores the data using a conditional IF statement.

```sas
libname mycas cas;       /*1*/
proc casutil;            /*2*/
   load data=sashelp.baseball promote;
run;
data mycas.baseball;    /*3*/
   set mycas.baseball;
   if nHits > 150
      then score=1;  /*4*/
run;
```

1. Create a CAS engine libref to access the data in CAS.
2. Load the SASHELP.BASEBALL data set into CAS as an in-memory CAS table.
3. Run the DATA step in CAS on the table. Use the CAS engine libref (Mycas) that you created with the LIBNAME statement in Step 1 to name the input and output tables.
4. Execute the score code.

**Key Ideas**

- When you run the DATA step in the CAS server, you are running it in multiple threads on in-memory, partitioned data, which enables faster processing.
- To run the DATA step in a CAS server session, you must specify a CAS engine libref on both the input and output data sets and you must ensure that all the language elements you specify in the step are valid for CAS server processing. See “SAS Language Element Support for Processing in SAS and in CAS” on page 28 for information about language element support.
Chapter 3
Working with User-Defined Formats

Common Tasks for User-Defined Formats ........................................ 64
User-Defined Format Basics .......................................................... 65
Permanent Formats Supplied by SAS ............................................. 66
How Are Format Libraries Different than Format Catalogs? ............. 66
Add and Save a User-Defined Format ............................................. 66
  Example ................................................................. 66
  Results ........................................................................ 68
  Key Ideas ................................................................. 69
Add a Format Library from a File .................................................... 69
  Prerequisites ............................................................. 69
  Example ................................................................. 69
  Results ........................................................................ 70
List Format Libraries, Formats, and Search Path ............................... 71
  Example ................................................................. 71
  SAS Log ................................................................. 72
  Key Ideas ................................................................. 73
Add Locale-Specific Format Libraries ............................................. 73
  Example ................................................................. 73
  SAS Log ................................................................. 74
Add Multiple Format Catalogs to a CAS Session .............................. 75
  Example ................................................................. 75
Promote a Format Library ............................................................. 75
  Example ................................................................. 75
  SAS Log ................................................................. 76
  Key Ideas ................................................................. 77
# Common Tasks for User-Defined Formats

Table 3.1  Common Tasks for User-Defined Formats

<table>
<thead>
<tr>
<th>Task</th>
<th>Sample Syntax</th>
</tr>
</thead>
</table>
| Add a format to SAS and SAS Cloud Analytic Services.                 | proc format lib=work.formats \[
casfmtlib="fmtlib1"; \[
value fmtname \[
... \[
; run; |

| List your format search path.                                         | cas casauto listfmtsearch;                                                  |

| List the format libraries that are available.                        | cas casauto listformats;                                                    |

| Add format library to your search path.                             | cas casauto fmtsearch=(fmtlib1);                                            |

| List the format names in all format libraries.                       | cas casauto listformats members;                                            |

*Note*: You can limit the results to a single format library with the FMTLIBNAME= option.

| Copy existing formats to a SAS Cloud Analytic Services format library. | libname existing '/path/...'; \[
catname work.mycat(existing.format1 \[
existing.format2 ...}); \[
proc format library=work.mycat \[
cntlout=temp; run; \[
proc format cntlin=temp \[
casfmtlib='fmtlib1'; run; |

| Persist a format library as a SASDAT file.                          | cas casauto savefmtlib \[
fmtlibname=fmtlib1 \[
caslib=casuser \[
table=fmtlibfile; |

| Add a format library from a SASDAT file.                            | cas casauto addfmtlib \[
fmtlibname=fmtlib2 \[
caslib=casuser \[
table=fmtlibfile; |

*Tip*: By default, ADDFMTLIB appends the format library to the search path. You can specify POSITION=INSERT to add the format library to the beginning of the search path.

*Note*: Some sites have a large number of formats that should always be available. Administrators should use the cas.fmtcaslib= and cas.addfmtlib= server configuration file options to add those formats automatically when the server starts.
User-Defined Format Basics

SAS provides formats for controlling how variables are printed. You can use PROC FORMAT in your SAS programs to create user-defined formats. PROC FORMAT supports the creation of user-defined formats in catalogs on the SAS client and it supports adding formats and format libraries for a CAS session.

The basics for working with formats on the SAS client are described in the FORMAT procedure. The information in this section covers how to work with user-defined formats with SAS Cloud Analytic Services.

A format library has a name and can be temporary (session-scope) for the duration of your programming session. The alternative is that it can be set to have global scope and exist for as long as the server is running. You can persist a format library in a SASHDAT file and you can add formats to your CAS session from a persisted file.

Your administrator can configure the server so that the following occur when the server starts:

- format libraries are added and promoted from persisted format libraries (libraries that are persisted as SASHDAT files in a caslib's data source).
- during session start up for each connection, the format search path is updated to include the format libraries.

The following table identifies some of the ways that you might want to work with formats and summarizes methods to achieve the outcome.

<table>
<thead>
<tr>
<th>Desired Outcome</th>
<th>Method</th>
<th>Format Library Scope</th>
</tr>
</thead>
</table>
| Formats are available to your CAS session only. | • PROC FORMAT with the CASFMTLIB= option  
• CAS statement with the ADDFMTLIB option | Session scope |
| Formats are available to all CAS sessions connected to the server. Users must use FMTSEACH= in each session to add the format library to the search path. | • CAS statement with the ADDFMTLIB and PROMOTE options  
• CAS statement with the PROMOTEFMTLIB | Global scope |
| Formats are made available to all CAS sessions when the server starts. The format library is automatically added to each user’s search path. | • Add cas.fmtcaslib= and cas.addfmtlib= option in the server configuration file | Global scope |
Permanent Formats Supplied by SAS

In addition to user-defined formats, SAS provides formats that can be used with SAS Cloud Analytic Services. For a list of formats that are supported, see *SAS Viya Formats and Informats: Reference*.

How Are Format Libraries Different than Format Catalogs?

Format catalogs are accessed by the SAS client and are used for formatting values in procedure and DATA steps that run on the SAS client. This applies to procedures like PROC PRINT, PROC SQL, and others.

Format libraries are used within SAS Cloud Analytic Services. These are server-side formats that the server uses when an analysis is performed according to formatted values. For example, when a procedure or DATA step that runs in the server works on formatted values, the format must be available in a format library. This applies to procedures like PROC CARDINALITY, PROC BINNING, and others. These procedures are documented in *SAS Viya Data Mining and Machine Learning: Procedures Guide*.

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

See Also

- See [add multiple format catalogs to a CAS session](#) for an example of migrating your personal format catalogs to format libraries.
- See [add a format library from a file](#) to see how to work with a format library after you save one as a file.
- See "Migrate User-Defined Format Catalogs to Format Libraries" in the administration documentation if your site has format catalogs that are used widely by many programmers.

Add and Save a User-Defined Format

Example

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

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

; cas casauto savefmtlib fmtlibname=casformats
  table=enginefmt replace;

proc casutil;
  format enginesize enginesize;
  load data=sashelp.cars casout="cars" replace;
  contents casdata="cars";
quit;

libname mycas cas;

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

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

/* run the following CASUTIL code if you plan
 * to run the next example that adds a format
 * library from a file.
 */
proc casutil;
  save casdata="cars" casout="cars_formatted";
  droptable casdata="cars";
quit;

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

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

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

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

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

The following display shows the partial results for the CONTENTS statement in the CASUTIL procedure. The results show that the EngineSize column uses the Enginesize format.

Figure 3.1  Partial Results for the CONTENTS Statement of the CASUTIL procedure

<table>
<thead>
<tr>
<th>Column</th>
<th>Type</th>
<th>Length</th>
<th>Formatted Length</th>
<th>Format</th>
</tr>
</thead>
<tbody>
<tr>
<td>Make</td>
<td>char</td>
<td>13</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>Model</td>
<td>char</td>
<td>40</td>
<td>40</td>
<td></td>
</tr>
<tr>
<td>Type</td>
<td>char</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>Origin</td>
<td>char</td>
<td>6</td>
<td>6</td>
<td></td>
</tr>
<tr>
<td>DriveTrain</td>
<td>char</td>
<td>5</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>MSRP</td>
<td>double</td>
<td>8</td>
<td>8</td>
<td>DOLLAR</td>
</tr>
<tr>
<td>Invoice</td>
<td>double</td>
<td>8</td>
<td>8</td>
<td>DOLLAR</td>
</tr>
<tr>
<td>EngineSize</td>
<td>Engine Size (L)</td>
<td>8</td>
<td>15</td>
<td>ENGINESIZE</td>
</tr>
<tr>
<td>Cylinders</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Horsepower</td>
<td>double</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MPG_City</td>
<td>MPG (City)</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>MPG_Highway</td>
<td>MPG (Highway)</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Weight</td>
<td>Weight (LBS)</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Wheelbase</td>
<td>Wheelbase (IN)</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>Length</td>
<td>Length (IN)</td>
<td>8</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

The following display shows the first few columns from the Mpg_hwy_by_size table.

Figure 3.2  Selected Columns from the Mpg_Hwy_By_Size Table

<table>
<thead>
<tr>
<th>Obs</th>
<th>EngineSize</th>
<th>EngineSize_f</th>
<th><em>Column</em></th>
<th><em>Min</em></th>
<th><em>Max</em></th>
<th><em>NObs</em></th>
<th><em>NMiss</em></th>
<th><em>Mean</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Very large</td>
<td>Very large</td>
<td>MPG_Highway</td>
<td>20</td>
<td>20</td>
<td>1</td>
<td>0</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>Large</td>
<td>Large</td>
<td>MPG_Highway</td>
<td>12</td>
<td>25</td>
<td>15</td>
<td>0</td>
<td>18.8</td>
</tr>
<tr>
<td>3</td>
<td>Medium</td>
<td>Medium</td>
<td>MPG_Highway</td>
<td>14</td>
<td>28</td>
<td>80</td>
<td>0</td>
<td>21.925</td>
</tr>
<tr>
<td>4</td>
<td>Small</td>
<td>Small</td>
<td>MPG_Highway</td>
<td>17</td>
<td>32</td>
<td>173</td>
<td>0</td>
<td>25.7630573</td>
</tr>
<tr>
<td>5</td>
<td>Very economical</td>
<td>Very economical</td>
<td>MPG_Highway</td>
<td>21</td>
<td>66</td>
<td>159</td>
<td>0</td>
<td>31.314465409</td>
</tr>
</tbody>
</table>
Key Ideas

- Use PROC FORMAT with the CASFMTLIB= option to add formats to your CAS session automatically.
- You can save all the formats in a format library in a SASHDAT file with the CAS statement and the SAVEFMTLIB option. Only caslibs with path-based data sources can be used, such as PATH, DNFS, and HDFS.
- Two ways to use formats that are saved in a SASHDAT file are as follows:
  - Add them to individual programming sessions as shown in the “Add a Format Library from a File” on page 69 example.
  - An administrator can make the formats available as a global-scope format library for all sessions by setting the cas.fmtcaslib= and cas.addfmtlib= values in the server configuration file.

Add a Format Library from a File

Prerequisites

If you ran the code in the add a user-defined format example, then run the following code to remove the format from the SAS client and the CAS session.

```
proc datasets lib=work memtype=catalog kill;
quit;

cas casauto dropfmtlib fmtlibname=casformats;
```

Example

When you use the CAS statement with the SAVEFMTLIB option, you create a SASHDAT file that includes the formats in a format library. One requirement is that you specify a caslib with a data source type that supports saving files. The data source types are PATH, DNFS, and HDFS.

This example shows the reverse of that process, adding a format library from a SASHDAT file.

```
cas casauto addfmtlib fmtlibname=fmtlib1                               /* 1 */
caslib=casuser
   table=enginefmt;

proc casutil;                                                          /* 2 */
   contents casdata="cars_formatted.sashdat";
   load casdata="cars_formatted.sashdat"
     casout="cars_formatted";

libname mycas cas;

proc mdsummary data=mycas.cars_formatted;
```
The **CAS statement** is used with the ADDFMTLIB option to add a format library to the CAS session. The file, Enginefmt.sashdat, is specified in the TABLE= option. The file is relative to the path for the CASUSER caslib.

2 The **CONTENTS statement** specifies the on-disk name, Cars_formatted.sashdat. As a result, the column information shows the format, Enginesize, that was associated with the column when the table was saved.

3 Because the Enginesize format has five values and the **VAR** statement specifies two columns, the **MDSUMMARY procedure** creates an output table, Weight_len_by_size, with 10 rows.

The following display shows the results of the **PRINT** procedure. The rows are grouped by the two different values of _Column_.

<table>
<thead>
<tr>
<th>Obs</th>
<th>EngineSize</th>
<th>EngineSize_f</th>
<th><em>Column</em></th>
<th><em>Min</em></th>
<th><em>Max</em></th>
<th><em>NOb</em></th>
<th><em>NMiss</em></th>
<th><em>Mean</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>Very economi</td>
<td>Length</td>
<td>143</td>
<td>194</td>
<td>158</td>
<td>0</td>
<td>176.19496855</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
<td>Very large</td>
<td>Length</td>
<td>176</td>
<td>176</td>
<td>1</td>
<td>0</td>
<td>176</td>
</tr>
<tr>
<td>3</td>
<td>6</td>
<td>Large</td>
<td>Length</td>
<td>179</td>
<td>238</td>
<td>15</td>
<td>0</td>
<td>203.73333333</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>Medium</td>
<td>Length</td>
<td>178</td>
<td>230</td>
<td>80</td>
<td>0</td>
<td>198.325</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>Small</td>
<td>Length</td>
<td>150</td>
<td>219</td>
<td>173</td>
<td>0</td>
<td>188.7283237</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obs</th>
<th>EngineSize</th>
<th>EngineSize_f</th>
<th><em>Column</em></th>
<th><em>Min</em></th>
<th><em>Max</em></th>
<th><em>NOb</em></th>
<th><em>NMiss</em></th>
<th><em>Mean</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>8</td>
<td>Very large</td>
<td>Weight</td>
<td>3410</td>
<td>3410</td>
<td>1</td>
<td>0</td>
<td>3410</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>Very economi</td>
<td>Weight</td>
<td>1850</td>
<td>3003</td>
<td>150</td>
<td>0</td>
<td>2951.163522</td>
</tr>
<tr>
<td>8</td>
<td>6</td>
<td>Large</td>
<td>Weight</td>
<td>3246</td>
<td>7190</td>
<td>15</td>
<td>0</td>
<td>4993.4</td>
</tr>
<tr>
<td>9</td>
<td>3</td>
<td>Small</td>
<td>Weight</td>
<td>2811</td>
<td>5086</td>
<td>173</td>
<td>0</td>
<td>3652.6242775</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>Medium</td>
<td>Weight</td>
<td>3347</td>
<td>5969</td>
<td>80</td>
<td>0</td>
<td>4398.925</td>
</tr>
</tbody>
</table>
**List Format Libraries, Formats, and Search Path**

*Example*

```sas
proc format lib=work.formats casfmtlib="casformats"; /* 1 */
  value ynm
  1='yes'
  2='no'
  3='maybe';
run;

proc format lib=work.formats casfmtlib="fmtlib2"; /* 2 */
  value sml
  1='small'
  2='medium'
  3='large';
run;

cas casauto listformats; /* 3 */
cas casauto listfmtranges fmtname=ynm; /* 4 */
cas casauto dropfmtlib fmtlibname=casformats; /* 5 */
cas casauto listfmtsearch; /* 6 */
```

1. The **FORMAT procedure** is used to create a format that is named Ynm. The format is added to the Work.formats catalog on the SAS client. The format is also added to the CAS session in a format library that is named Casformats.

2. The FORMAT procedure is used again to create a format that is named Sml. The format is added to the Work.formats catalog on the SAS client. The format is also added to the CAS session in a format library that is named Fmtlib2.

3. The **LISTFORMATS option** is used to list the format libraries that are available to the CAS session. Both Casformats and Fmtlib2 are shown in the SAS log.

4. The **LISTFMTSRANGES option** is used with FMTNAME= to print a brief view of the format values for the Ynm format in the SAS log.

5. The DROPFMTLIB option is used to drop the Casformats format library. By default, the format library name remains in the format search path. You can specify the FMTSEARCHREMOVALDE option to remove the format library from the format search.

6. The **LISTFMTSEARCH option** is used to display the format libraries that are in the search path. In this example, because the Casformats library was dropped, but the FMTSEARCHREMOVALDE option was not used, the name remains in the search path. "Not Found" is used to indicate that Casformats is not available as a session-scope or global-scope format library.
SAS Log

NOTE: This session is in interactive mode.

1          proc format lib=work.formats casfmtlib="casformats";
NOTE: Both CAS-based formats and catalog-based formats will be written.
2            value ynm
3            1='yes'
4            2='no'
5            3='maybe';
NOTE: Format library CASFORMATS added. Format search update using parameter
APPEND completed.
NOTE: Format YNM has been output.
6          run;
7
8          proc format lib=work.formats casfmtlib="fmtlib2";
NOTE: Both CAS-based formats and catalog-based formats will be written.
9            value sml
10           1='small'
11           2='medium'
12           3='large';
NOTE: Format library FMTLIB2 added. Format search update using parameter APPEND
completed.
NOTE: Format SML has been output.
13         run;
14
15         cas casauto listformats;
NOTE: FmtLib = FMTLIB2
Scope = Session
Fmtsearch = YES
NOTE: FmtLib = CASFORMATS
Scope = Session
Fmtsearch = YES
16
17         cas casauto listfmtranges fmtname=ynm;
NOTE: Format Name = YNM
Range = 1=yes
NOTE: Format Name =
Range = 2=no
NOTE: Format Name =
Range = 3=maybe
NOTE: Request to LISTFMTRANGES YNM completed for session CASAUTO.
18
19         cas casauto dropfmtlib fmtlibname=casformats;
NOTE: Request to DROPFMTLIB CASFORMATS completed for session CASAUTO.
20
21         cas casauto listfmtsearch;
NOTE: FmtLibName = CASFORMATS
Scope = Not Found
NOTE: FmtLibName = FMTLIB2
Scope = Session
NOTE: Request to LISTFMTSEARCH completed for session CASAUTO.
Key Ideas

- When you use the FORMAT procedure with the CASFMTLIB= option, the format library and formats are automatically made available to your CAS session.
- You can use the LISTFMTSEARCH option for the CAS statement to display the format libraries that are in the search path.
- You can print a brief view of format values in the SAS log with the LISTFMTRANGES and FMTNAME= options. The format name that you specify must be in the search path.

Add Locale-Specific Format Libraries

Example

```sas
/* For each locale, copy the catalogs one at a time. */
options locale=en_US;
proc format locale library=work.formats casfmlib="casformats";
   value abc
       1='one'
       2='two'
       3='three';
run;

options locale=fr_FR;
proc format locale library=work.formats casfmlib="casformats";
   value abc
       1='un'
       2='deux'
       3='trois';
run;

/* Confirm that the format is available in each locale. */
cas casauto sessopts=(locale="en_US");
cas casauto listfmtranges fmtname=abc;

/* 1 */
cas casauto sessopts=(locale="fr_FR");
cas casauto listfmtranges fmtname=abc;

/* 2 */
cas casauto listformats members;

/* 3 */
```

1 The OPTIONS statement is used to set the locale for the SAS client to French. The subsequent PROC FORMAT step includes the LOCALE option that specifies to create a locale-specific format in the format catalog.

2 The SESSOPTS= option is used with the CAS statement to set the CAS session locale to United States English. The subsequent CAS statement prints a brief view of the format values in the SAS log.

3 The LISTFORMATS and MEMBERS options are used to list the format catalog, Casformats, and the format names, fr_fr-abc and en_us-abc, in the SAS log.
NOTE: This session is in interactive mode.

```sas
options locale=fr_fr;
proc format locale library=work.formats casfmtlib="casformats";

NOTE: Both CAS-based formats and catalog-based formats will be written.
value abc
  1='un'
  2='deux'
  3='trois';

NOTE: Format ABC has been written to WORK.FORMATS_fr_FR.
run;

options locale=en_us;
proc format locale library=work.formats casfmtlib="casformats";

NOTE: Both CAS-based formats and catalog-based formats will be written.
value abc
  1='one'
  2='two'
  3='three';

NOTE: Format ABC has been written to WORK.FORMATS_en_US.
run;

cas casauto sessopts=(locale="en_US");
NOTE: The CAS statement request to update one or more session options for session CASAUTO completed.

ca casauto listfmtranges fmtname=abc;
NOTE: Format Name = ABC
Range = 1=one
NOTE: Format Name =
Range = 2=two
NOTE: Format Name =
Range = 3=three
NOTE: Request to LISTFMTRANGES ABC completed for session CASAUTO.

cas casauto sessopts=(locale="fr_FR");
NOTE: The CAS statement request to update one or more session options for session CASAUTO completed.

ca casauto listfmtranges fmtname=abc;
NOTE: Format Name = ABC
Range = 1=un
NOTE: Format Name =
Range = 2=deux
NOTE: Format Name =
Range = 3=trois
NOTE: Request to LISTFMTRANGES ABC completed for session CASAUTO.

cas casauto listformats members;
NOTE: Fmtlib = CASFORMATS
Scope = Session
Fmtsearch = YES
Format = fr_fr-abc
Format = en_us-abc
NOTE: Request to LISTFORMAT completed for session CASAUTO.
```
Add Multiple Format Catalogs to a CAS Session

Example

The following example assumes that format catalogs are included in two locations, sasuser.formats and orion.mailfmt.

```plaintext
    catname work.mycat(sasuser.formats orion.mailfmt);       /* 1 */
    proc format library=work.mycat cntlout=temp;              /* 2 */
      run;
    proc format cntlin=temp casfmtlib='myfmtlib';             /* 3 */
      run;
    cas casauto savefmtlib fmtlibname=myfmtlib
      caslib=casuser table=myfmtlib;                          /* 4 */
```

1. The CATNAME statement is used to combine the two catalogs into the Work.Mycat catalog.
2. The FORMAT procedure uses the combined catalog as input and creates a SAS data set on the SAS client that stores information about the formats.
3. The second PROC FORMAT step reads the SAS data set and adds a format library in your session that is named Myfmtlib.
4. You are not required to save the format library as a SASHDAT file. However, you might choose to save them if the catalogs are large and impact performance during the second PROC FORMAT step.

Promote a Format Library

Example

Example Code 3.1  Add and Promote a Format Library

```plaintext
    proc format lib=work.formats casfmtlib="myfmtlib";
      value abc
        1 = 'yes'
        2 = 'no'
        3 = 'maybe'
      ;
      run;

    cas casauto promotefmtlib fmtlibname=myfmtlib replace;       /* 2 */
```

75
When you promote a format library, other users’ sessions do not automatically add the format library to the format search path. You can reproduce another users’ experience by opening a new tab in SAS Studio.

**Example Code 3.2  Add a Promoted Format Library to the Search Path**

```sas
/* from another tab */
cas casauto listformats; /* 3 */

cas casauto fmtsearch=(myfmtlib) position=insert;

cas casauto listformats; /* 4 */
```

1. The FORMAT procedure is used with the CASFMTLIB= option to add a format library named Myfmtlib to your CAS session. The format library is added with session scope.

2. The PROMOTEFMTLIB option is used to promote the format library to global scope. Other sessions that connect to the same server can access the Myfmtlib format library, but it is not automatically added to their format path.

3. The first use of the LISTFORMATS option shows that the Myfmtlib format library is available to the session, that it is not in the format search path.

4. After the FMTSEARCH= option is used to add the format library to the search path, the second use of LISTFORMATS shows that the format library has global scope and is

**SAS Log**

```
NOTE: This session is in interactive mode.
1 proc format lib=work.formats casfmtlib="myfmtlib";
NOTE: Both CAS-based formats and catalog-based formats will be written.
2 value abc
3 1 = 'yes'
4 2 = 'no'
5 3 = 'maybe'
6 ;
NOTE: Format ABC has been output.
7 run;
8 cas casauto promotefmtlib fmtlibname=myfmtlib replace;
NOTE: Request to PROMOTEFMTLIB MYFMTLIB completed for session CASAUTO.
```
The SAS log for any session that runs the code to add the promoted format library to the search path is similar to the following.

```
1          /* from another tab */
2          cas casauto listformats;
NOTE: Fmtlib = MYFMTLIB
        Scope = Global
        Fmtsearch =
NOTE: Request to LISTFORMAT completed for session CASAUTO.
3          cas casauto fmtsearch=(myfmtlib) position=insert;
NOTE: Request to FMTSEARCH completed for session CASAUTO.
4          cas casauto listformats;
NOTE: Fmtlib = MYFMTLIB
        Scope = Session
        Fmtsearch = YES
NOTE: Fmtlib = MYFMTLIB
        Scope = Global
        Fmtsearch =
NOTE: Request to LISTFORMAT completed for session CASAUTO.
```

**Key Ideas**

- When you promote a format library from a programming session, the format library is available to other sessions. However, it is not automatically added to the search path for the other sessions.
- You can use the FMTSEARCH= option to add a format library to your search path.
- For frequently used formats that you want to be available for all programming sessions, you can save the format library as a SASHDAT file. Afterward, an administrator can modify the server configuration file to add the format library from the file when the server starts.
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.