SAS® Cloud Analytic Services 3.4: DATA Step Programming

SAS® Documentation
April 30, 2019

The correct bibliographic citation for this manual is as follows: SAS Institute Inc. 2018. *SAS® Cloud Analytic Services 3.4: DATA Step Programming*. Cary, NC: SAS Institute Inc.

SAS® Cloud Analytic Services 3.4: DATA Step Programming

Copyright © 2018, SAS Institute Inc., Cary, NC, USA

All Rights Reserved. Produced in the United States of America.

For a hard copy book: No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, electronic, mechanical, photocopying, or otherwise, without the prior written permission of the publisher, SAS Institute Inc.

For a web download or e-book: Your use of this publication shall be governed by the terms established by the vendor at the time you acquire this publication.

The scanning, uploading, and distribution of this book via the Internet or any other means without the permission of the publisher is illegal and punishable by law. Please purchase only authorized electronic editions and do not participate in or encourage electronic piracy of copyrighted materials. Your support of others' rights is appreciated.

U.S. Government License Rights; Restricted Rights: The Software and its documentation is commercial computer software developed at private expense and is provided with RESTRICTED RIGHTS to the United States Government. Use, duplication, or disclosure of the Software by the United States Government is subject to the license terms of this Agreement pursuant to, as applicable, FAR 12.212, DFAR 227.7202-1(a), DFAR 227.7202-3(a), and DFAR 227.7202-4, and, to the extent required under U.S. federal law, the minimum restricted rights as set out in FAR 52.227-19 (DEC 2007). If FAR 52.227-19 is applicable, this provision serves as notice under clause (c) thereof and no other notice is required to be affixed to the Software or documentation. The Government's rights in Software and documentation shall be only those set forth in this Agreement.

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

May 2019

SAS® and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration.

Other brand and product names are trademarks of their respective companies.

3.4-P3:casdspgm

Contents

Chapter 1

DATA Step Basics

Common Tasks

The following table contains simplified code examples for performing common DATA step tasks. For more in-depth and detailed examples, see Chapter 2, “DATA Step Examples,” on page 31.

For information about how to run the examples, see Set Up Code For Examples on page 32.

Run the DATA step in CAS	`cas casauto;`
	`libname CAS-engine-libref cas;`
	`data CAS-engine-libref.table-name;`
	`  put 'Hello from ' _hostname_`
	`  'thread # ' _threadid_;`
	`  x=1;`
	`run;`

Load a SAS data set to CAS	`cas casauto;`
	`libname SAS-engine-libref "/file-path/";`
	`libname CAS-engine-libref cas;`
	`data CAS-engine-libref.table-name;`
	`  set SAS-engine-libref.data-set-name;`
	`run;`

Run the DATA step on a table in CAS	`cas casauto;`
	`libname CAS-engine-libref cas;`
	`data CAS-engine-libref.table-name;`
	`  set CAS-engine-libref.table-name;`
	`  <more-statements>;`
	`run;`

Create CHAR and VARCHAR variables	`cas casauto;`
	`libname CAS-engine-libref cas;`
	`data CAS-engine-libref.table-name;`
	`  length variable-name varchar(n);`
	`  length variable-name $ n;`
	`run;`

Promote a CAS table	`cas casauto;`
	`libname CAS-engine-libref cas;`
	`data CAS-engine-libref.table-name(copies=1 promote=yes);`
	`  set CAS-engine-libref.table-name;`
	`run;`

Merge CAS tables	`cas casauto;`
	`libname CAS-engine-libref cas;`
	`data CAS-engine-libref.table-name;`
	`  merge CAS-engine-libref.table-1 CAS-engine-libref.table-2;`
	`  by variable-name;`
	`run;`

The DATA Step and CAS

Role of the DATA Step with CAS

The DATA step can run in both Cloud Analytic Services (CAS) and in SAS. CAS is the cloud-based run-time environment that enables multithreaded, parallel DATA step execution. To understand more about what it means for the DATA step to run in CAS, see "Multithreaded Processing on Distributed Data" on page 8.

How you run the DATA step in CAS is almost identical to how you run it in SAS with the traditional DATA step. The DATA step in CAS supports most of the same language elements and provides much of the same functionality.

The DATA step serves two roles in the CAS environment:

- **Loading a SAS data set to CAS** – processing is done in SAS in a single thread in the SAS client session. A CAS engine libref is used to load the data after it has been processed in the SAS DATA step.

```
libname mycas cas;
data mycas.cars;
   set sashelp.cars;
run;
```

- **Processing in CAS** – processing is done in CAS in multiple threads. Processing is done on an in-memory CAS table. Note that this is a separate process from loading the data and can be done only in a separate DATA step on data already loaded to CAS.

```
libname mycas cas;
data mycas.cars2;
   set mycas.cars;
run;
```

For information about how to run the examples, see Set Up Code For Examples on page 32.

You can use the CASUTIL procedure to load Excel, .csv, .txt, and other types of data to CAS. See "CASUTIL Procedure" in *SAS Cloud Analytic Services: User's Guide* for more information. For an example of loading a .csv file to CAS, see "Load a CSV File into CAS" in *SAS Cloud Analytic Services: User's Guide*.

Why Run the DATA Step in CAS?

Big data running in a single thread is slow. The DATA step runs faster in CAS because it can run in multiple threads, in parallel. In CAS, hundreds of threads are available, across several machines. When you program your DATA step to run in CAS, all of those threads become available for you to use to improve the performance with massively parallel processing. Parallel processing speeds the execution of big data by starting the DATA step on multiple machines and dividing processing workload among threads on these machines.

Running the DATA Step in CAS

Requirements

To run the DATA step in CAS on page 8, you must perform the following steps:

Table 1.1 *Requirements for Running the DATA Step in CAS*

Requirement	Sample Code

1. Start a CAS session.
2. Create a CAS engine libref.
3. Specify CAS engine librefs on both the input and output tables.
4. Use only CAS-supported language elements.
5. Ensure that there is at least one variable in the output table (CAS does not support zero-column tables).
	`cas casauto;  /* CAS session  */`
	`libname mycas cas;  /* CAS engine libref  */`
	`data mycas.random;  /* CAS output table  */`
	`x=rand('UNIFORM');  /* CAS-supported function */`
	`run;`

Note: Librefs in the CAS DATA step must be associated with the same CAS session.

Recommendations

Best Practices for Running the DATA Step in CAS

The following are recommended best practices for running the DATA step in CAS. To understand what it means for the DATA step to run in CAS, see "Multithreaded Processing on Distributed Data" on page 8.

- Check the log after the program runs to verify where the DATA step processed. The message "**`Running DATA step in Cloud Analytic Services`**" always appears in the log when the DATA step runs in CAS.
- Specify the MSGLEVEL=i system option to get more detailed information about where the DATA step ran.

- Specify the SESSREF= data set option if you want to ensure that the DATA step always runs in CAS.

For information about the language elements that are supported in CAS, see "Language Elements Support" on page 17.

DATA Step That Runs in SAS with an Unsupported Language Element

If all of the requirements for processing in CAS have *not* been met, then the DATA step does not necessarily return error codes or stop executing. For example, if an unsupported language element is specified in the DATA step, but all other requirements have been met, then the DATA step defaults to running in SAS.

For example, this DATA step uses the RANUNI function, which is not supported for processing in CAS:

```
cas casauto;
libname mycas cas;

data mycas.randomU;
   x=ranuni(104);
run;
```

And, as the following log output indicates, the DATA step ran successfully, but it did *not* run in CAS:

```
NOTE: The data set MYCAS.RANDOMU has 1 observations and 1 variables.
```

DATA Step That Runs in CAS with a Supported Language Element

This DATA step uses the RAND function, which is supported for processing in CAS:

```
cas casauto;
libname mycas cas;

data mycas.random;
   x=rand('UNIFORM');
run;
```

And, as the following log output indicates, the DATA step ran successfully in CAS:

```
Running DATA step in Cloud Analytic Services.
NOTE: The DATA step has no input data set and will run in a single thread.
NOTE: The table random in caslib CASUSER(sasdemo) has 1 observations and 1
variables.
```

DATA Step with the MSGLEVEL=i System Option

You can specify **i** in the MSGLEVEL= system option to get more detailed information in the log about where the DATA step runs.

```
options msglevel=i;

cas casauto;
libname mycas cas;

data mycas.randomU;
   x=ranuni(104); /* function is not supported for processing in CAS */
```

```
run;

data mycas.test;
   x = SYMGET('SYSDATE9'); /* macro is not supported for processing in CAS */
   put x;
run;
```

The following log output shows the extra information that is printed to the log when MSGLEVEL=i is specified:

```
NOTE: Could not execute DATA step code in Cloud Analytic Services. Running DATA
step in the SAS client.
NOTE: The data set MYCAS.RANDOMU has 1 observations and 1 variables.
NOTE: Could not execute DATA step code in Cloud Analytic Services. Running DATA
step in the SAS client.
28MAR2019
NOTE: The data set MYCAS.TEST has 1 observations and 1 variables
```

DATA Step with the SESSREF= Option

For DATA steps that you intend to always run in CAS, specify the SESSREF= option in the DATA statement to explicitly run the DATA step in CAS.

In this way, the DATA step either runs in CAS because all the requirements have been met, or it does not process at all. It also displays an error in the log. When you add this option in the DATA statement, you can be sure of where the DATA step is processing at all times. The following example uses the same unsupported DATA step function from the previous example but this time with the SESSREF= option specified in the DATA statement:

Example Code 1 *DATA Step That Uses SESSREF= to Stop Processing When it Cannot Run in CAS*

```
options msglevel=n;

cas casauto;
libname mycas cas;

data mycas.rand / sessref=casauto;
   x=ranuni(104);
run;
```

In this case, DATA step processing was stopped because the SESSREF= option forces the program to run in CAS rather than falling back to the SAS client for local processing. The following log output shows the error message:

```
NOTE: Running DATA step in Cloud Analytic Services.
ERROR: The function RANUNI is unknown, or cannot be accessed.
ERROR: The action stopped due to errors.
NOTE: The SAS System stopped processing this step because of errors.
```

Determining Where the DATA Step Is Running

The message "**`Running DATA step in Cloud Analytic Services`**" appears in the SAS log output when the DATA step runs in CAS.

Figure 1.1 *Log Output for a DATA Step That Runs in CAS*

```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
```

Where the DATA step runs is driven by where your tables are stored:

- If your data sets are stored in SAS, then the DATA step runs in SAS.
- If your input and output tables are stored in CAS, then the DATA step runs in CAS.
- If your tables are stored in a mix of locations, then the DATA step runs in SAS.

Note: If you do not see the message **`Running DATA step in Cloud Analytic Services`** in the SAS log, then the DATA step did not run in CAS. See "Best Practices for Running the DATA Step in CAS" on page 4 and "Requirements" on page 4 for more information.

Controlling Where the DATA Step Runs by Default

You can use the DSCAS | NODSCAS system option to control whether the DATA step runs in CAS. The default behavior is for the DATA step to run in CAS. This means that the DATA step first tries to run in CAS, and if it fails, it runs in SAS.

The DATA step fails to run in CAS if any of the requirements for running in CAS are not met. See "Running the DATA Step in CAS" on page 4 for a list of these requirements.

If you want the DATA step to run in SAS only, without first trying to run in CAS, you can specify the NODSCAS system option:

```
options nodscas;

data mycas.test;     /* This DATA step runs in SAS */
   set mycas.air;
run;
```

Knowing and controlling where the DATA step is running is important for the following reasons:

- performance can be enhanced or degraded depending on the size of the table and where you run the DATA step. Running the DATA step in multiple threads in CAS is recommended for large tables.
- programs that rely on values retained from one row to the next (calculations that have inter-row dependencies) behave differently depending on where the DATA step runs. See "Inter-row Dependencies" on page 11.
- grouping and ordering of data is handled differently in the multithreaded CAS DATA step. Because each thread operates independently on a portion of the table, there is no global ordering of the table. See "BY-Group Processing" on page 14.

Processing Modes

Single versus Multithreaded Processing

Single-threaded
the DATA step runs in SAS or in CAS in a single thread on an entire, non-distributed table. Running the DATA step in a single thread is not recommended for very large tables.

Multithreaded (default)
the DATA step runs in CAS in multiple threads in parallel. Each DATA step thread operates on a portion of the table. The DATA step can run in multiple threads in a single-machine (non-distributed) server or in a distributed server.

Distributed versus Non-Distributed Data

Distributed server
a multi-node server environment in which CAS distributes the rows of a table across multiple physical machines. DATA step processing is distributed across the nodes so that the processing power of multiple CPUs running in parallel is combined and runs on the partitioned table. This is ideal for very large tables that require a large amount of processing power.

Single-Machine server
a single-machine server in a non-distributed environment that is configured to run as the server. When you run the DATA step in multiple threads in a single-machine server, the processing is performed on each CPU core on that machine. This enables multithreaded, parallel DATA step execution.

For more information about CAS server architecture, see *SAS Cloud Analytic Services: Fundamentals*.

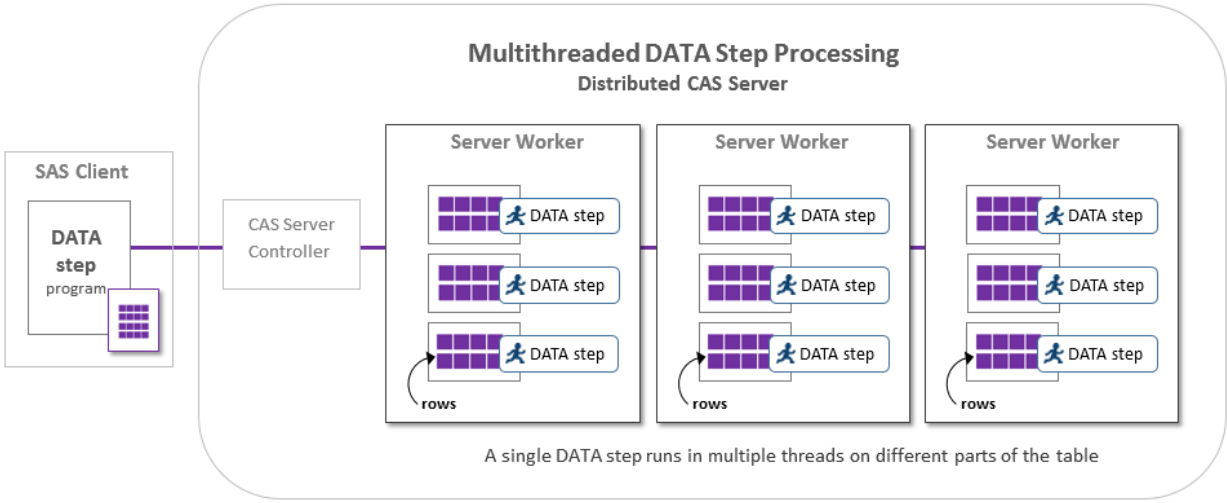
Scenarios

Scenarios for DATA step processing in CAS include the following:

- "Multithreaded Processing on Distributed Data"
- "Multithreaded Processing on a Single Machine "
- "Single-threaded Processing"

Multithreaded Processing on Distributed Data

A multithreaded DATA step is one in which the processing workload is shared among multiple, identical DATA steps that run concurrently on different parts of the table. In a distributed data environment, the table schema is duplicated on every node and each node holds different rows of the table.



In a distributed CAS server environment, DATA step processing is distributed across worker nodes so that the processing power of multiple CPUs running in parallel is combined. When the DATA step runs in a distributed server, it runs in multiple threads on each node. On each node, CAS divides the table into groups or blocks of data, and the DATA step runs on each group, allocating one DATA step thread per group. This is ideal for very large tables that require a large amount of processing power.

The DATA step runs by default in multiple threads on every available node.

Note: Because the rows of a table are distributed across multiple machines and partitioned on each node in this way, each DATA step thread processes only a portion of the table. This separation of table rows might affect how some functions that have inter-row dependencies behave. See "Inter-row Dependencies" on page 11 for more information.

Multithreaded Processing on a Single Machine

In non-distributed CAS environments, a single-machine is configured to run as the CAS server. The single-machine contains multiple CPU cores to support multithreaded, parallel processing. When the DATA step runs in multiple threads on a single-machine server, the entire table is loaded onto this single machine. CAS partitions the table based on the number of cores available and each DATA step thread runs on each partition.

Splitting the table among threads in this way speeds up processing. Instead of having one DATA step thread work on the entire table, there are many DATA steps working on the table in parallel.

Single-threaded Processing

Single-threaded DATA step processing occurs when the DATA step runs in a single thread on an entire, non-distributed, non-partitioned table. You can run the DATA step in a single thread, even when there are multiple CAS servers available. When you run the DATA step in a single thread in a distributed environment, processing is localized to a single machine regardless of the number of available worker nodes.

To specify that the DATA step runs on a single thread, specify the SINGLE= option in the DATA statement. For an example, see "Run the DATA Step in a Single Thread" on page 69.

Running the DATA step in a single thread is not recommended for very large tables. Single-threaded processing might be useful when you need to compute sums across multiple rows. See "Sum a Variable across an Entire Table" on page 44 for an example that shows how single-threaded DATA step processing might be needed.

CAUTION:

If a single-threaded DATA step processes large tables, not only does the program run slowly, the machine executing the single thread can run out of disk or memory resources.

Controlling Threaded Processing

Default Processing

A DATA step that runs in CAS, runs in all available threads by default. If the DATA step has no input table, then it runs in a single thread.

Example Code 2 *The DATA Step Runs in Multiple Threads by Default When There Is an Input Table*

```
data mycas.test;
   set mycas.test;
   put 'The DATA step is running in ' _nthreads_ 'threads';
run;
```

Output 1.1 *Log Output*

```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step will run in multiple threads.
The DATA step is running in 192 threads
```

Example Code 3 *A DATA Step with No Input CAS Table Runs in a Single Thread*

```
data mycas.test;
   put 'The DATA step is running in ' _nthreads_ 'thread';
run;
```

```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step has no input data set and will run in a single thread.
The DATA step is running in 1 thread
```

The number of available threads depends on the number of active server nodes and CPU cores available on each node. The DATA step typically runs one DATA step thread per core.

For example, if your CAS configuration consists of four worker nodes and each node supports 32 threads (or cores), then the DATA step will run in 128 threads. The number of DATA step threads equals the number of nodes multiplied by the number of threads running on each node.

For information about how to run the examples, see Set Up Code For Examples on page 32.

Controlling the Number of Threads

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

- SINGLE=YES specifies that the DATA step runs in a single thread.

```
data mycas.class / single=yes;
   set mycas.class;
run;
```

- SINGLE=NO specifies that the DATA step runs in all available threads (this is the default behavior):

```
data mycas.class / single=no;
   set mycas.class;
run;
```

- SINGLE=NOINPUT causes a DATA step without input tables to run in a single thread instead of in all available threads. This is the default value for a DATA step with no input tables when you do not specify the SESSREF= option.
- THREADS= specifies the number of threads that the DATA step runs in on each node. A value of zero means that the DATA step will run in the maximum number of threads allowed. This option is for fine tuning and it might show different results depending on the number of CPU cores on each node.

Inter-row Dependencies

A multithreaded DATA step can reduce run time and improve performance. But functions or statements that rely on information from one row to another might return unexpected results when they are used in a multithreaded DATA step.

For example, the RETAIN statement has inter-row dependencies because it retains a value from one row to the next. But, in a multithreaded DATA step, the rows in a table are not necessarily processed together in the same DATA step thread, nor is their order guaranteed.

Therefore, the RETAIN statement can no longer operate on the entire table, which is needed for this type of operation.

Here are some SAS language elements that have inter-row dependent operations:

- RETAIN statement
- DIF and LAG functions
- .FIRST and .LAST automatic DATA step variables (see "How SAS Identifies the Beginning and End of a BY Group" in *SAS DATA Step Statements: Reference*)
- "Sum Statement" in *SAS DATA Step Statements: Reference*
- Temporary arrays

The calculations performed by these types of statements are often referred to as *full-table operations*. Full-table operations require reading the entire contents of a table as a whole, rather than just reading selected portions.

Full-table operations can be performed in two steps: the first step involves performing a local operation, such as computing a partial sum. The second step involves performing a global operation, such as computing the final sum from a series of partial sums.

This two-step operation works well because it uses the processing power of all the threads to process the entire table and a single thread to process the smaller table subset.

For an example of this type of program, see Sum a Variable Across an Entire Table. on page 44

For more information about temporary arrays, see ARRAY Statement.

Ordering in CAS

In SAS, the order of variables in a data set is based on the order in which the variables are listed in the input data set. This implicit ordering of data by the DATA step does not occur in CAS. In CAS, there is no guarantee of ordering of data.

For example, suppose that you have the following SAS data set:

var1	var2
1	N
1	Y
1	Y
2	Y
2	Y
2	N

This data set is then read in by the following DATA step:

Example Code 4 *DATA Step in SAS*

```
data out;
  set in;
  by var1;
  if first.var1 and var2 = 'N';
run;
```

The DATA step reads the rows as they are listed in the input. Here are the results for the Example Code 1.4 on page 12 shown above:

var1	var2
1	N

Notice that there is one row of output that meets the conditions where **`first.var1=1`** and **`var2='N'`**.

When you read in this data set using the DATA step in CAS, CAS also groups the rows based on **`var1`**. But, the order of the rows within the BY group can vary. This variance causes the DATA step to produce different results each time that it is executed.

For example, the CAS DATA step might read the rows in this order:

var1	var2
1	Y
1	N
1	Y
2	N
2	Y
2	Y

And, this produces different results than SAS. With this order, there is one output row in which **var1=2** and **var2='N'**. Here are the results for the same DATA step in CAS:

var1	var2
2	N

One way to ensure stable ordering of data in this case is to add a sequence number to the rows as they are loaded into CAS.

Example Code 5 *DATA Step in SAS (Loads Data to CAS)*

```
data cas.mydata;
  set sas.mydata;
  seqno = _N_;
run;
```

This DATA step produces the following table in CAS:

var1	var2	seqno
1	Y	1
1	N	2
1	Y	3
2	N	4
2	Y	5
2	Y	6

You then specify the sequence number in the BY statement. When you do this, the DATA step groups the rows by **var1**, and then it orders the rows, first by **var1**, and then by **seqno**. This causes the rows to be read in the same order in CAS as in SAS.

Example Code 6 *DATA Step in CAS*

```
data cas.out;
  set cas.mydata;
  by var1 seqno;
  if first.var1 and var2 = 'N';
run;
```

To see an example of how you can add a unique sequence number to rows in a CAS table, see "Add a Unique Row Identifier" on page 45.

BY-Group Processing

How CAS Groups Data with BY Variables

When the DATA step runs in a distributed server, CAS distributes the input table across multiple servers. Each server gets only a portion of the rows in the table.

The BY statement in a SAS DATA step divides table data into groups of rows that share the same values of the BY variables. When you use the BY statement in a distributed server, CAS groups the rows based on the first BY variable. When CAS distributes the table across multiple nodes, it keeps the BY groups intact. That is, rows that share the same BY variables are stored together on a single machine.

Here are some key points to remember about BY-group processing in the CAS DATA step:

- CAS groups and distributes table rows based on the values of the first BY variable, then orders the rows within each BY group based on all the BY variables.
- On the CAS server, there is no guarantee of global ordering between BY groups. Each DATA step thread can group and order only the rows that are in the same data partition.
- If a format is applied to the first BY variable, then CAS creates BY groups based on the *formatted* values of the first BY variable (rather than the raw values). If there is not a format applied to the first BY variable, CAS creates BY groups based on the raw values.
- The DATA step orders data within each BY group based on the raw values of all the BY variables. When a BY group is defined, the DATA step orders only the rows within that BY group.
- The DATA step runs in multiple threads on each node, one thread per BY group.
- Formats can have a noticeable impact on performance when they are used with BY variables. This is because CAS groups and distributes rows based on the format of the first BY variable. If a high cardinality format is applied to the first BY variable, then more threads will be used. Conversely, if a low cardinality format is applied to the first BY variable, then fewer threads will be used.

  See the example in "How BY Variables Affect Multithreaded DATA Step Execution" on page 15 for an illustration of low and high-cardinality BY-groups.

For information about BY-Group processing in SAS, see "BY-Group Processing in the DATA Step" in *SAS Language Reference: Concepts*.

Identifying the First and Last Rows in BY-Groups

In the DATA step, SAS identifies the beginning and end of each BY group by creating two temporary variables. One temporary variable is created for the first row of the BY group and another temporary variable is created for the last row of the BY group. CAS creates a FIRST. and LAST. variable for every node. These variables can be used to manipulate how data is grouped, subdivided, and ordered.

Here are two examples that use the .FIRST variable to group rows in a table:

- "Group and Order Rows Using the DATA Step BY Statement" on page 64

- "Group Data by Two Variables" on page 66

For information about how the DATA step identifies BY groups and how you can manipulate them, see FIRST and LAST DATA Step Variables.

How BY Variables Affect Multithreaded DATA Step Execution

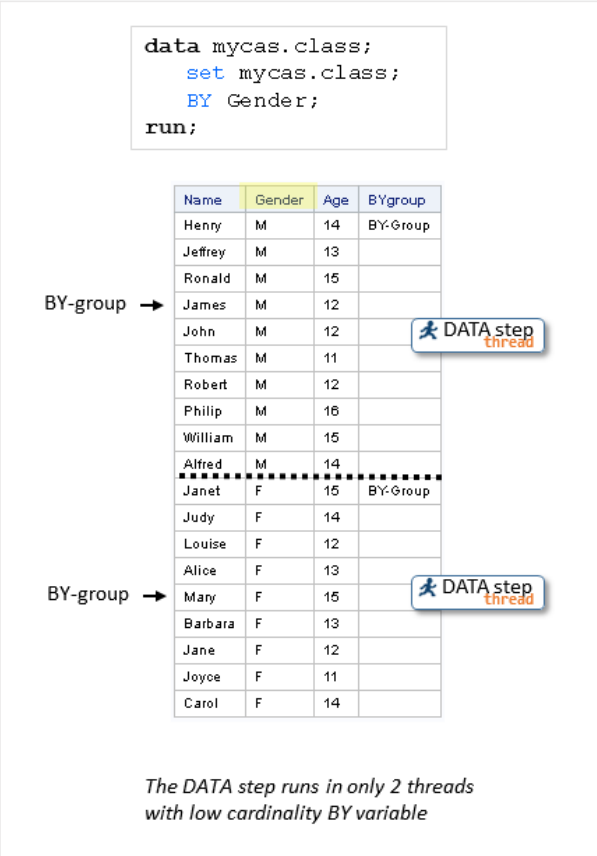
Because an entire BY group is operated on by a single DATA step thread, fewer threads are used if there are fewer BY groups. Fewer BY groups are created when the first BY variable has a low cardinality or is represented by a format with a low cardinality. Therefore, choose your first BY variable wisely and be aware of formats and their cardinality. You might need to assign or reassign formats to your BY variables to enhance performance.

Here are some examples of low and high cardinality formats:

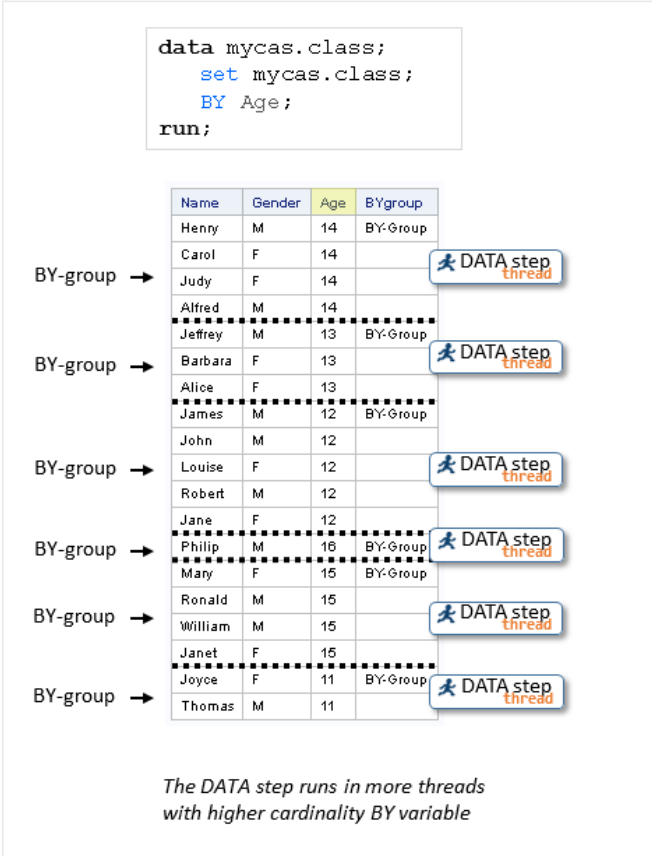
Formats with a High Cardinality	Formats with a Low Cardinality
• DATEw.	• QTRw.
• DATETIMEw.d	• MONNAME.
• DDMMYYw.	• WEEKDAYw.

The following image shows the relationship between the cardinality of the first BY variable and the number of threads that are spawned by the DATA step in BY-group processing.

Note: This image depicts a Bygroup column that shows the start of each BY group. For simplicity, the full code used to create this column is not shown in the image. To see the full example, see "View DATA Step Processing Information Using Automatic Variables" on page 70.



The DATA step runs in only 2 threads with low cardinality BY variable



The DATA step runs in more threads with higher cardinality BY variable

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

BY-Group Processing versus Partitioning

BY-group processing in the CAS DATA step is an ad hoc grouping of data that is useful for performing one-time DATA step computations on small to medium tables. When you perform BY-group processing on a CAS table, the DATA step creates a temporary copy of the in-memory table that exists for the duration of the DATA step job. Once the DATA step finishes processing, the temporary table and all its BY groups disappear. Future data access to BY groups requires that the table is re-grouped and re-distributed each time. In other words, BY-group processing in the CAS DATA step does not affect the structure or distribution of the original in-memory table. BY-group processing is, therefore, more suitable for instantaneous DATA step computation.

The partitioning feature in CAS provides a more permanent and efficient solution for grouping and managing data. The work required for partitioning does not have to be repeated as it does when using BY groups in the DATA step. When you partition a table as you load it using the Partition action, it becomes a partitioned in-memory table that can be accessed by subsequent operations.

To partition CAS tables, you can use the partition action. See "Partition a Table Using the Partition Action" in *SAS Viya: System Programming Guide* for examples and syntax for using the Partition action in CAS.

Other Methods for Partitioning, Grouping, and Ordering

Here are some other methods for ordering and grouping data:

- PARTITION action
- ORDERBY= and GROUPBY= options in the PROC CASUTIL LOAD statement
- PARTITION= and ORDERBY= data set options in the SAS DATA statement

For an example of using the BY statement as described in this section, see "Group and Order Rows Using the DATA Step BY Statement" on page 64.

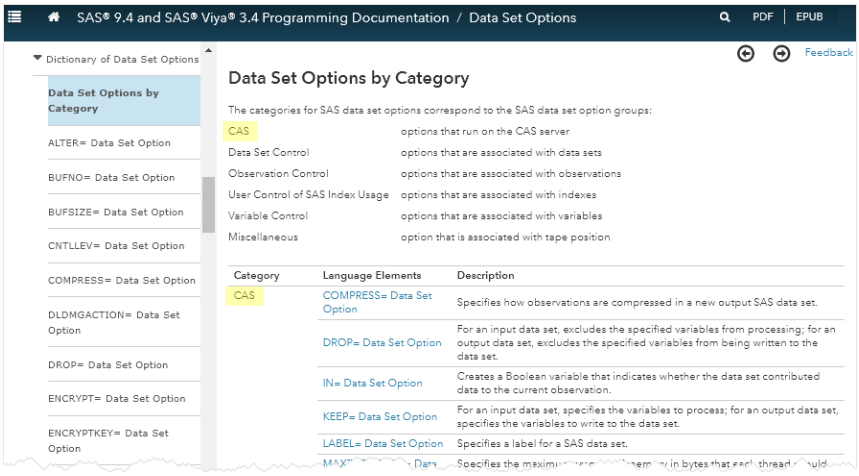
Language Elements Support

Supported Language Elements

Not all SAS language elements are supported in CAS. Below are links to the reference documentation for the SAS DATA Step language. Each document contains a "By Category" table that lists the CAS supported language elements under the CAS table entry:

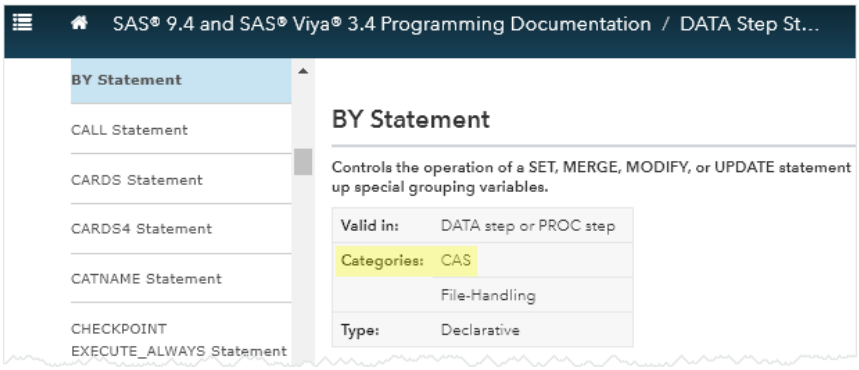
- "Data Set Options by Category" in *SAS Data Set Options: Reference*
- "Formats by Category" in *SAS Formats and Informats: Reference* and "Informats by Category" in *SAS Formats and Informats: Reference*.
- "SAS Functions and CALL Routines by Category" in *SAS Functions and CALL Routines: Reference*
- "DATA Step Statements by Category" in *SAS DATA Step Statements: Reference*

Figure 1.2 *Example Documentation Showing How the CAS-supported Language Elements Are Listed*



Support for CAS is also listed on the individual language element syntax pages for all the DATA step language elements. The support is indicated the categories field as shown in the example document below:

Figure 1.3 *Example Documentation Showing CAS Support on Individual Syntax Page*



For CAS-specific DATA step language elements, see *SAS Cloud Analytic Services: User's Guide*.

Restrictions

The following table contains restrictions on the use of some DATA step language elements in CAS.

Table 1.2 *Restrictions and Notable Behaviors for DATA Step Processing*

Feature	Description
APPEND= data set option	You cannot set both the PROMOTE= and the APPEND= data set options to YES at the same time (in the same DATA step). You can set either PROMOTE= or APPEND= to YES. An error results if you set both to YES at the same time. For more information, see "APPEND= Data Set Option" in *SAS Cloud Analytic Services: User's Guide*.
BY variables	BY variables are supported in the DATA step in CAS but there are some differences between how SAS performs groups variables and how CAS groups them. See "Ordering in CAS" on page 12 and "BY-Group Processing" on page 14 for more information.
BY statement, DESCENDING option	The DESCENDING option in the BY statement is not supported in a DATA step that is running in CAS.
FORMAT procedure	User-defined formats that are created with PROC FORMAT must exist both in SAS and in CAS when using them in a DATA step that is running in CAS. If the format is not available to SAS, the DATA step will fail before it tries to run in CAS. For more information, see "FORMAT Procedure" in *Base SAS Procedures Guide*.

Formats	These formats are not supported in a DATA step that is running in CAS: • WORDS • WORDF • WORDDATE (use NLDATES instead) • WEEKDATE (use NLDATEW instead) The NLDATEW format and the NLDATE format automatically translate the date values to the language specified in your LOCALE= setting.
Functions	Not all functions are supported in a DATA step that is running in CAS. See Functions and CALL Routines by Category for a list of supported functions.
Functions and Session Encoding	The default encoding method for sessions in CAS is UTF-8. In a UTF–8 environment, data can contain multibyte characters that are equivalent to more than 1 byte. Some 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 rather than the number of characters. See "About Session Encoding" on page 26 for information about how session encoding affects behavior. For an example showing how some functions behave, see "Index CHAR and VARCHAR Character Strings" on page 55 for an example of this behavior.
INFILE, INPUT, and DATALINES statements	These language elements are not supported in a DATA step that is processing in CAS. You can use these statements in a SAS client session and then upload the results to CAS.
Macros	The SYMGET and SYMPUT functions are not supported in a DATA step that is running in CAS. For more information about macro processing in SAS, see "Macro Processing" on page 20.
PROMOTE= data set option	You can set either PROMOTE= or APPEND= to YES. An error results if you set both to YES at the same time.
MODIFY, REMOVE, and REPLACE statements	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 and then upload the results to CAS.
VARCHAR data type	There are some restrictions on using the VARCHAR data type. See "CAS LIBNAME Statement" in *SAS Cloud Analytic Services: User's Guide* for more information.
Views	Views can be stored only in a Base SAS library (a library created using the V9 engine). The VIEW option in the DATA statement is not supported with the DATA step in SAS Cloud Analytic Services. For more information, see "Create a DATA Step View" on page 52 and "VIEW=view-name" in *SAS DATA Step Statements: Reference*.
WHERE= data set option	CAS does not support the WHERE= data set option for output tables. If you specify the WHERE= data set option on the output data set, the DATA step automatically runs in SAS rather than in CAS. Specify the MSGLEVEL=i option to see more processing information in the log.

zero-column tables	CAS does not support zero-column tables. This is different from the DATA step in SAS, which does support zero-column tables. You will get an error if you try to create a table in CAS that has no variables.

Macro Processing

Support for Macro Processing

Macro code itself does not execute on the server. It is precompiled and runs in the SAS client session. The code that the macro generates can be made to run in CAS, however. See "Examples" on page 21.

The following DATA Step Functions for Macros are not supported in a DATA step that is running in CAS:

- SYMGET
- SYMGETN

The following DATA Step CALL Routines for Macros are not supported in a DATA step that is running in CAS:

- CALL SYMPUT Routine
- CALL SYMPUTN Routine
- CALL SYMPUTX Routine

These restrictions in macro processing do not apply to the SAS client. These functions are supported in a DATA step that is processing in SAS:

Example Code 7 *SYMGET Is Supported in SAS Client Session Processing*

```
libname sasdata 'c:\Users\sasuser\';
data sasdata.test;
  x = SYMGET('SYSDATE9');
  put x;
run;;
```

But, they are not supported in a DATA step that is running in CAS:

Example Code 8 *DATA Step Explicitly Running in CAS with Unsupported Language Elements*

```
libname mycas cas;
data mycas.test / sessref=casauto;
  x = SYMGET('SYSDATE9');
  put x;
run;
```

Output 1.2 *Log Output Showing DATA Step Explicitly Running in CAS with Unsupported Language Elements*

```
NOTE: Running DATA step in Cloud Analytic Services.
ERROR: The function SYMGET is unknown, or cannot be accessed.
ERROR: The action stopped due to errors.
```

Note: If you do not see the message **`Running DATA step in Cloud Analytic Services`** in the SAS log, then the DATA step did not run in CAS.

SAS does not necessarily return an error code or stop processing if you specify unsupported language elements in a DATA step that is writing out to a CAS table. Instead, the DATA step processes in the SAS client, and then sends the results to the CAS server.

TIP To ensure that the DATA step runs in CAS, specify the SESSREF= option in the DATA statement. This causes the DATA step to explicitly run in CAS.

TIP Specify the MSGLEVEL= option in OPTIONS system option to get more details in the SAS log:

```
options msglevel=i;
data mycas.test;
   set mycas.test;
   x = SYMGET('SYSDATE9');
   put x;
run;
```

Output 1.3 *Log Output Showing Error When MSGLEVEL=i Is Specified and SYMGET Is Used in a CAS DATA Step*

```
NOTE: Could not execute DATA step code in Cloud Analytic Services. Running DATA
step in the SAS
      client.
19MAR2019
```

For more information about macro processing in SAS, see "Using the Macro Facility in SAS Viya" in *SAS Macro Language: Reference*

Examples

You can use SAS macros in a variety of ways with the CAS server environment. For example, you can create macros that generate SAS code that executes in CAS. But, the macro code itself does not execute on the server.

Here are some examples showing how to use macros to execute the CAS procedure, the CASUTIL procedure, and the CAS DATA step.

Example Code 9 *Create a Macro for Uploading a Sashelp Data Set into CAS*

```
%macro loadit(name=);
proc casutil;
   load data=sashelp.&name
   outcaslib='casuser'
   replace;
run; quit;
%mend loadit;

%loadit(name=cars)
```

Example Code 10 *Create a Macro for Saving a Sashelp Data Set in CAS*

```
%macro loadsaveit(name=);
proc casutil;
   load data=sashelp.&name
   outcaslib='casuser'
   replace;
   save casdata="&name" replace;
run; quit;
%mend loadsaveit;
```

```
%loadsaveit(name=cars)
```

Example Code 11 *Create a Macro for Dropping a Table from Memory*

```
%macro dropit(name=);
proc casutil incaslib="casuser";
  droptable casdata="&name";
  list files;
run; quit;
%mend dropit;

%dropit(name=cars)
```

Example Code 12 *Create a Macro for Getting Server Information Using the serverStatus Action*

```
%macro aboutServer;
   serverStatus;
%mend;
proc cas;
   %aboutServer;
run;
```

CAS DATA Step Action

CAS actions are requests that are sent by the user to the server to perform a task. You can use CAS actions to submit a DATA step. CAS actions are submitted within a PROC CAS block.

For more information about the DATA step actions available in CAS, see "DATA Step Action Set" in *SAS Viya: System Programming Guide*.

Data Types

For information about data types, see "Data Types" in *SAS Cloud Analytic Services: User's Guide*.

Rules for Names

For information about rules for names in CAS, see "Variable Names and Data Set Names in CAS Engine" in *SAS Cloud Analytic Services: User's Guide*.

Automatic Variables

Overview

Automatic variables are system variables that are created automatically by the SAS DATA step and saved in memory. They are not included in the output 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. For an example that uses automatic variables, see "View DATA Step Processing Information Using Automatic Variables" on page 70.

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

For information about how to run the examples, see Set Up Code For Examples on page 32.

ERROR

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

ERROR is initialized to 0. It is set to 1 when an error is encountered. Errors include input data errors, conversion errors, or math errors, such as division by 0. You can use the value of this variable to help locate errors in data records and to print error messages to the SAS log.

In the following example, the IF statement writes 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:

```
data mycas.test;
input x y;
if _error_ = 1 then
put "****** Error in row " _n_ " ******";
datalines;
1 1
2 3
3 g
4 4
;
```

```
NOTE: Invalid data for y in line 64 3-3.
 ****** Error in row 3  ******
 RULE:      ----+----1----+----2----+----3----+----4----+----5--
 64         3 g
 x=3 y=. _ERROR_=1 _N_=3
 NOTE: The data set MYCAS.TEST has 4 observations and 2 variables.
```

_N_

is an internal system variable that counts the iterations of the DATA step as it automatically loops through the rows of an input data set. The _N_ variable is initially set to 1 and increments by 1 each time the DATA step loops past the DATA statement. The DATA step loops past the DATA statement for every row that it encounters in the

input data. Because the DATA step is a built-in loop that iterates through each row in a table, the _N_ variable can be used as a counter variable.

Note: The DATA step creates and stores the _N_ automatic variable internally but it does not include the variable in the output data set.

You can use the PUT statement with the _ALL_ argument to print the values of _N_ and _ERROR_ to the SAS log.

In a multi-node server session, _N_ returns the number of DATA step iterations (table rows) per worker node, per thread. For example, suppose that an eight-row table is distributed across four worker nodes. If each node processes 2 row in a single thread, then the log displays all rows processed on each node:

```
data mycas.test;
   set mycas.test;
   put "This is iteration (row) # " _n_
       "on " _hostname_;
run;
```

```
NOTE: Running DATA step in Cloud Analytic Services.
 NOTE: The DATA step will run in multiple threads.
 This is iteration (row) # 1 on srvr004
 This is iteration (row) # 2 on srvr004
 This is iteration (row) # 1 on srvr005
 This is iteration (row) # 2 on srvr005
 This is iteration (row) # 1 on srvr002
 This is iteration (row) # 2 on srvr002
 This is iteration (row) # 1 on srvr003
 This is iteration (row) # 2 on srvr003
```

HOSTNAME

returns the name of the worker node or host that the DATA step is running on. In a DATA step that is processing in SAS, _HOSTNAME_ is the name of the machine SAS is running on.

```
data mycas.bigcars(where=(weight>5000));
   set sashelp.cars;
   keep make weight;
run;
data mycas.bigcars;
   set mycas.bigcars;
   keep make weight;
   put _hostname_;
run;
```

```
 NOTE: Running DATA step in Cloud Analytic Services.
 NOTE: The DATA step will run in multiple threads.
 server006
 server005
 server007

 NOTE: Duplicate messages output by DATA step:
 server006  (occurred 3 times)
 server005  (occurred 3 times)
 server007  (occurred 2 times)
...
 NOTE: There were 21 observations read from the table BIGCARS
       in caslib casuser(user-id).
 NOTE: The table bigcars in caslib CASUSER(user-id) has 21 observations
       and 2 variables.
```

NTHREADS

returns the number of DATA step threads running in a server session. In a DATA step that is processing in SAS, _NTHREADS_ is always 1.

```
data mycas.bigcars(where=(weight>5800));
   set sashelp.cars;
   keep make weight;
run;
data mycas.bigcars;
   set mycas.bigcars;
   put "The DATA step is running in " _nthreads_ "threads";
run;
```

```
NOTE: Running DATA step in Cloud Analytic Services.
 NOTE: The DATA step will run in multiple threads.
 The DATA step is running in 256  threads
 NOTE: Duplicate messages output by DATA step:
 The DATA step is running in 256  threads  (occurred 5 times)
```

THREADID

returns the number that is associated with the thread that the DATA step is running in a server session. In a DATA step that is processing in SAS, _THREADID_ is always 1.

```
data mycas.bigcars(where=(weight>5800));
   set sashelp.cars;
   keep make weight;
run;

data mycas.bigcars;
   set mycas.bigcars;
   if _N_ = 1 then do;
     put "The ThreadID is " _threadid_;
   end;
run;
```

```
NOTE: Running DATA step in Cloud Analytic Services.
 NOTE: The DATA step will run in multiple threads.
 The ThreadID is 65
 The ThreadID is 33
 The ThreadID is 97
 The ThreadID is 129
 The ThreadID is 1
```

Avoiding Data Truncation

About Session Encoding

If your SAS session uses an encoding method other than UTF-8, SAS automatically transcodes data that you move from SAS to CAS, and vice versa. Transcoding is the automatic process by which SAS converts data from one session encoding to another.

The CAS server uses UTF-8 session encoding. Your local SAS session can use any number of different encoding methods depending on your locale and the settings that are configured for your site. The default SAS session encoding is WLatin1 for Windows and Latin1 for UNIX.

The number of bytes for a character in your local SAS session encoding can be different from the number of bytes for the same character in CAS. For example, the characters é and ë in a Latin1 session encoding require only one byte for storage but in UTF-8 encoding these same characters require 2 bytes for storage. This is because UTF-8 is a multi-byte character set (MBCS) while Latin1 is a single-byte character set (SBCS). Therefore, transcoding can result in data truncation.

Complete information about session encoding and how you can manage transcoding issues in SAS can be found in here:

- "Transcoding for NLS" in *SAS National Language Support (NLS): Reference Guide*
- "Encoding for NLS" in *SAS National Language Support (NLS): Reference Guide*
- *SAS National Language Support (NLS): Reference Guide*

Specific information about the VARCHAR data type can be found in "VARCHAR Data Type" in *SAS Cloud Analytic Services: User's Guide*.

Determine the Session Encoding

To determine session encoding for your SAS session, specify the PROC OPTIONS statement:

```
proc options option=encoding; run;
```

The session encoding information is printed in the SAS log.

To determine session encoding for a specific SAS data set, specify the CONTENTS procedure:

```
proc contents data=sashelp.manage; run;
```

Examples of Truncation Errors

Overview

Truncation of the data can occur if your SAS session encoding is not UTF-8.

When you declare a variable in a CAS DATA step, the length of the variable is calculated in the SAS session then sent to CAS for processing. CAS creates a table with columns based on the variable length information from SAS. Truncation of the data can occur if your local SAS session is not compatible with the UTF-8 CAS session. Or, you might get a transcoding error in the SAS log.

For information about how to run the examples, see Set Up Code For Examples on page 32.

Example 1

In the following example, the local SAS session encoding is EUC-CN (Chinese) and the DATA step runs in CAS. The code is first partially compiled in the SAS client where information about the length of the variables is determined and then sent to CAS for processing. CAS creates a table based on these column widths.

In this example, the EUC-CN character set requires 2 bytes per character of storage in SAS. The length of the CHAR variable, **c**, is automatically set to 8 bytes (to enable saving 4 EUC-CN characters). SAS sends this length information to CAS and CAS creates a table column with a width of 8 bytes. When the DATA step runs in CAS, the value 我的例子 is assigned to the variable **c**. Truncation occurs because 我的例子 requires 3 bytes per character in UTF-8 (12 bytes total) and the variable has a length of only 8 bytes.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
data mycas.example;
   c = '我的例子';
run;
```

Solution: Explicitly define the length of character variables to ensure that they are long enough to accommodate the lengths when they are converted to UTF-8 encoding in CAS. If your local session encoding uses a DBCS, then set the CHAR length to be equal to the original length times 2. In the solution below, the length of the CHAR is set to 16 bytes, enough to hold 4 multibyte characters.

```
data mycas.example;
   length c $ 16; /* in UTF-8 this variables might */
    c = '我的例子'; /* require up to 4 bytes per character */
run;
```

Example 2

In the following example, the DATA step is running in CAS. The CHAR variable contains a Latin1 character with diaeresis. The Latin1 character, **ä**, requires 1 byte in Latin1 and 2 bytes in UTF-8. SAS compiles the DATA step in the local SAS session, automatically declaring the variable **c** to be 4 bytes long, and sends information about the variable lengths to CAS. The value **äbcd** uses 5 bytes in UTF-8 in CAS. When this value is assigned to variable **c**, which was automatically declared to be 4 bytes long, truncation occurs.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
data mycas.example;
   length c $4;
    c = 'äbcd'; /* length in Latin1 is 4 bytes and 5 bytes in UTF-8 */
run;

proc print data=mycas.example; run;
```

Obs	c
1	äbc

Solution: Increase the length of character variables to ensure that they are long enough to accommodate the lengths when they are converted to UTF-8 encoding in CAS.

```
data mycas.example2;
   length c $5;
    c = 'äbcd'; /* length in Latin1 is 4 bytes */
run;

proc print data=mycas.example2; run;
```

Obs	c
1	äbcd

Solution: Change the variable type to VARCHAR. VARCHAR variables are declared in units of characters instead of bytes, making it simpler to work with multibyte characters.

```
data mycas.example2;
   length vc varchar(*);
   vc = 'äbcd'; /* length in CAS UTF-8 is 5 */
run;

proc print data=mycas.example2;
```

Obs	vc
1	äbcd

Note: The NCHARMULTIPLIER=, TRANSCODE_FAIL, and ENCODING= data set options are not supported in a DATA step that runs in CAS. However, they can be used effectively in a DATA step that is loading to CAS to manage transcoding errors.

Example 3

When you save a CAS table that contains a VARCHAR variable as a SAS data set, SAS automatically converts the variable to a CHAR variable. You will see the following in the log.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
NOTE: One or more variables were converted because the data type is not
supported by the V9 engine.
```

If you load a SAS data set to CAS, you might get a transcoding error due to incompatible session encodings between SAS and CAS. When you transfer SAS data sets to CAS, SAS converts the data in your local SAS session into UTF-8 data for the CAS session.

```
ERROR: Some character data was truncated during transcoding in the dataset
MYCAS.CITIES. Use of
       the NCHARMULTIPLIER option is recommended.
```

For example, the following program might cause an error if the local SAS session encoding is Latin1:

```
data cities;
   city = "Sévérac-le-Château"; /* 15 Latin1 characters totaling 15 bytes */
run;                            /* 15 UTF-8 characters totaling 18 bytes */
data mycas.cities;
   set cities;
run;
```

Solution: Specify the NCHARMULTIPLIER= data set option on the output data set. This option increases the length of the transcoded variables.

```
data mycas.cities(NCHARMULTIPLIER=2);
   set cities;
run;
```

VARCHAR Length Conversion

Support

VARCHAR variables are not supported by the SAS V9 engine, so SAS automatically converts VARCHAR variables to CHAR variables.

If you try to store a data containing a VARCHAR in anything other than a CAS table, SAS automatically converts the VARCHAR variables to CHAR variables. When converting a variable from a VARCHAR(*n*) to a CHAR, SAS sets the length of the converted VARCHAR based on your local SAS session encoding. To calculate the length, SAS multiplies the current length of the VARCHAR, *n*, by the maximum value that a character's length can be in the local SAS session encoding.

For information about how to run the examples, see Set Up Code For Examples on page 32.

Example 1

In this example, assume that the local SAS session encoding is Latin1.

```
proc options option=encoding; /* determine the encoding for your local session */
run;
data mycas.example;
   length c $10;
   length vc varchar(4);
    c = 'abcd';
   vc = 'abcd';
run;
proc contents data=mycas.example; run;
```

Here is the Log output for the OPTIONS procedure:

```
ENCODING=WLATIN1 Specifies the default character-set encoding for the SAS
session.
NOTE: PROCEDURE O
```

Here is the PROC CONTENTS output for the original CAS table:

Alphabetic List of Variables and Attributes					
			Len		Max Bytes Used
#	Variable	Type	Bytes	Chars	
1	c	Char	10		
2	vc	Varchar	4	4	4

```
data example;
   set mycas.example;
run;
proc contents data=example; run;
```

Here is the PROC CONTENTS output for the SAS data set with the VARCHAR variable converted to a CHAR variable. Notice how the length stays the same when the local session encoding is Latin1. SAS calculates the length of the converted VARCHAR variable by multiplying the original length by the maximum number of bytes required to store characters in the character set. Latin1 is a single-byte character set, so the length is 4 x 1, or 4 bytes.

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	c	Char	10
2	vc	Char	4

Example 2

For this example, use the program in "Example 1" on page 29 and assume that the local session encoding is UTF-8. When the VARCHAR variable is converted to a CHAR variable, the length of the CHAR variable is 4 times the length of the largest possible character in UTF–8. The largest possible number of bytes required to store a single character in UTF-8 is 4. So, the length is 4 x 4, or 16 bytes.

Here is the output when above is run in a local UTF-8 SAS session:

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	c	Char	10
2	vc	Char	16

More Information about SAS Session Encoding

- SAS® and UTF-8: Ultimately the Finest
- Multilingual Computing with SAS® 9.4
- Usage Note 51586: How to change the default SAS® session encoding to UTF-8

Chapter 2

DATA Step Examples

Set Up Code for Examples

User Interfaces

Connection with the CAS server is automatically created when you use SAS Studio 4.4 on Linux or SAS Studio 5.1 on Linux and Windows, which are the web programming interfaces for SAS Viya. The connection information, the host name and port, are set during installation. You can view those values in the CASHOST and CASPORT system options using this code:

```
cas casauto;
```

For more information about connecting to CAS, see "Connecting to the CAS Server and Creating a CAS Session" in *An Introduction to SAS Viya Programming*.

Set Up Code

The following code connects you to a CAS server, starts a CAS session, creates a CAS engine libref, and associates SAS librefs with caslibs. You might not need to manually connect to the CAS server (for example, if you are running SAS Studio, which connects to CAS automatically). If you do need to connect to the server, uncomment the OPTIONS statement below and follow the numbered instructions.

```
/* options cashost="cloud.example.com" casport=5570 */      /*1*/
cas casauto sessopts=(caslib='casuser');                     /*2*/
libname mycas cas;                                           /*3*/
caslib _all_ assign;                                         /*4*/
```

1 Specify the host name of your CAS server in quotation marks in the CASHOST= option. Specify the port number that the server is running on in the CASPORT= option.

2 Start a CAS session named Casauto and specify Casuser as the active caslib. Casuser is a personal caslib. For more information, see "Caslibs" in *SAS Cloud Analytic Services: Fundamentals*.

3 If you have not already done so, create a CAS engine libref.

4 Assign SAS librefs to existing caslibs to allow caslib names to appear in the user interface.

TIP Specify the "SESSREF=session-reference-name" in *SAS DATA Step Statements: Reference* option in the "DATA Statement" in *SAS DATA Step Statements: Reference* to explicitly run the DATA step in CAS. Also, specify `i` in the "MSGLEVEL= System Option" in *SAS System Options: Reference* system option to get more detailed log results, as shown in this example:

Run the DATA Step in CAS

Example Code

The following example shows how to run the DATA step in CAS.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
data mycas.hello;                        /* 1 */
   put 'Hello from ' _hostname_          /* 2 */
       'thread #'   _threadid_;          /* 3 */
   x=1;                                  /* 4 */
run;
```

1 A CAS engine libref is specified in the DATA statement to create an in-memory CAS table named mycas.hello.

2 The _HOSTNAME_ automatic DATA step variable is specified in a PUT Statement to demonstrate how the DATA step runs in CAS in a distributed environment.

3 The _THREADID_ automatic DATA step variable is specified to demonstrate multithreaded processing in CAS.

4 CAS tables require at least one variable. An assignment statement is used to create a variable.

Output 2.1 *Log Output Showing Multithreaded DATA Step Processing in a Distributed CAS Environment*

```
NOTE: Running DATA step in Cloud Analytic Services.
 Hello from server001 thread #2
 Hello from server001 thread #3
 Hello from server001 thread #4
 Hello from server001 thread #1
 NOTE: The table hello in caslib CASUSER(username) has 4 observations and 1
variables.…
```

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 controlling DATA processing, see "Controlling Threaded Processing" on page 10.
- For a DATA step to run in CAS, you must make sure you have met the requirements for processing in CAS on page 4.
- You can also use the CASUTIL procedure to load your SAS data sets into CAS. This method is usually more efficient for transferring large data sets to CAS. For more information, see "CASUTIL Procedure" in *SAS Cloud Analytic Services: User's Guide* for more information.
- Not all language elements are supported in a CAS DATA step. See "Language Elements Support" on page 17 for information about what language elements are supported in the CAS DATA step.

See Also

- "Running the DATA Step in CAS".
- "Automatic Variables" on page 23 (_THREADID_ and _HOSTNAME_)
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "CASLIB Statement" in *SAS Cloud Analytic Services: User's Guide*
- "Accessing Data" in *SAS Cloud Analytic Services: User's Guide*
- "CASUTIL Procedure" in *SAS Cloud Analytic Services: User's Guide*

Run the DATA Step in CAS on a CAS Table

Example Code

To run the DATA step in CAS using input data, both the input and output must be CAS tables and both tables must use the same CAS engine libref. In the following example, the CASUTIL Procedure is used to load a SAS data set into CAS as an in-memory table. The SAS data set name is preceded by the CAS engine libref, **mycas**.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
proc casutil outcaslib='casuser';
   load data=sashelp.cars;                /* 1 */
run;

data mycas.cars2 / sessref=casauto;       /* 2 */
   set mycas.cars;                        /* 3 */
   if mpg_city > 25 then eff='Y';
      else eff='N';
   put 'Thread number: '  _threadid_
       'on worker node '  _hostname_;
```

```
run;
```

1. Use PROC CASUTIL to load the **`Sashelp.Cars`** data set to CAS. This creates an in-memory CAS table named Cars. The OUTCASLIB= option loads the CAS table to the CASUSER caslib.
2. Specify a CAS table as the output data using the CAS engine libref.
3. 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 controlling DATA processing, see "Controlling Threaded Processing" on page 10.
- For a DATA step to run in CAS, you must make sure you have met the requirements for processing in CAS on page 4.
- You can also load external files to CAS using the CASUTIL procedure with the LOAD FILE statement. The following examples show how to load files into CAS using PROC CASUTIL: "Load a CSV File into CAS" in *SAS Cloud Analytic Services: User's Guide*, Load a Client-Side File and Load a Server-Side File.
- The DATA step automatically runs in CAS when you use a CAS engine libref on the input and output data sets. For more information about controlling where the DATA step runs by default, see "Controlling Where the DATA Step Runs by Default" on page 7.
- Not all language elements are supported in a CAS DATA step. See "Language Elements Support" on page 17 for information about what language elements are supported in the CAS DATA step.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "CASUTIL Procedure" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB "_ALL_ " in *SAS Cloud Analytic Services: User's Guide* statement
- "SESSOPTS=(session-option(s))" in *SAS Cloud Analytic Services: User's Guide*
- "The Active Caslib" in *SAS Cloud Analytic Services: User's Guide*
- "Accessing Data" in *SAS Cloud Analytic Services: User's Guide*

Load a CSV File into CAS and Save as a CAS Table

Example Code

The following example shows how to load a comma-separated value file (CSV) into CAS using the DATA step. This example uses the names.csv file, which can be found here: http://support.sas.com/documentation/onlinedoc/viya/examples.htm.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
filename names url
 "http://support.sas.com/documentation/onlinedoc/viya/exampledatasets/names.csv";

data mycas.names;
  infile names dsd truncover firstobs=2;                    /* 1 */
  input BRTH_YR :$10. GNDR :$10. ETHCTY :$10. NM :$10.
            CNT :$10.  RNK :$10.;                           /* 2 */
run;

proc casutil incaslib='casuser';                            /* 3 */
   save casdata='names' outcaslib='casuser' replace;
   list;
run;
```

Below is a partial view of the CSV file and its contents:

```
BRTH_YR,GNDR,ETHCTY,NM,CNT,RNK
2011,FEMALE,HISPANIC,GERALDINE,13,75
2011,FEMALE,HISPANIC,GIA,21,67
2011,FEMALE,HISPANIC,GIANNA,49,42
2011,FEMALE,HISPANIC,GISELLE,38,51
        <more rows>
2014,MALE,WHITE NON HISPANIC,Youssef,24,88
2014,MALE,WHITE NON HISPANIC,Yusuf,16,96
2014,MALE,WHITE NON HISPANIC,Zachary,90,39
2014,MALE,WHITE NON HISPANIC,Zev,49,65
```

1. In SAS, load the external comma-separated file using the INFILE statement. Specify a CAS engine libref on the output table. The TRUNCOVER option enables SAS to correctly read in variable-length records. Variables without any values assigned are set to missing.
2. Specify the INPUT statement to list the column names and read them in as informats.
3. Save a permanent copy of the in-memory CAS table.

Key Ideas

- You can load external files to CAS by using an INFILE statement in the DATA step and a CAS engine libref on the output table.
- You can also load external files to CAS using the CASUTIL procedure with the LOAD FILE statement. The following examples show how to load files into CAS using PROC CASUTIL: "Load a CSV File into CAS" in *SAS Cloud Analytic Services: User's Guide*, Load a Client-Side File and Load a Server-Side File.

See Also

- "FILENAME Statement, URL Access Method" in *SAS Global Statements: Reference*
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "CASUTIL Procedure" in *SAS Cloud Analytic Services: User's Guide*

- "SESSOPTS=(session-option(s))" in *SAS Cloud Analytic Services: User's Guide*
- "INFILE Statement" in *SAS DATA Step Statements: Reference*
- "About Informats" in *SAS Formats and Informats: Reference*
- "CASLIB Statement" in *SAS Cloud Analytic Services: User's Guide*

Manage Tables

Convert a CAS Table to a SAS Data Set

Example Code

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 creates a SAS data set in the local SAS **Work** library.

For information about how to run the examples, see Set Up Code For Examples on page 32.

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

libname mysas "/file-path/";                /* 2 */
data mysas.earnings;                        /* 3 */
   set mycas.earnings;
run;
```

1 Create variables for the table and calculate interest earned using a DO statement.

2 Create a libref named Mysas for storing the data set. The libref Mysas represents the physical location in which the Earnings data set is stored.

3 Read in the CAS table and write it out as a SAS data set to the SAS library.

Obs	Amount	Rate	month	Earned
1	1000	.00625	13	77.6326

Key Ideas

- A CAS DATA step that has an output table but no input table runs in a single thread by default. You can force the DATA step to run in multiple threads by specifying the SINGLE=NO option in the DATA statement. See "SINGLE=NO | YES | NOINPUT" in *SAS DATA Step Statements: Reference* for information about specifying the SINGLE= option.
- For more information about where the DATA step runs and controlling DATA step processing modes, see "Processing Modes" on page 8 and "Controlling Where the DATA Step Runs by Default" on page 7.
- The CAS DATA step works only on CAS tables.
- Not all language elements are supported in a CAS DATA step. See "Language Elements Support" on page 17 for information about what language elements are supported in the CAS DATA step.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB _ALL_ statement
- "LIBNAME Statement" in *SAS Global Statements: Reference*
- "Accessing Data" in *SAS Cloud Analytic Services: User's Guide*
- "DATA Statement" in *SAS DATA Step Statements: Reference*

Combine Tables Using the SET and BY Statements

Example Code

The following example creates two data sets and then loads them into CAS tables using PROC CASUTIL. The first DATA step combines the two data sets by concatenating one to the other. This is similar to appending tables to one another. The second DATA step interleaves the two data sets by adding the BY statement. The BY variables are **Common** and **Number**.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
data animal1;                                /* 1 */
input Common $ Animal $ 3-8
      Number 10-11;
datalines;
a Ant    1
b Bird   2
c Cat    3
d Dog    4
e Eagle  5
f Frog   6
g Goose  7
h Hawk   8
i Impala 9
;
data plant1;                                 /* 2 */
input Common $ Plant $ 3-10
      Number 12-13;
```

```
datalines;
a Grape    1
c Hazelnut 2
e Indigo   3
g Jicama   4
i Kale     5
;

data append;                                /* 3 */
  set animal1 plant1;
run;

proc print data=append;
title '1. Plant1 Appended to Animal1 in a ';
title2 'DATA Step Running in SAS'; run;

data interleaveByCommon;                    /* 4 */
  set animal1 plant1;
  by common; run;

proc print data=interleaveByCommon;
title '2. Animals and Plants Interleaved
          by Variable COMMON';
title2 'in a DATA Step Running in SAS';
run;

data interleaveByCommonNumber;              /* 5 */
   set animal1 plant1;
   by common Number; run;

proc print data=interleaveByCommonNumber;
title '3. Animals and Plants Interleaved
          by Common and Number';
title2 'in a DATA Step Running in SAS';
run;

proc casutil incaslib="casuser"             /* 6 */
             outcaslib="casuser";
  load data=animal1 replace;
  load data=plant1 replace;
run;

data mycas.appendCas / sessref=casauto;     /* 7 */
   set mycas.animal1 mycas.plant1;
   hostname=_hostname_; run;

proc print data=mycas.appendCas;
title  '4. Plant1 Appended to Animal1 in a';
title2 'Multithreaded DATA Step
        Running in CAS';
run;


data mycas.byCommon / sessref=casauto;
   set mycas.animal1 mycas.plant1;          /* 8 */
   by common;
```

```
   hostname=_hostname_; run;

proc print data=mycas.byCommon;
title '5. Animal and Plant Combined
          by Common';
title2 'in a Multithreaded DATA Step
        Running in CAS';
run;

data mycas.ByCommonNumber / sessref=casauto; /* 9 */
   set mycas.animal1 mycas.plant1;
   by common Number;
   hostname=_hostname_;
run;

proc print data=mycas.ByCommonNumber;
title '6. Animal and Plant Combined
         by Common and Number';
title2 'in a DATA Step Running in CAS';
run;
```

1 Create the first data set.

2 Create the second data set.

3 Append the Plant data set to the Animal data set by specifying the SET statement. No BY statement is used.

4 Interleave the two data sets using the BY statement with 1 variable.

5 Interleave the two data sets using the BY statement with 2 variables.

6 Load the data sets into CAS.

7 Combine the two data sets by specifying only the SET statement in a DATA step running in CAS. No BY statement is used.

8 Combine the two data sets using the BY statement with 1 variable in a CAS DATA step.

9 Combine the two data sets using the BY statement with 2 variables in a CAS DATA step.

Key Ideas

- Concatenating data sets is the combining of two or more data sets, one after the other, into a single data set. Concatenating uses the SET statement with multiple input tables. All observations from the first data set are read, followed by all observations from the second data set.
- Interleaving uses a SET statement and a BY statement to combine multiple data sets into one new data set.
- Before you can interleave data sets, the data must be sorted by the same variable or variables that are used with the BY statement that accompanies your SET statement.
- The order of rows within a BY group is not stable within CAS. Unless you explicitly specify additional BY variables to order the rows, the ordering can change from one DATA step job to the next. If you need a stable ordering, explicitly specify additional BY variables. See "How CAS Groups Data with BY Variables" on page 14.
- The BY statement combines the rows in the two data sets that have the same values for the BY variable.
- Instead of concatenating tables, you can append them and produce the same results as concatenating. To avoid reading all the records in both tables, you can append one table to another by using the APPEND= data set option.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB Statement"_ALL_ " in *SAS Cloud Analytic Services: User's Guide*
- "BY Statement" in *SAS DATA Step Statements: Reference*
- "MERGE Statement" in *SAS DATA Step Statements: Reference*
- "APPEND= Data Set Option" in *SAS Cloud Analytic Services: User's Guide*
- "Combining SAS Data Sets: Methods" in *SAS Language Reference: Concepts*

Append Tables Using the APPEND= Data Set Option

Example Code

The following example appends a CAS table to a promoted CAS table by using the APPEND= data set option.

First, create a table to append:

```
data mycas.class(promote=yes);          /*1*/
   set sashelp.class;
   where Weight>115;                    /*2*/
run;
proc print data=mycas.class; run;      /* 3*/
```

1. Create the table mycas.class by reading in the class table from the Sashelp library. Promote the table to global scope by specifying the PROMOTE= data set option.
2. Use the WHERE Statement to select only those rows where the variable Age is greater than 15.
3. Print the results using the PRINT procedure.

Output 2.2 *PROC PRINT Output Showing the Promoted Mycas.Class Table*

Obs	Name	Sex	Age	Height	Weight
1	Philip	M	16	72.0	150
2	Robert	M	12	64.8	128
3	Ronald	M	15	67.0	133

Output 2.3 *Log Output*

```
NOTE: There were 19 observations read from the data set SASHELP.CLASS.
NOTE: The data set MYCAS.CLASS has 19 observations and 5 variables.
NOTE: The data set CLASS was promoted.
```

Create a second table to append to the mycas.class table. The second table is created from a CSV file containing a single row:

```
Aaron,M,15,64,115
```

```
data mycas.addClass;                                        /*1*/
   infile "/<file-path>/addClass.csv" dlm=",";              /*2*/
   input Name $ Sex $ Age Height Weight;                    /*3*/
run;
```

1 Read the CSV file into CAS. Specify mycas.addClass as the output CAS table. This table will be appended to the original table, mycas.class. The CSV file contains the same variables as the first table, mycas.class.

2 Use the INFILE statement to read in the CSV file. Specify the DELIMITER= option to tell SAS that the external file contains comma-delimited data.

3 Use the INPUT statement to tell SAS the names and types of variables that are contained in the external file.

Now, append the mycas.addClass table to the mycas.class table:

```
data mycas.class(append=yes);  /*1*/
   set mycas.addClass;
run;
```

1 Specify the original table, mycas.class, in the DATA statement with the APPEND= data set option. Specify the second table in the SET statement.

Output 2.4 *Log Output for Appending Tables That Contain Variables with Different Lengths*

```
ERROR: The variable Sex has been specified with incompatible data types.
ERROR: The action stopped due to errors.
```

Even though the tables contain the same variables, their lengths do not match. Specifically, the variable **Sex** in the table that was read in from the CSV file has a length of 8 bytes. In the original file read from the Sashelp library, the variable has a length of 1 byte. You can see these differences by specifying the CONTENTS procedure on both tables:

```
proc contents data=mycas.addClass; run;
proc contents data=mycas.class; run;
```

Output 2.5 *PROC CONTENTS Output Showing Variable Information for Appended Tables with Different Data Types for a Variable*

Data Set Name	MYCAS.ADDCLASS

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	8
5	Weight	Num	8

Data Set Name	MYCAS.CLASS

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
3	Age	Num	8
4	Height	Num	8
1	Name	Char	8
2	Sex	Char	1
5	Weight	Num	8

To solve this problem, you can specify the FORCE option in the APPEND= data set option to force the append to concatenate tables whose variables contain different lengths:

```
data mycas.class(append=force);
   set mycas.addClass;
run;
proc print data=mycas.class; run;
```

Output 2.6 *PROC PRINT Output Showing the Results for Appending One Table to Another Table in CAS*

Obs	Name	Sex	Age	Height	Weight
1	Philip	M	16	72.0	150
2	Robert	M	12	64.8	128
3	Ronald	M	15	67.0	133
4	Aaron	M	15	64.0	120

Output 2.7 *Log Output for Appending One Table to Another Table in CAS*

```
NOTE: The table class in caslib CASUSER(casdemo) has 4 observations and 5
variables.
NOTE: The APPEND operation for table class in caslib CASUSER(casdemo) added 1
observations.
```

Alternatively, you can change the length of the variable using the LENGTH statement:

```
data mycas.addClass;
   length Sex $1;
   set mycas.class;
run;
```

Key Ideas

- Unless you specify APPEND=FORCE, the lengths of the variables in the input tables must be the same.
- The FORCE option uses the character length of the in-memory table to determine the length of input observations. A truncation error occurs when the character length sizes of the input data exceeds the character length sizes of the output table.
- Variable attributes, such as name, length, and format, on the input data sets must be the same.
- You can append rows to a globally scoped CAS table.

See Also

- "APPEND= Data Set Option" in *SAS Cloud Analytic Services: User's Guide*
- "LENGTH Statement" in *SAS DATA Step Statements: Reference*
- "CASUTIL Procedure" in *SAS Cloud Analytic Services: User's Guide*
- "DATASETS Procedure" in *Base SAS Procedures Guide*

Manipulate Variables

Sum a Variable across an Entire Table

Example Code

This example sums a variable across all the rows of a very large, distributed table in CAS. To do this, the example uses a two-step process: In the first step, a DATA step runs in parallel on the "big data." In the second step, the DATA step runs in a single thread on the smaller data set to gather the final sum. The data that is collected in the final sum is the total number of homeowners in the data set mycas.purchase.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
/* Multiple Partial sums are computed by multiple threads */
data mycas.sums;
  retain homeowners_sum;                /* 1 */
  keep homeowners_sum;                  /* 2 */
  set mycas.purchase end = done;        /* 3 */
  if demog_ho = 1 then
     homeowners_sum + 1;                /* 4 */
  if done then output;                  /* 5 */
run;
title 'DATA Step in CAS';
proc print data=mycas.sums; run;

data mycas.sums_all / single = yes;     /* 6 */
  retain homeowners_sum_all;            /* 7 */
  keep homeowners_sum_all;              /* 8 */
  set mycas.sums end = done;            /* 9 */
  homeowners_sum_all + homeowners_sum;  /* 10 */
  if done then output;
run;
title 'Final Sum';
proc print data=mycas.sums_all; run;    /* 11 */
```

1 Retain the value of the homeowners_sum variable from one row to the next.

2 Keep only the homeowners_sum variable in the output table.

3 Read in the input table, mycas.purchase. Specify END= to create a flag for the conditional IF statement.

4 Create a sum of the variable Homeowners that includes the values in the rows that are being processed in that DATA step thread (on that node). Each thread calculates a partial sum for the rows that it processes. Store each sum for each thread total in

homeowners_sum. See "Sum Statement" in *SAS DATA Step Statements: Reference* for more information.

5 When the last row is reached, SAS writes out the results.

6 Use the SINGLE=YES option to run the DATA step in a single thread.

7 Retain the value of the homeowners_sum_all variable from one row to the next.

8 Keep only the homeowners_sum_all variable in the output table.

9 Read in the input table, mycas.sums. Specify END= to create a flag for the conditional IF statement.

10 Create a total sum of Homeowners by adding the partial sums from the first DATA step. Because this DATA step runs in a single thread, the total sum of all Homeowners is calculated.

11 Print the results showing the total sum.

Key Ideas

- There is no sharing between DATA step threads.
- Each thread has its own set of rows from the table and computes values based on those rows.
- This solution speeds processing by running as much as possible in multiple threads.
- When you use a retained variable to hold a sum in a multithreaded DATA step, the variable contains only the sum of the values that the processing thread can see.

See Also

- "RETAIN Statement" in *SAS DATA Step Statements: Reference*
- "Sum Statement" in *SAS DATA Step Statements: Reference*
- "END=variable" in *SAS DATA Step Statements: Reference*
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB Statement"_ALL_ " in *SAS Cloud Analytic Services: User's Guide*
- "Inter-row Dependencies" on page 11.

Add a Unique Row Identifier

Example Code

The _N_ automatic DATA step variable holds the number of iterations that a DATA step makes as it reads in rows from a table or other data source. Since the SAS DATA step usually reads rows sequentially, one row at a time, _N_ is often used to create row numbers for observations in a SAS data set.

This works well as long as the DATA step is running in a single thread. When the DATA step runs in multiple threads, there are multiple 'copies' or threads of the DATA step running on a single table simultaneously. The _N_ values are repeated for every DATA step thread. Therefore, the _N_ value alone cannot be used to uniquely identify rows with multithreaded DATA step processing.

The following example shows how you can use the _N_ and _THREADID_ automatic variables to create unique row numbers for a table that is processed in multiple DATA step threads.

For information about how to run the examples, see Set Up Code For Examples on page 32.

First, data is created for the example by loading the **Sashelp.Cars** data set to an in-memory CAS table. For simplicity, the KEEP statement is used to limit the number of variables that are included in the output table.

```
/* Create a subset of the Cars data set and load it to CAS */
data mycas.cars1;
   set sashelp.cars;
   keep make type;
run;
```

In the next DATA step, the _N_ automatic variable is stored in the variable **rowID**. **RowID** counts the iterations within each DATA step thread. The _THREADID_ automatic variable is stored in the variable **thread_ID**. The _THREADID_ automatic variable assigns a number to each thread that the DATA step is running in.

```
data mycas.cars2;
   set mycas.cars1;
   rowID = _n_;
   threadID = _threadid_;
run;
proc print data=mycas.cars2; run;
```

Notice how the _N_ value numbering repeats, starting again at 1 for the rows where **thread_ID** changes. Because the _N_ values are repeated across multiple threads, they are not unique table row numbers. Row 1, for example, has the same _N_ value as rows 55, row 109, and so on. The _THREADID_ automatic variable is also not unique since there are multiple rows assigned to a single thread. For example, rows 1 through 54 are all being processed by the same thread with an ID of 1, and rows 55 through 108 are running in thread 33.

Output 2.8 *Partial PROC PRINT Output for Cars1 Showing How _N_ and _THREADID_ Values Are Not Unique in the Distributed Table*

Obs	Make	Type	rowID	threadID
1	Acura	SUV	1	1
2	Audi	Sedan	2	1
3	Audi	Sedan	3	1
54	Volvo	Sedan	54	1
55	Acura	Sedan	1	33
56	Audi	Sedan	2	33
57	Audi	Sedan	3	33
108	Volvo	Sedan	54	33
109	Acura	Sedan	1	65
110	Audi	Sedan	2	65
111	Audi	Sedan	3	65

To create a unique row identifier for every row in the table, you can concatenate the values for _N_ with the values for _THREADID_:

```
data mycas.cars2;
   set mycas.cars1;
   rowID = _n_;
   threadID = _threadid_;
   uniqueRowID = _n_ + (_threadid_ * 1E4);
run;
proc print data=mycas.cars2; run;
```

Here is the same example with a BY group defined:

```
data mycas.cars3;
   set mycas.cars1;
   rowID = _n_;
   threadID = _threadid_;
   uniqueRowID = _n_ + (_threadid_ * 1E4);
   by make;
run;
proc print data=mycas.cars3(obs=20); run;
```

Output 2.9 *Partial PROC PRINT Output for Cars3 Table Showing Unique Row Identifiers with BY Groups*

Obs	Make	Type	rowID	threadID	uniqueRowID
1	Saturn	Sedan	1	8	80001
2	Saturn	Sedan	2	8	80002
3	Saturn	Sedan	3	8	80003
4	Saturn	Wagon	4	8	80004
5	Saturn	Sedan	5	8	80005
6	Saturn	Sedan	6	8	80006
7	Saturn	Sedan	7	8	80007
8	Saturn	SUV	8	8	80008
9	Saab	Sedan	1	14	140001
10	Saab	Sedan	2	14	140002
11	Saab	Wagon	3	14	140003
12	Saab	Sedan	4	14	140004
13	Saab	Sedan	5	14	140005
14	Saab	Sedan	6	14	140006
15	Saab	Sedan	7	14	140007
16	Subaru	Sports	1	19	190001
17	Subaru	Sedan	2	19	190002
18	Subaru	Truck	3	19	190003
19	Subaru	Sedan	4	19	190004
20	Subaru	Wagon	5	19	190005

Key Ideas

- The _THREADID_ automatic variable can be combined with the _N_ automatic variable to create a unique row identifier in a multithreaded DATA step.
- The _N_ automatic DATA step variable holds the number of iterations that a DATA step makes as it reads in rows from a table or other data source.
- The _N_ automatic variable is often used to add row numbers to observations in SAS data sets.
- The _N_ automatic variable values alone are not unique in a multithreaded DATA step and cannot be used to uniquely identify rows in a table.
- The _THREADID_ automatic variable returns the number that is associated with the thread that the DATA step is running in a server session.
- There is no sharing between DATA step threads in a multithreaded DATA step. Each thread has its own set of rows from the table and computes values based on those rows.

See Also

- "_N_" on page 23
- "_THREADID_" on page 25

Create a VARCHAR Variable Using the LENGTH Statement

Example Code

In the following example, the first LENGTH statement creates a VARCHAR variable, **x**, with a length of 30 characters. The second LENGTH statement creates a CHAR variable, **y**, with a length of 30 bytes.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
data mycas.string;
   length x varchar(30);
   length y $30;
   x = 'abc'; y = 'def';
run;

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

Engine/Host Dependent Information	
Data Limit	100MB
Caslib	CASUSER (sasdemo)
Scope	Session

Alphabetic List of Variables and Attributes					
			Len		Max Bytes Used
#	Variable	Type	Bytes	Chars	
1	x	Varchar	30	30	3
2	y	Char	30		

If you read the table into a SAS data set and then run PROC CONTENTS on the output data set, you can see how SAS converts the variable.

```
data string;
   set mycas.string;
run;
proc contents data=string; run ;
```

When SAS converts a VARCHAR to a CHAR in automatic data type conversion, SAS determines the length of the CHAR variable based on the SAS session encoding. If the local session encoding uses a single-byte character set, then SAS multiplies the original length by 1 to get the length of the converted CHAR variable. The length is multiplied by 1 in the calculation because 1 is the largest possible value for any character in a single-byte character set.

Output 2.10 *PROC CONTENTS Output When the Local SAS Session Encoding Is Single Byte (Latin1)*

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	x	Char	30
2	y	Char	30

If the local session encoding is a multi-byte encoding such as UTF-8, then SAS multiplies the length times the longest possible value of any character in the character set. The example here shows the length of the converted VARCHAR when the local session encoding is UTF-8. Because 4 bytes is the largest possible value for any character in UTF-8, the length is calculated as the original length, 30, times 4 (120).

Output 2.11 *PROC CONTENTS Output When the Local SAS Session Encoding Is Multi-byte (UTF-8)*

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	x	Char	120
2	y	Char	30

Key Ideas

- The VARCHAR data type is not supported by the SAS V9 engine. Therefore, to store a VARCHAR variable in a table, you must use a CAS engine libref on the output table.
- To use the CAS engine with the DATA step, specify the CAS libref on the output data set. The example, "Convert a CAS Table to a SAS Data Set" on page 37, shows how to use the CAS engine to create a CAS table as output.
- The length of VARCHAR variables is determined based on the number of characters that the string contains. The length of CHAR variables is determined based on the number of bytes that the characters in the string requires.
- It is usually better to declare character strings as VARCHAR types than to declare them as CHAR types to avoid truncation of data in columns. Truncation can occur, for example, when multi-byte characters are stored in a CHAR data type. For example, if the DATA step converts character variables from UTF-8 to Wlatin1, then the variable length (in bytes) might not be long enough and values are truncated.
- If the data is consistently short, such as in an ID column or in two-letter state abbreviations, consider using a VARCHAR with a fixed length 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

- "Avoiding Data Truncation" on page 26
- "VARCHAR Data Type" in *SAS Cloud Analytic Services: User's Guide*
- "Create a VARCHAR Variable Using the LENGTH Statement" on page 48.
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB Statement"_ALL_ " in *SAS Cloud Analytic Services: User's Guide*

Impute Missing Values Using the LAG Function

Example Code

The following example uses the LAG function to impute missing values in data. The missing price is imputed based on a 10% increase from the price of the previous year. If the price of the previous year is also missing, then the missing price is imputed based on a 20% increase from the price of the previous year.

The following data set Mycas.Example is first created from raw input data using the DATALINES statement and written out to an in-memory CAS table. The LAG function compares values for PRICE between the rows in the Mycas.Example table. Therefore, in a distributed CAS server, where rows are located on separate machines, the LAG function does not perform as expected. To use the LAG function in a distributed CAS server, specify the SINGLE=YES statement to ensure that the data is confined to a single node.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
data mycas.example;
input county $ year price;
datalines;
1001 2001 200000
1001 2002 .
```

```
1001 2003 .
1001 2004 280000
1001 2005 310000
1002 2001 .
1002 2002 240000
1002 2003 270000
1002 2004 300000
1002 2005 340000
1003 2001 280000
1003 2002 300000
1003 2003 330000
1003 2004 370000
1003 2005 .
;

data mycas.example2 (drop=lag_price lag2_price lag_county lag2_county)
                     / single=yes;
  set mycas.example;
  by county year;
   lag_price = lag(price);
   lag2_price = lag2(price);
   lag_county = lag(county);
   lag2_county = lag2(county);
    if price NE . then price_impute = price;
    else if price = . and county = lag_county and lag_price ne .
       then price_impute = lag_price*1.1;
          else if price = . and lag_price = . and county = lag2_county
           and lag2_price NE . then price_impute = lag2_price*1.2;
run;
proc print data=mycas.example2;
run;
```

Obs	county	year	price	price_impute
1	1001	2001	200000	200000
2	1001	2002	.	220000
3	1001	2003	.	240000
4	1001	2004	280000	280000
5	1001	2005	310000	310000
6	1002	2001	.	.
7	1002	2002	240000	240000
8	1002	2003	270000	270000
9	1002	2004	300000	300000
10	1002	2005	340000	340000
11	1003	2001	280000	280000
12	1003	2002	300000	300000
13	1003	2003	330000	330000
14	1003	2004	370000	370000
15	1003	2005	.	407000

Key Ideas

- For more information about multithreaded processing and inter-row dependencies, see "Inter-row Dependencies".
- For information about the LAG function, see "LAG Function" in *SAS Functions and CALL Routines: Reference*.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB _ALL_ statement
- "LAG Function" in *SAS Functions and CALL Routines: Reference*
- "Inter-row Dependencies"

Create a DATA Step View

Example Code

The following example shows how to create a DATA step view. First, an in-memory CAS table named Mycas.Class is created from the Class data set in the Sashelp library. The SET statement selects all rows where **Sex=M**. A VARCHAR variable, **Note**, is created so that any Males who are 12 years old and taller than 60 inches are highlighted. See "Key Ideas" on page 54 for more information about DATA step views in CAS.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
data mycas.class;                        /* 1 */
   set sashelp.class(where=(sex='M'));   /* 2 */
   length Note varchar(10);              /* 3 */
   if age="12" and height>60 then do;    /* 4 */
     Note="stature";                     /* 5 */
   end;
run;

data BMI / view=BMI;                     /* 6 */
   set mycas.class;                      /* 7 */
   BMI = weight / height**2 * 703;       /* 8 */
   format BMI 6.2;                       /* 9 */
run;
proc print data=BMI; run;                /* 10 */
```

1 Create an in-memory CAS table named Mycas.Class.

2 Use the Class data set in the Sashelp library as the input data. The SET statement selects all rows where Sex=M using the WHERE statement. The rows are selected from the input data set before the output table is created in CAS.

3 Create a VARCHAR variable with a length of 10 characters.

4 Use a subsetting IF statement to select rows where Age =12 and Height is greater than 60.

5 If Age=12 and Height is greater than 60, then set the Note variable to "stature."

6 Create a SAS DATA step view named BMI from the Mycas.Class table. The VIEW= option in the DATA statement defines the view.

7 Read the Mycas.Class table using the SET statement.

8 Calculate the BMI based on the Weight and Height variable values and store the result in the variable BMI.

9 Format the BMI value using the FORMAT statement.

10 Print the DATA step view using the PRINT procedure.

Here is the log and PROC PRINT output.

```
NOTE: DATA STEP view saved on file WORK.BMI.
NOTE: A stored DATA STEP view cannot run under a different operating system.
NOTE: DATA statement used (Total process time):
      real time           0.00 seconds
      cpu time            0.01 seconds
```

Obs	Name	Sex	Age	Height	Weight	Note	BMI
1	Alfred	M	14	69.0	112.5		16.61
2	Robert	M	12	64.8	128.0	stature	21.43
3	Henry	M	14	63.5	102.5		17.87
4	Ronald	M	15	67.0	133.0		20.83
5	James	M	12	57.3	83.0		17.77
6	Thomas	M	11	57.5	85.0		18.07
7	Jeffrey	M	13	62.5	84.0		15.12
8	William	M	15	66.5	112.0		17.80
9	John	M	12	59.0	99.5		20.09
10	Philip	M	16	72.0	150.0		20.34

To see how the SAS DATA step view handles the VARCHAR variable, the CONTENTS procedure is specified:

```
proc contents data=BMI;
run;
```

The CONTENTS Procedure

Data Set Name	WORK.BMI	Observations	.
Member Type	VIEW	Variables	7
Engine	SASDSV	Indexes	0
Created	08/14/2017 14:08:39	Observation Length	51
Last Modified	08/14/2017 14:08:39	Deleted Observations	0
Protection		Compressed	NO
Data Set Type		Sorted	NO
Label			
Data Representation	Default		
Encoding	Default		

Engine/Host Dependent Information	
DATA Step view type	INPUT

Alphabetic List of Variables and Attributes

#	Variable	Type	Len	Format
3	Age	Num	8	
7	BMI	Num	8	6.2
4	Height	Num	8	
1	Name	Char	8	
6	Note	Char	10	
2	Sex	Char	1	
5	Weight	Num	8	

Key Ideas

- You can use the DATA step to create a SAS view from a CAS table. When using a CAS table as input, you must run the DATA step locally, in a Base SAS session and not in a CAS server session.
- Views can be stored only in a Base SAS library (a library created using the V9 engine). The VIEW option in the DATA statement is not supported with the DATA step in SAS Cloud Analytic Services. This means that you cannot use a DATA step that is running in CAS to create a SAS view. However, you can create a SAS view using a DATA step that is running in SAS and that is using a CAS table as input.
- Views that contain VARCHAR variables can be stored in a Base SAS library. The SAS DATA step can read and return VARCHAR variable values that are stored in SAS views. However, the DATA step cannot read and return VARCHAR variables that are contained in CAS tables or other data sources.
- A DATA step view can include VARCHAR variables but a SAS data set cannot.
- Even though a view is stored in a Base engine library, views can read data sets stored in non-Base engine libraries. You can write a view that is stored in a Base engine library and accesses data sets in a CAS engine library.
- Update views are not supported.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB Statement "_ALL_ " in *SAS Cloud Analytic Services: User's Guide*
- "SAS Views" in *SAS Language Reference: Concepts*

Create a VARCHAR Using an ARRAY Statement

Example Code

In the following example, three VARCHAR variables are created with a maximum length for the array, Test.

For information about how to run the examples, see Set Up Code For Examples on page 32.

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

Output 2.12 *Log Output for Mycas.Test Table*

```
NOTE: Running DATA step in Cloud Analytic Services.
NOTE: The DATA step has no input data set and will run in a single thread.
a
b
c
```

Key Ideas

- For more information about array processing in the DATA step, see "Array Processing" in *SAS Language Reference: Concepts*.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB Statement "_ALL_ " in *SAS Cloud Analytic Services: User's Guide*
- "Data Types" in *SAS Cloud Analytic Services: User's Guide*

Index CHAR and VARCHAR Character Strings

Example Code

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 ('abc'). The INDEX function returns the position in the string that the target value first appears. 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.

For information about how to run the examples, see Set Up Code For Examples on page 32.

Note: Assume that the local SAS session encoding is UTF-8.

```
data mycas.string / sessref=casauto;
   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;
```

Output 2.13 *Log Output*

```
xi = 2
yi = 2
```

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 = 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 measures data in terms of characters and the CHAR data type measures data in terms of bytes.

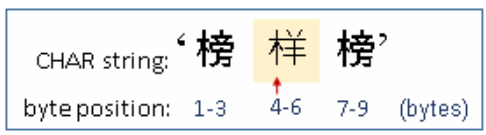
```
data mycas.string2 / sessref=casauto;
  length x varchar(10);
  length y $10;
  x = '榜样榜';  y = '榜样榜';
  xi = index(x,'样');
  yi = index(y,'样');
  put "xi = " xi;
  put "yi = " yi;
run;
```

Output 2.14 *Log Output*

```
xi = 2
yi = 4
```

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

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



Key Ideas

- The INDEX function uses character-based positional indexing on VARCHAR variables, and it uses byte-based indexing on CHAR 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');**. The function returns h = 3, because the first letter of the "**hat**" string is the third from the left starting with 1.
- The VARCHAR data type is not supported by the SAS V9 engine. Therefore, to store a VARCHAR variable in a table, you must use a CAS engine libref on the output table.
- The VARCHAR data type 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 no more than 3 or 4 bytes.
- The length of a CHAR variable when it is converted from a VARCHAR variable is based on your local session encoding. For more information, see "Avoiding Data Truncation" on page 26.
- VARCHAR variables can increase memory use when the column contains only 2- or 3-digit integer values, such as a three-letter airport. For more information about when to use a VARCHAR data type, see "When to Use a VARCHAR Data Type" in *SAS Cloud Analytic Services: User's Guide*.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "_ALL_ " in *SAS Cloud Analytic Services: User's Guide*
- "CAS LIBNAME Statement" in *SAS Cloud Analytic Services: User's Guide*
- "INDEX Function" in *SAS Functions and CALL Routines: Reference*
- "Data Types" in *SAS Cloud Analytic Services: User's Guide*

Implicit Variable Type Conversion

Example Code

The following example shows how a VARCHAR variable is automatically converted to a CHAR type variable when it is saved as a SAS data set. The DATA step supports the processing of VARCHAR data, but only the CAS engine supports the VARCHAR data type. The first DATA step is shown for the purposes of creating data containing a VARCHAR for this example.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
data mycas.test;                                    /* 1 */
     length x varchar(*);
     length y varchar(20);
     x="hello";
     y="goodbye";
run;
proc contents data=mycas.test;
run;
```

```
data simple;                                          /* 2 */
   set mycas.test;
run;

proc contents data=simple;                            /* 3 */
run;
```

1 Create a CAS table that contains a VARCHAR variable for the example. The in-memory table, Mycas.Test, contains two VARCHAR variables.

2 Save the CAS table as a SAS data set. Note there is no CAS engine libref on the output data set.

3 View the contents of the data set to see how VARCHAR variables are converted to CHAR variables.

Alphabetic List of Variables and Attributes					
			Len		Max Bytes Used
#	Variable	Type	Bytes	Chars	
1	x	Varchar	.	.	5
2	y	Varchar	20	20	7

Alphabetic List of Variables and Attributes			
#	Variable	Type	Len
1	x	Char	5
2	y	Char	20

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.
```

Key Ideas

- You cannot create a VARCHAR variable using an assignment statement alone. SAS sets the type to CHAR if the value on the right side of the assignment is a character string. To create a VARCHAR variable, you must define it in either a LENGTH statement or in an ARRAY statement.
- 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 32,767 bytes.
- If a VARCHAR variable is defined with a specific length and then is converted to a CHAR, SAS determines the length for the CHAR based on the local SAS session encoding. SAS calculates the length of the converted VARCHAR based on the SAS session encoding. See "Avoiding Data Truncation" on page 26 and "VARCHAR Length with Implicit Type Conversion" in *SAS Cloud Analytic Services: User's Guide* for more information.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "_ALL_ " in *SAS Cloud Analytic Services: User's Guide*
- "VARCHAR Data Type" in *SAS Cloud Analytic Services: User's Guide*

Create a DATA Step Hash Object with VARCHAR Key and Data Variables

Example Code

In the following example, a hash object is created in which VARCHAR variables are defined for the KEY and DATA values. The DATA step processes in CAS. The LET macro statement substitutes the word **dataopts** for the SESSREF= option. When **&dataopts** is specified in the DATA statement, this causes the DATA step to run in CAS.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
%let dataopts=%str(/sessref=casauto);

/* Simple add and iterate */
data _null_ &dataopts;
  length vc1 vc2 varchar(*);

  dcl hash h();
  h.definekey('vc1');
  h.definedata('vc2');
  h.definedone();

  vc1 = 'abcd';    vc2 = 'ABCD';    h.add();
  vc1 = 'efghijk'; vc2 = 'EFGHIJK'; h.add();
  vc1 = 'lmn';     vc2 = 'LMN';     h.add();

  do vc1 = 'lmn', 'opqrst', 'efghijk', 'abcd';
    rc = h.find();
    if rc ^= 0 then
      put 'Didn''t find key ' vc1=;
    else
      put _all_;
  end;

  put '----------------------------';

  dcl hiter hi('h');
  rc = hi.next();
  do until (rc ^= 0);
    put _all_;
    rc = hi.next();
  end;
run;

/* Add with parameters */
data _null_ &dataopts;
  length vc1 vc2 varchar(*);
```

```
  length vc1b vc2b varchar(*);

  dcl hash h();
  h.definekey('vc1');
  h.definedata('vc2');
  h.definedone();

  vc1b = 'abcd';    vc2b = 'ABCD';    h.add(key:vc1b, data:vc2b);
  vc1b = 'efghijk'; vc2b = 'EFGHIJK'; h.add(key:vc1b, data:vc2b);
  vc1b = 'lmn';     vc2b = 'LMN';     h.add(key:vc1b, data:vc2b);

  do vc1b = 'lmn', 'opqrst', 'efghijk', 'abcd';
    rc = h.find(key:vc1b);
    if rc ^= 0 then
      put 'Didn''t find key ' vc1b=;
    else
      put _all_;
  end;

  put '-----------------------------';

  dcl hiter hi('h');
  rc = hi.next();
  do until (rc ^= 0);
    put _all_;
    rc = hi.next();
  end;
run;
```

Output 2.15 *Partial Log Output*

```
234        put 'Didn''t find key ' vc1=;
235      else
236        put _all_;
237    end;
238
239    put '-----------------------------';
240
241    dcl hiter hi('h');
242    rc = hi.next();
243    do until (rc ^= 0);
244      put _all_;
245      rc = hi.next();
246    end;
247  run;

NOTE: Running DATA step in Cloud Analytic Services.
vc1=lmn vc2=LMN rc=0 _ERROR_=0 _N_=1
Didn't find key vc1=opqrst
vc1=efghijk vc2=EFGHIJK rc=0 _ERROR_=0 _N_=1
vc1=abcd vc2=ABCD rc=0 _ERROR_=0 _N_=1
-----------------------------
vc1=abcd vc2=EFGHIJK rc=0 _ERROR_=0 _N_=1
vc1=abcd vc2=LMN rc=0 _ERROR_=0 _N_=1
NOTE: Duplicate messages output by DATA step:
vc1=lmn vc2=LMN rc=0 _ERROR_=0 _N_=1  (occurred 192 times)
Didn't find key vc1=opqrst  (occurred 192 times)
vc1=efghijk vc2=EFGHIJK rc=0 _ERROR_=0 _N_=1  (occurred 192 times)
vc1=abcd vc2=ABCD rc=0 _ERROR_=0 _N_=1  (occurred 384 times)
-----------------------------  (occurred 192 times)
vc1=abcd vc2=EFGHIJK rc=0 _ERROR_=0 _N_=1  (occurred 192 times)
vc1=abcd vc2=LMN rc=0 _ERROR_=0 _N_=1  (occurred 192 times)
NOTE: DATA statement used (Total process time):
      real time           0.36 seconds
      cpu time            0.06 seconds
```

Key Ideas

- You can use VARCHAR variables for KEY and DATA values when defining a hash object for a DATA step that is running in CAS or in SAS. VARCHAR variables are supported with all hash object methods where character values are supported.
- Ordered hashes are not supported in CAS (for example, **`dcl hash h(ordered:'yes');`** is not supported in the DATA step that is running in CAS.
- A fixed-width, 32-byte SHA-256 hash digest value is computed for VARCHAR keys.

See Also

- *SAS Component Objects: Reference*
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "_ALL_ " in *SAS Cloud Analytic Services: User's Guide*

Group Data

Group and Order a CAS Table Using the PARTITION= Data Set Option

Example Code

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

For information about how to run the examples, see Set Up Code For Examples on page 32.

Note: To use the ORDERBY= option, you must first specify the PARTITION= option.

```
proc casutil outcaslib='casuser';                          /*1*/
   load data=sashelp.baseball replace;
run;

data mycas.baseball (partition=(team) orderby=(nHome name)); /*2*/
   set mycas.baseball;
   keep team name nHome;
run;
proc print data=mycas.baseball;
run;
```

1 Load the Sashelp.Baseball data set to a distributed CAS table.

2 Use the PARTITION= data set option to partition the table by team. This groups the rows that contain the same value for **Team** together, onto the same server node. The ORDERBY= option then orders the rows within each partition by **nHome** and **Name**.

Output 2.16 *Partial Output for Partitioned and Ordered Baseball Table*

Obs	Name	Team	nHome
1	Quinones, Rey	Boston	2
2	Romero, Ed	Boston	2
3	Barrett, Marty	Boston	4
4	Boggs, Wade	Boston	8
5	Armas, Tony	Boston	11
6	Gedman, Rich	Boston	16
7	Buckner, Bill	Boston	18
8	Rice, Jim	Boston	20
9	Evans, Dwight	Boston	26
10	Baylor, Don	Boston	31
11	Cruz, Julio	Chicago	0
12	Cangelosi, John	Chicago	2
13	Guillen, Ozzie	Chicago	2
14	Bonilla, Bobby	Chicago	3
15	Tolleson, Wayne	Chicago	3
16	Dernier, Bob	Chicago	4
17	Boston, Daryl	Chicago	5
18	Hairston, Jerry	Chicago	5
19	Mumphrey, Jerry	Chicago	5
20	Skinner, Joel	Chicago	5

Note: There is no global ordering of rows across nodes or across partitions. You should not see rows for the same team spread across different nodes or different partitions. However, you might see the global order vary. The partitioning is done first, then the ordering is done within each partition.

Key Ideas

- A partition is a group of rows that share the same values for a specified variable, or a partition key. Rows that share the same partition key are arranged together on the same worker node. After the rows are partitioned, they are ordered within each node. Rows are not ordered globally across nodes unless you run the DATA step in a single thread. See "Run the DATA Step in a Single Thread" on page 69 to see an example of running the DATA step in a single thread.
- A CAS table can be partitioned using the PARTITION= data set option and ordered within each partition using the ORDERBY= option.
- You must specify the PARTITION= option before you can specify the ORDERBY= option.
- CAS partitions tables based on the formatted values of the PARTITION= variable (rather than the raw values). For example, a table containing dates that are formatted with the QTR format is divided into four partitions, one partition for each yearly quarter.
- If user-defined formats are used, then the format for the specified key variable must be available to the CAS server when the table is loaded into memory.

See Also

- "PARTITION= Data Set Option" in *SAS Cloud Analytic Services: User's Guide*
- "ORDERBY= Data Set Option" in *SAS Cloud Analytic Services: User's Guide*
- SAS Cloud Analytic Services: User-Defined Formats
- "CASLIB Statement" in *SAS Cloud Analytic Services: User's Guide*
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "_ALL_ " in *SAS Cloud Analytic Services: User's Guide*

Group and Order Rows Using the DATA Step BY Statement

Example Code

This example shows how to use the BY statement in a CAS DATA step to group and order variables in a table. The BY statement groups the table rows by **Make** and then orders the rows within each group by **Type**. The example also shows how to concatenate values from multiple rows into one row. The values for car **Type** are concatenated into a single variable so that they can be grouped into one row. In this way, all the CarStyles for each make of car are listed together in one row, in order.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
proc casutil;                                                  /*1*/
   load data=sashelp.cars
   outcaslib='casuser'
   replace;
run; quit;

data mycas.cars2;
  length CarStyles varchar(*);                                 /*2*/
  retain CarStyles;                                            /*3*/
  set mycas.cars;
  by make type;                                                /*4*/
     if first.make then CarStyles = trim(type);                /*5*/
           else CarStyles = CarStyles || "," || trim(type);    /*6*/
     if last.make then output;                                 /*7*/
     keep make CarStyles;                                      /*8*/
run;

proc print data=mycas.cars2; var Make CarStyles;               /*9*/
run;
```

1 Load the Sashelp.Cars data set to CAS.

2 Create a new variable, **CarStyles**, to be the aggregation variable to hold the concatenated values from other rows.

3 Retain the values for **CarStyles**, which are set equal to the existing variable **Type**. **Type** represents the style of car (SUV, wagon, and so on). For more information about retaining variables in the DATA step, see "RETAIN Statement" in *SAS DATA Step Statements: Reference*.

4 Create BY groups based on the Make of car and the style (type) of car. The values are grouped and ordered by the variables **Make** and **Type**.

5 If the DATA step encounters the first row in the BY group, then the variable **`CarStyles`** is set equal to the value for **`Type`**. The value stored in the variable **`CarStyles`** is retained from the previous row.

6 If the DATA step does not encounter the first or last row of the BY group, then the variable **`CarStyles`** equals the retained value from the previous row, **`Type`**. These values are separated by concatenating a comma to the end of the first value. The TRIM function removes trailing blanks.

7 If the DATA step encounters the last row of the BY group, then execute the OUTPUT statement and write the contents of the current row to the output table.

8 Keep only the variables **`Make`** and **`CarStyles`**. The variable **`Type`** is not needed because it is now stored in the variable **`CarStyles`**.

9 Print the output.

Output 2.17 *Partial PROC PRINT Output for the Mycas.Cars2 Data Set*

Obs	Make	CarStyles
1	BMW	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,Sports,Sports,Sports,SUV,SUV,Wagon
2	Isuzu	SUV,SUV
3	Jaguar	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,Sports,Sports,Sports
4	Cadillac	Sedan,Sedan,Sedan,Sedan,Sports,SUV,SUV,Truck
5	Saturn	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,SUV,Wagon
6	MINI	Sedan,Sedan
7	Dodge	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,SUV,Truck,Truck,Truck
8	Infiniti	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Wagon,Wagon
9	Buick	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,SUV,SUV
10	Volkswagen	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,SUV,Wagon,Wagon,Wagon
11	Pontiac	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,SUV,Wagon
12	Land Rover	SUV,SUV,SUV
13	Mercedes-Benz	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,Sports,Sports,Sports,Sports,SUV,SUV,Wagon,Wagon,Wagon
14	Hyundai	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,SUV
15	Nissan	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,Sports,SUV,SUV,SUV,Truck,Truck,Wagon
16	Suzuki	Sedan,Sedan,Sedan,Sedan,Sedan,SUV,SUV,Wagon
17	Saab	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Wagon
18	Honda	Hybrid,Hybrid,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,SUV,SUV,SUV
19	Mercury	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,SUV,Wagon
20	Audi	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,Sports,Sports,Sports,Wagon,Wagon
21	Scion	Sedan,Wagon
22	Oldsmobile	Sedan,Sedan,Sedan
23	Lincoln	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,SUV,SUV
24	Chrysler	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,Wagon
25	Chevrolet	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,Sports,SUV,SUV,SUV,SUV,Truck,Truck,Truck,Truck,Truck,Wagon
26	GMC	Sedan,SUV,SUV,SUV,Truck,Truck,Truck,Truck
27	Jeep	SUV,SUV,SUV
28	Acura	Sedan,Sedan,Sedan,Sedan,Sedan,Sports,SUV
29	Porsche	Sports,Sports,Sports,Sports,Sports,Sports,SUV
30	Mitsubishi	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,Sports,Sports,SUV,SUV,SUV,Wagon
31	Toyota	Hybrid,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,Sports,SUV,SUV,SUV,SUV,SUV,Truck,Truck,Truck,Wagon
32	Hummer	SUV
33	Lexus	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sports,SUV,SUV,SUV,Wagon
34	Volvo	Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,Sedan,SUV,Wagon,Wagon

Key Ideas

- In SAS, when using the BY statement in a DATA step, you must first sort the data. This requirement is also true for a DATA step that is running in a single thread in CAS.
- You do not have to pre-sort data to use the BY statement in a multithreaded CAS DATA step. CAS performs implicit ordering on BY variables. For example, the following DATA step is valid and pre-sorting is not required.
- On the CAS server, there is no guarantee of global ordering between BY groups. Each DATA step thread can group and order only the rows within each BY group. When a BY group is defined, the DATA step runs in multiple threads, one thread per BY group. Ordering is done only within each BY group. For more information about BY-group processing with the DATA step in CAS, see "How CAS Groups Data with BY Variables" on page 14.

See Also

- "How CAS Groups Data with BY Variables" on page 14
- "BY Statement" in *SAS DATA Step Statements: Reference*
- "RETAIN Statement" in *SAS DATA Step Statements: Reference*
- "IF-THEN/ELSE Statement" in *SAS DATA Step Statements: Reference*
- "How SAS Identifies the Beginning and End of a BY Group" in *SAS DATA Step Statements: Reference*
- "Combining Multiple Observations and Grouping Them Based on One BY Value" in *SAS DATA Step Statements: Reference*
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB _ALL_ statement

Group Data by Two Variables

Example Code

This example shows how to group a table by two variables and keep only the unique rows in each group. The example first concatenates the values from two variables into a single new string variable, then orders by that new variable, keeping only the first in each group.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
proc casutil;                                          /*1*/
   load data=sashelp.cars
   outcaslib='casuser'
   replace;
run; quit;

data mycas.grouped;                                    /*2*/
 set mycas.cars;
 keep make cylinders groupkey;
 groupkey = cats('make', make, 'cyl', cylinders);      /*3*/
run;

data mycas.final;                                      /*4*/
```

```
   set grouped;
   by groupkey;                                    /* 5 */
   if first.groupkey then output;                  /* 6 */
   drop groupkey;                                  /* 7 */
run;
```

1 Load the Sashelp.Cars data set to CAS.

2 Specify the loaded CAS table as the input table and create a new table called mycas.grouped.

3 Specify the CATS function to concatenate the values in the Make and Cyl columns and store them as a concatenated string in the variable Groupkey. The CATS function also removes leading and trailing blanks.

4 Run another DATA step to isolate the second group-by variable.

5 Group the table rows by the variable Groupkey. Note: When running the DATA step in CAS, you do not have to pre-sort the data before grouping using the BY statement.

6 Specify the FIRST. automatic variable in an IF statement to identify the beginning of each BY-group. Specify the OUTPUT statement to write only the first row of each group to the output table.

7 Drop the Groupkey column.

Key Ideas

- In SAS, when using the BY statement in a DATA step, you must first sort the data. This requirement is also true for a DATA step that is running in a single thread in CAS.
- You do not have to pre-sort data to use the BY statement in a multithreaded CAS DATA step. CAS performs implicit ordering on BY variables.
- On the CAS server, there is no guarantee of global ordering between BY groups. Each DATA step thread can group and order only the rows within each BY group. When a BY group is defined, the DATA step runs in multiple threads, one thread per BY group. Ordering is done only within each BY group. For more information about BY-group processing with the DATA step in CAS, see "How CAS Groups Data with BY Variables" on page 14.

See Also

- "How CAS Groups Data with BY Variables" on page 14
- "BY Statement" in *SAS DATA Step Statements: Reference*
- "RETAIN Statement" in *SAS DATA Step Statements: Reference*
- "IF-THEN/ELSE Statement" in *SAS DATA Step Statements: Reference*
- "How SAS Identifies the Beginning and End of a BY Group" in *SAS DATA Step Statements: Reference*
- "Combining Multiple Observations and Grouping Them Based on One BY Value" in *SAS DATA Step Statements: Reference*
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB _ALL_ statement

Score an In-Memory CAS Table

Example Code

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

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
proc casutil;                              /* 1 */
   load data=sashelp.baseball replace;
run;
data mycas.baseball / sessref=casauto;     /* 2 */
   set mycas.baseball(where=(team='Boston'));
   if nHits > 125
     then score=1;                         /* 3 */
   keep Name Team nHits score;
run;

proc print data=mycas.baseball; run;       /* 4 */
```

1 Load the Sashelp.Baseball data set into CAS as an in-memory CAS table.

2 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.

3 Execute the score code.

4 Run the PRINT procedure to view the output table showing the scored values.

Obs	Name	Team	nHits	score
1	Buckner, Bill	Boston	168	1
2	Romero, Ed	Boston	49	.
3	Gedman, Rich	Boston	119	.
4	Rice, Jim	Boston	200	1
5	Quinones, Rey	Boston	68	.
6	Armas, Tony	Boston	112	.
7	Boggs, Wade	Boston	207	1
8	Evans, Dwight	Boston	137	1
9	Barrett, Marty	Boston	179	1
10	Baylor, Don	Boston	139	1

Key Ideas

- When you run the DATA step in the CAS server, you are running it in multiple threads on in-memory data, which enables faster processing.
- For a DATA step to run in CAS, you must make sure you have met the requirements for processing in CAS on page 4.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- CASLIB "_ALL_ " in *SAS Cloud Analytic Services: User's Guide* statement
- "CASUTIL Procedure" in *SAS Cloud Analytic Services: User's Guide*
- "WHERE Statement" in *SAS DATA Step Statements: Reference*

Run the DATA Step in a Single Thread

Example Code

The following example shows how to use the SINGLE= option to run the DATA step in a single thread. This option is useful for running functions that have inter-row dependencies.

```
data mycas.shoes / single=yes;                        /* 1 */
   set mycas.shoes;
   put "The number of Threads is " _nthreads_;        /* 2 */
run;
```

1 Using the Mycas.Shoes data set from the previous example, load the Mycas.Shoes table. Specify the SINGLE=YES option.

2 Specify the automatic variable, _NTHREADS_ with the PUT statement to verify the number of threads that are processing.

Output 2.18 *Partial Log Output*

```
NOTE: Running DATA step in Cloud Analytic Services.
The number of Threads is 1
```

Key Ideas

- The DATA step in CAS automatically runs in multiple threads on every available computer in your system by default. To run the DATA step in multiple threads, specify the SINGLE=YES option.
- A SAS DATA step (a DATA step that is not executing in CAS) always runs in a single thread.
- For more information about CAS sessions, see "Sessions" in *SAS Cloud Analytic Services: Fundamentals*.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- SESSOPTS=
- SINGLE= in *SAS DATA Step Statements: Reference*
- "Inter-row Dependencies" on page 11
- "DATA Statement" in *SAS DATA Step Statements: Reference*
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "_ALL_" in *SAS Cloud Analytic Services: User's Guide*

View DATA Step Processing Information Using Automatic Variables

Example Code

This first example uses the _THREADID_ automatic variable to get information about a multithreaded DATA step that contains two BY variables. When the DATA step runs in CAS, the table is grouped and distributed by the first BY variable, **Make**. Each BY group is then ordered by the second BY variable, **Type**. The FIRST. automatic variable is used to show the distinct BY groups in the output table.

For information about how to run the examples, see Set Up Code For Examples on page 32.

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

data mycas.cars;
   set mycas.cars(where=(weight>4000));
   by make type;
   if first.make then first="BY Group";
   keep make type first threadid;
   threadid = _threadid_;
run;
```

```
proc print data=mycas.cars;
  title 'Cars By Make By Type - Multithreaded'; run; title;
run;
```

Cars By Make By Type - Multithreaded

Obs	Make	Type	first	threadid
1	BMW	Sedan	BY Group	2
2	BMW	Sedan		2
3	BMW	SUV		2
4	BMW	SUV		2
5	Isuzu	SUV	BY Group	3
6	Jaguar	Sedan	BY Group	11
7	Jaguar	Sports		11
8	Cadillac	Sedan	BY Group	20
9	Cadillac	SUV		20
10	Cadillac	SUV		20
11	Cadillac	Truck		20
12	Dodge	Sedan	BY Group	23
13	Dodge	SUV		23
14	Dodge	Truck		23
15	Infiniti	Wagon	BY Group	23
16	Infiniti	Wagon		23
17	Buick	SUV	BY Group	29
18	Buick	SUV		29
19	Volkswagen	Sedan	BY Group	32
20	Volkswagen	Sedan		32
21	Volkswagen	SUV		32
22	Volkswagen	Wagon		32
23	Pontiac	Sedan	BY Group	43
24	Land Rover	SUV	BY Group	44
25	Land Rover	SUV		44

You can also get information about the node (host) name and the number of threads the DATA step is running in by specifying the _HOSTNAME_ and _NTHREADS_ automatic variables. In the following example, the host name and thread count results are printed in the SAS log.

```
data mycas.cars;
   set mycas.cars;
   put _hostname_;
   put _nthreads_;
run;
```

Output 2.19 *Log Output for View DATA Step Processing Information Using Automatic Variables*

```
1          OPTIONS NONOTES NOSTIMER NOSOURCE NOSYNTAXCHECK;
 72
 73         data mycas.cars;
 74         set mycas.cars;
 75         put _hostname_;
 76         put _nthreads_;
 77         run;

 NOTE: Running DATA step in Cloud Analytic Services.
 NOTE: The DATA step will run in multiple threads.
 srvrnode002
 srvrnode005
 srvrnode007
 192
 srvrnode003
 srvrnode004
 srvrnode006
 NOTE: Duplicate messages output by DATA step:
 srvrnode002  (occurred 22 times)
 srvrnode005  (occurred 27 times)
 srvrnode007  (occurred 12 times)
 192  (occurred 103 times)
 srvrnode003  (occurred 17 times)
 srvrnode004  (occurred 14 times)
 srvrnode006  (occurred 11 times)
 NOTE: There were 103 observations read from the table CARS in caslib
CASUSER(sasdemo).
 NOTE: The table cars in caslib CASUSER(sasdemo) has 103 observations and 4
variables.
 NOTE: DATA statement used (Total process time):
       real time           0.42 seconds
       cpu time            0.03 seconds
```

Key Ideas

- A DATA step that has an input table automatically runs in multiple threads in CAS. This is the case even in a single-machine CAS environment. By default, the DATA step runs in all available threads on every computer node in the cluster.
- Single-threaded DATA step processing is useful on small- to medium-sized tables where performance is not an issue and where preserving order or maintaining inter-row dependencies is important.
- When the DATA step runs in CAS, the same DATA step program is replicated across the CAS cluster. On each node in the cluster, the DATA step runs multiple threads. Each thread runs on only a portion of the table.
- There are restrictions on what language elements you can use in a DATA step running in CAS. For information about these restrictions, see "Restrictions" on page 18.

See Also

- "Automatic Variables" on page 23
- "DATA Statement" in *SAS DATA Step Statements: Reference*

- "Controlling Threaded Processing" on page 10
- "About SAS Cloud Analytic Services Sessions" in *SAS Cloud Analytic Services: Fundamentals*
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "_ALL_ " in *SAS Cloud Analytic Services: User's Guide*

Use the Macro Facility to Generate CAS DATA Step Code

Example Code

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. Macros compile on the SAS server.

For information about how to run the examples, see Set Up Code For Examples on page 32.

```
data mycas.class;                                                /* 1 */
   set sashelp.class; run;
data mycas.cars;
   set sashelp.cars; run;
data mycas.air;
   set sashelp.air; run;
libname mysas "/u/file-path/mycas/";                             /* 2 */
%macro load(tables=);                                            /* 3 */
%let n=%sysfunc(countw(&tables,%str( )));                        /* 4 */
%do i=1 %to &n;                                                  /* 5 */
   %let dsn=%scan(&tables,&i);
   data mysas.&dsn;                                              /* 6 */
      set mycas.&dsn;
   run;
%end;
%mend;                                                           /* 7 */
%load(tables=class cars air);
```

1. Use the DATA step to load three SASHELP data sets to CAS.
2. Create a SAS libref.
3. Start the macro definition.
4. Set *n* equal to the number of tables.
5. Loop through *n* times (for each table).
6. Create a SAS data set from each CAS table.
7. End macro definition.

Log 2.2 *Partial Log Output for Macro*

```
NOTE: Libref MYSAS was successfully assigned as follows:
      Engine:        V9
      Physical Name: /file-path/mycas

 NOTE: There were 19 observations read from the data set MYCAS.CLASS.
 NOTE: The data set MYSAS.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 MYSAS.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 MYSAS.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 Macro Language: Reference*
- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "_ALL_ " in *SAS Cloud Analytic Services: User's Guide*

DATA Step Program Walk-through

Example Code

The following example shows the steps that you need to perform to run the DATA step in CAS.

For information about how to run the examples, see Set Up Code For Examples on page 32.

1. **Start a CAS session**.

```
NOTE: 'CASUSER(sasdemo)' is now the active caslib.
NOTE: The CAS statement request to update one or more session options for
session CASAUTO
      completed.
386  caslib _all_ assign;
NOTE: A SAS Library associated with a caslib can only reference library member
names that conform
      to SAS Library naming conventions.
NOTE: CASLIB CASUSER(sasdemo) for session CASAUTO will be mapped to SAS Library
CASUSER.
NOTE: CASLIB CASUSERHDFS(sasuser) for session CASAUTO will be mapped to SAS
Library CASUSERH.
387  libname mycas cas;
NOTE: Libref MYCAS was successfully assigned as follows:
      Engine:        CAS
```

The SESSOPTS= option in the CAS statement is used with the CASLIB= session option to ensure that the CASUSER personal caslib is set as the active caslib.

The CASLIB _ALL_ ASSIGN maps caslibs to SAS libraries. Caslibs provide a way to access in-memory tables and an associated data source. In this example, the personal caslib CASUSER is being used, so no CASLIB statement is needed. For more information about the CASUSER and CASUSERHDFS personal caslibs, see Personal, Predefined, and Custom Caslibs in SAS Cloud Analytic Services: Fundamentals:"Caslibs, Files, and Tables" in *SAS Cloud Analytic Services: Fundamentals*.

The LIBNAME CAS statement creates a SAS library that is associated with the CAS engine. Mycas is the CAS engine libref.

2. **Load data to CAS**.

   Use the CASUTIL procedure to load data to CAS. The data is loaded from the Sashelp library in the local SAS client session into an in-memory table in CAS. The DATA step processes in SAS. The output table, Cars, is loaded into the CAS server as an in-memory table.

```
proc casutil;
   load data=sashelp.cars
   outcaslib='casuser'
   replace;
run; quit;
```

   Here is the log output:

```
NOTE: The UUID 'session-UUID' is connected using session CASAUTO.
318     load data=sashelp.cars
319     outcaslib='casuser'
320     replace;
NOTE: SASHELP.CARS was successfully added to the "casuser" caslib as "CARS".
```

3. **Run the DATA step in CAS**. The DATA step runs in CAS in multiple threads by default.

```
data mycas.cars2(promote=yes)/sessref=casauto;          /* 1 */
   set mycas.cars;                                      /* 2 */
      format Combined 2.;
      Combined = (MPG_Highway*.45) + (MPG_City*.55);    /* 3 */
      if type="SUV" and origin="USA";                   /* 4 */
```

```
      keep make type MPG_City MPG_Highway Combined;
      by make;
run;
title 'Combined Average MPG for SUVs in the USA By Make';
proc print data=mycas.cars2; run;                    /* 5 */
```

1 Run the DATA step in CAS and specify Cars2 as the output table. Promote the output table to global scope by specifying the PROMOTE= data set option. Global scope lets other sessions access the table.

2 Specify the in-memory table that you loaded in 2. Load data to CAS, on page 75 as the input table.

3 Create a new variable, Combined, to calculate the combined average miles per gallon for SUVs in the USA.

4 Specify the KEEP statement to keep some variables in the output table and drop the rest. Use a subsetting IF statement to subset the data and select only rows containing SUV for Type and USA for Origin.

5 Print the results.

Here is the PROC PRINT output.

Combined Average MPG for SUVs in the USA By Make

Obs	Make	Type	MPG_City	MPG_Highway	Combined
1	Cadillac	SUV	16	21	18
2	Cadillac	SUV	14	18	16
3	Saturn	SUV	21	26	23
4	Dodge	SUV	15	21	18
5	Buick	SUV	19	26	22
6	Buick	SUV	15	21	18
7	Pontiac	SUV	19	26	22
8	Mercury	SUV	16	21	18
9	Lincoln	SUV	13	18	15
10	Lincoln	SUV	13	18	15
11	Chevrolet	SUV	19	22	20
12	Chevrolet	SUV	14	18	16
13	Chevrolet	SUV	16	21	18
14	Chevrolet	SUV	14	18	16
15	GMC	SUV	16	19	17
16	GMC	SUV	13	17	15
17	GMC	SUV	15	19	17
18	Jeep	SUV	20	24	22
19	Jeep	SUV	16	19	17
20	Jeep	SUV	16	21	18
21	Hummer	SUV	10	12	11
22	Ford	SUV	18	23	20
23	Ford	SUV	10	13	11
24	Ford	SUV	15	20	17
25	Ford	SUV	15	19	17

Here is the log output:

```
348
349  data mycas.cars2(promote=yes)/sessref=casauto;
350     set mycas.cars;
351        format Combined 2.;
352        Combined = (MPG_Highway*.45) + (MPG_City*.55);
353        if type="SUV" and origin="USA";
354        keep make type MPG_City MPG_Highway Combined;
355        by make;
356  run;

NOTE: Running DATA step in Cloud Analytic Services.
NOTE: There were 428 observations read from the table CARS in caslib
CASUSER(sasdemo).
NOTE: The table cars2 in caslib CASUSER(sasdemo) has 25 observations and 5
variables.
NOTE: DATA statement used (Total process time):
      real time           0.34 seconds
      cpu time            0.03 seconds


357
358  title 'Combined Average MPG for SUVs in the USA By Make';
359  proc print data=mycas.cars2; run;

NOTE: PROCEDURE PRINT used (Total process time):
      real time           0.11 seconds
      cpu time            0.04 seconds
```

4. **Save the table and list information about the table.**

   The in-memory CAS table, Cars2, can be saved permanently to disk in the server. Use the SAVE statement in the CASUTIL procedure to save the table. Use the CONTENTS statement to view information about the table. The REPLACE statement enables you to run the same procedure again and overwrite the table without an error.

```
proc casutil;
   save casdata='cars2'
   outcaslib='casuser' replace;
   contents casdata='cars2';
run; quit;
```

The CASUTIL Procedure

Table Information for Caslib CASUSER([redacted])

Table Name	Number of Rows	Number of Columns	NLS encoding	Created	Last Modified	Promoted Table	Repeated Table	View	Source Name	Compressed
CARS2	25	5	utf-8	20Jul2017:00:09:30	20Jul2017:00:09:30	Yes	No	No	_T_HRVNY7CW_JJDTSFIW_WY2I2XV4NW	No

Detail Information for cars2 in Caslib CASUSER([redacted]).

Node	Number of Blocks	Active Blocks	Rows	Fixed Data size	Variable Data size	Blocks Mapped	Memory Mapped	Blocks Unmapped	Memory Unmapped	Blocks Allocated	Memory Allocated
ALL	20	10	25	1200	0	0	0	20	2400	0	0

Column Information for CARS2 in Caslib CASUSER([redacted])

Column	Label	Type	Length	Format Name	Formatted Length	Format Width	Format Decimal
Make		char	13		13	0	0
Type		char	8		8	0	0
MPG_City	MPG (City)	double	8		12	0	0
MPG_Highway	MPG (Highway)	double	8		12	0	0
Combined		double	8	F	2	2	0

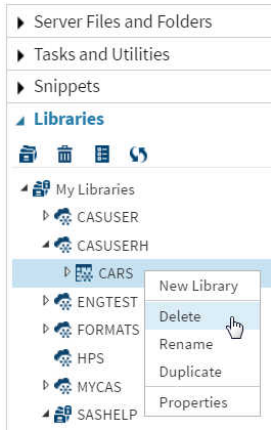
Here is the log output:

```
361  proc casutil;
NOTE: The UUID 'session-UUID' is connected using session CASAUTO.
362     save casdata='cars2'
363     outcaslib='casuser' replace;
NOTE: Cloud Analytic Services saved the file cars2.sashdat in caslib
CASUSER(sasdemo).
NOTE: The Cloud Analytic Services server processed the request in 0.293424
seconds.
364     contents casdata='cars2';
NOTE: Cloud Analytic Services processed the combined requests in 0.013627
seconds.
365  run;
```

5. **Drop the promoted table from memory**. You cannot replace a promoted table with the same name, so you must drop the table first if you want to replace it.

```
proc casutil;
   droptable casdata='cars2';
run; quit;
```

Note: You can also delete tables in SAS Studio by using the **Libraries tab** in the left pane. Right-click on the table that you want to delete, and select delete from the pop-up menu.



Here is the entire program together.

```
options cashost="srvrnode001" casport=10380;
cas casauto sessopts=(caslib='casuser');
libname mycas cas;
caslib _all_ assign;

proc casutil;
   load data=sashelp.cars
   outcaslib='casuser'
   replace;
run; quit;

data mycas.cars2(promote=yes)/sessref=casauto;
   set mycas.cars;
      format Combined 2.;
      Combined = (MPG_Highway*.45) + (MPG_City*.55);
      if type="SUV" and origin="USA";
      keep make type MPG_City MPG_Highway Combined;
```

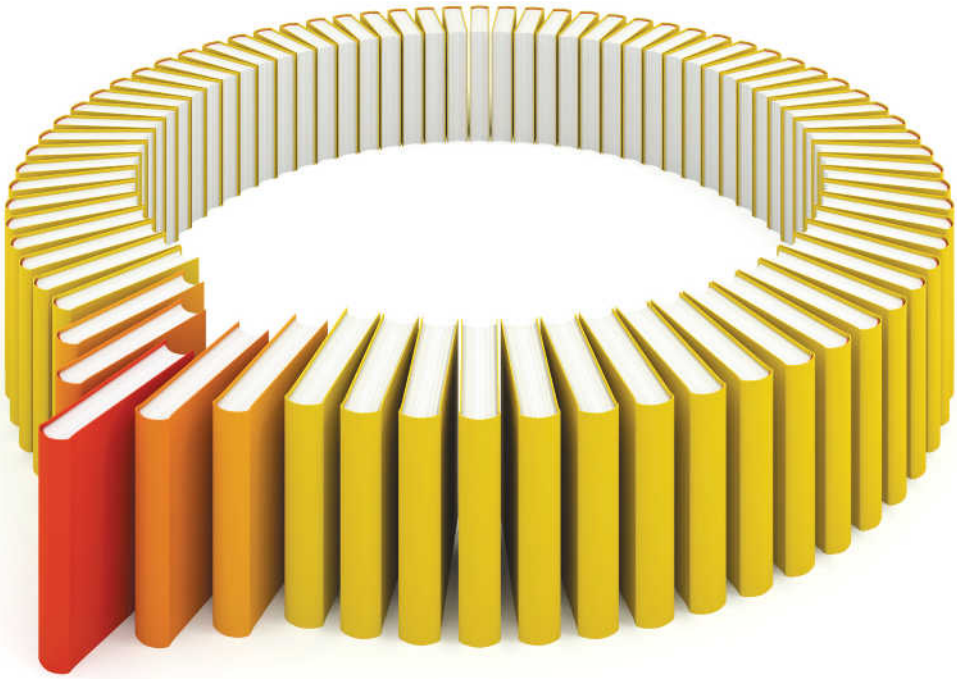
```
      by make;
run;

title 'Combined Average MPG for SUVs in the USA By Make';
proc print data=mycas.cars2; run;

proc casutil;
   save casdata='cars2'
   outcaslib='casuser' replace;
   contents casdata='cars2';
run; quit;

proc casutil;
   droptable casdata='cars2';
run; quit;
```

Key Ideas

- You can use many of the same statements that you use in a traditional SAS DATA step in CAS. For example, the SET, KEEP, and DROP statements perform the same functions in CAS as they do in a SAS DATA step. For a complete list of supported language elements, see "Language Elements Support" on page 17.
- When the DATA step runs in CAS, it runs where the tables are stored. Running a DATA step in CAS means that the DATA step code and all the statements within it are executing in a CAS server session. For information about how the DATA step runs in CAS, see "The DATA Step and CAS" on page 3.
- When the DATA step runs in CAS, it runs in all available threads by default. See "Controlling Threaded Processing" on page 10 for more information about controlling threaded processing.
- For a list of requirements for running the DATA step in CAS, see "Running the DATA Step in CAS" on page 4.
- You can use the CASUTIL procedure to do more than load your SAS data sets into CAS. For more information, see "CASUTIL Procedure" in *SAS Cloud Analytic Services: User's Guide*.

See Also

- "CAS Statement" in *SAS Cloud Analytic Services: User's Guide*
- "_ALL_" in *SAS Cloud Analytic Services: User's Guide*
- "Accessing Data" in *SAS Cloud Analytic Services: User's Guide*
- "CASUTIL Procedure" in *SAS Cloud Analytic Services: User's Guide*
- *An Introduction to SAS Viya Programming*



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.

sas
THE POWER TO KNOW.

SAS and all other SAS Institute Inc. product or service names are registered trademarks or trademarks of SAS Institute Inc. in the USA and other countries. ® indicates USA registration. Other brand and product names are trademarks of their respective companies. © 2013 SAS Institute Inc. All rights reserved. S107969US.0613