SAS4546-2020

Next Steps: Important Considerations for Moving Your Data and Formats into CAS

Kevin Russell, SAS Institute Inc.

ABSTRACT

Now that your SAS® Cloud Analytic Services (CAS) environment is available, you are ready to start moving your SAS data sets and formats from SAS® 9.4 to CAS. However, there are a number of considerations to think about before, during, and after you move your data and formats. This paper, written for the beginner CAS user, focuses on three main considerations in the context of examples that illustrate how to load SAS data sets and formats. The first consideration that is discussed is the likely change in character encoding from LATIN1 or WLATIN1 to UTF-8 and how to prevent possible truncation of data. Next, the advantages of loading data into CAS in parallel versus serially is discussed. After you load your data, you might see that the size of the loaded table is larger than the source data set. This paper explains how the increase in size affects memory and how to reduce the size of a loaded table. The most effective way to discuss these considerations is to put them in the context of examples that show how to move your SAS data sets and formats to CAS. This paper covers two main examples. The first example shows how to use the CASUTIL procedure to load a data set. The second example illustrates how to move a format catalog to CAS by using the CNTLOUT= and CNTLIN= options in the FORMAT procedure statement. This example also shows how to save the format so that the format is available for subsequent CAS sessions.

INTRODUCTION

CAS is the cloud-based, run-time environment for data management and analytics that is available in SAS® Viya®. The CAS platform is designed for high-performance analytics by processing data in memory and across multiple worker nodes. The CAS platform enables you to produce fast and accurate results, no matter how complex your analytical workload is.

CAS can run either on a single machine (SMP mode) or on multiple machines (MPP mode) as a distributed server. With either configuration, the servers can be on-site, be in the cloud, or be a combination of both on-site and cloud configurations. SMP mode consists of just one machine running CAS. SMP mode can perform analytics on multiple processors without having to make an investment in multiple machines or having all machines running in a cloud environment. In MPP mode, CAS uses a distributed server comprising a session controller and multiple worker nodes. The controller node communicates with client applications and controls the processing that is performed by the worker nodes. When a request is made, the data is distributed across the worker nodes and the code works with the data that is in memory on that node. (See Figure 1.)

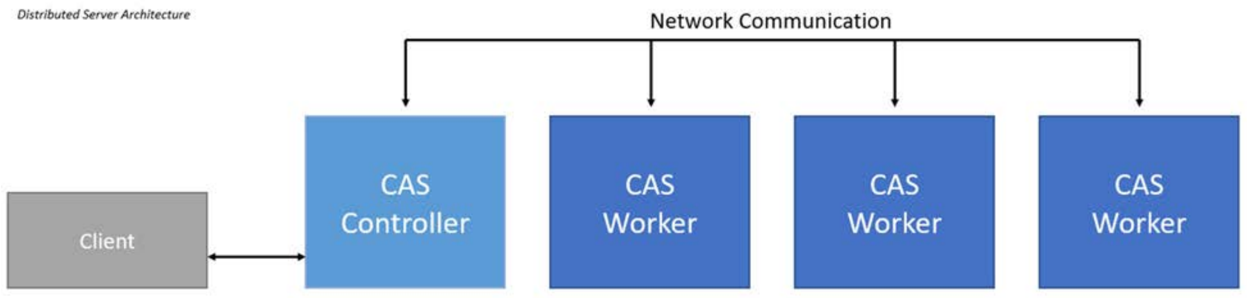

Figure 1. Distributed Server Architecture

Here are some benefits of CAS:

- You can set up CAS as a distributed server that consists of multiple machines. When you can distribute the workload across multiple machines, you have rapid processing for all your large-scale analytics needs.
- You do not need to be concerned if one or more of the nodes becomes unresponsive, because the distributed servers are fault tolerant. If the controller node cannot communicate with a worker node, another worker node takes over by analyzing a copy of the data.
- The distributed server configuration scales horizontally, which means that you can add more worker nodes to distribute the processing and improve processing time.
- If you are more familiar with a programming language other than SAS, you can use the standardized code base in CAS to use not just SAS, but also Python, Java, Lua, and even R.

One of the first steps you need to take in order to start using CAS is to load your data and formats into CAS. As you are performing these tasks, you should consider the following things:

- The encoding of your data likely will change from either LATIN1 or WLATIN1 to UTF-8. This paper discusses the possible data truncation that occurs when encoding is changed and how to prevent that truncation.
- You can use two different modes, parallel and serial, to load data into CAS. You should understand these modes as well as the benefit of loading data into CAS in parallel mode versus in serial mode.
- The size of your CAS table likely will be larger than the source SAS data set. The reason for this increase is explained here, and a demonstration can show you how you can control the table size.

All of the above information is combined in this paper into two examples. The first example shows loading a SAS data set into CAS. The second example shows how to use the CNTLIN= and CNTLOUT= options to move a format catalog into CAS. The example also provides a simple statement that will save your formats and ensure that you can access them in subsequent CAS sessions.

ENCODING IN CAS

WHAT IS ENCODING?

It is important to understand what a character encoding is before you learn how to change session encoding from WLATIN1 or LATIN1 to UTF-8. Here are a few terms and concepts that will help to define encoding:

- A character set is the set of characters and symbols that are used by a language or a group of languages. The character set contains the following:
    - the unaccented Latin characters A–Z
    - the digits 0–9
    - characters such as punctuation (special characters)
    - characters that are specific to a nation or group of nations (national characters)
    - the control characters that are used by the computer

- An encoding method assigns the numeric representations to a set of characters. This method includes determining the ranges in the code page where each character appears, as well as the number of bits used to store the numeric representation of the character.
- A code page is a table of all the code points that are the unique representations of each character in a character set.
- Finally, a character encoding is the result of applying an encoding method to a character set. The encoding maps each character in a character set to the unique numeric representation within the code page. An encoding basically defines the way those numbers are stored in the computer.

Each SAS session has an assigned encoding. You can see which encoding your SAS session is using by submitting the following OPTIONS procedure:

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

This code writes the following output to the log:

```
2097  proc options option=encoding;
2098  run;

    SAS (r) Proprietary Software Release 9.4  TS1M6

    ENCODING=WLATIN1  Specifies the default character-set encoding for
    the SAS session.
```

You can use the CONTENTS procedure to determine the encoding method of the SAS data set that you are moving over to CAS:

```
proc contents data=mylib.class;
run;
```

The PROC CONTENTS output shows the encoding of the MYLIB.CLASS data set (which is just a copy of SASHELP.CLASS). As you can see, the encoding is confirmed as “wlatin1 Western (Windows)”:

```
                      The CONTENTS Procedure

Data Set Name         MYLIB.CLASS         Observations          19
Member Type           DATA                Variables             5
Engine                V9                  Indexes               0
```

```
Created                06/25/2015           Observation Length    40
                       00:07:09
Last Modified          06/25/2015           Deleted Observations  0
                       00:07:09
Protection                                  Compressed            NO
Data Set Type                               Sorted                NO
Label                  Student Data
Data Representation    WINDOWS_64
Encoding               wlatin1  Western
                       (Windows)
```

Western Language Encoding

Most users in the United States with SAS®9 installations use a single-byte Western language encoding as the default. A single-byte encoding means that each supported character is stored as a single byte of memory. The Western language encodings support Western languages like English and French, but do not support any of the Asian languages. The default encoding in a UNIX environment is LATIN1 and WLATIN1 on Windows. These two encodings are basically identical. The WLATIN1 encoding also contains some special punctuation characters that are not found in LATIN1.

UTF-8 Encoding

UTF-8 is becoming the most used and preferred encoding because it is capable of representing all of the characters used in modern software. It is important that CAS be able to analyze any and all data, so UTF-8 is the only encoding that is available within CAS.

UTF-8 is a multibyte encoding that can represent all of the characters available within the Unicode character set. A UTF-8 character can be represented by up to 4 bytes. Table 1 shows the number of bytes that are required to display different characters.

Number of Bytes	**Characters**
1	U.S. English or ASCII characters
2	Eastern and Western European, Baltic, Greek, Turkish, Cyrillic, Hebrew, and Arabic characters
3	Chinese, Japanese, Korean, Thai, Indic, and certain control characters
4	Uncommon Chinese, Japanese, and Korean characters, and various historic scripts

Table 1. Width of the Characters

Each of the 128 characters that make up the ASCII character set is represented as 1 byte in UTF-8. All other characters in UTF-8 each require 2–4 bytes in memory. (See Table 1.) For example, the registered trademark symbol, ®, can be represented by 1 byte in the single-byte encoding WLATIN1, but it must be represented by 2 bytes in UTF-8. For example, the following list demonstrates these representations:

- WLATIN1 hexadecimal representation for ®—AE
- UTF-8 hexadecimal representation for ®—C2AE

EXAMPLE OF TRANSCODING DATA TO UTF-8 ENCODING

Transcoding is the process of converting data from one encoding to another. When a WLATIN1 data set is loaded into CAS, the data must be transcoded into the UTF-8 encoding.

When data is transcoded to UTF-8, it is possible that some characters require additional bytes in order to be represented in UTF-8. If the variable is not wide enough to store the transcoded value, the data might be truncated. You then see an error when you try to use the data.

As mentioned in the last section, the registered trademark symbol requires additional bytes to store the character in UTF-8. Without making any adjustments to your code, moving a data set that contains this symbol into CAS and then referencing it results in an error. Here is a simple example. In this example, a data set is created on a PC and then loaded into CAS using PROC CASUTIL:

```
/* This code is submitted on the PC using WLATIN1 encoding */
libname mylib 'u:\';
data mylib.registered;
   x='I work at SAS ®';
   y=999;
run;

/* This code is then executed in the SAS Viya environment */
proc casutil;
load casdata='registered.sas7bdat' incaslib='mycas' outcaslib='casuser'
     promote
     casout='registered9';
quit;
```

Here is the resulting log output, with a transcoding error:

```
86   proc casutil;
NOTE: The UUID '90da91b7-12da-4045-bdfe-648e43b712a2' is connected
using session CASAUTO.
87   load casdata='registered.sas7bdat' incaslib='mycas'
outcaslib='casuser' promote
88   casout='registered9';
ERROR: BASE Data Connector, Cross Environment Data Access (CEDA)
translation failed. Some character data was lost during transcoding
while loading the table. Either the data contains characters that are
not representable in the new encoding or truncation occurred during
transcoding. The charMultiplier= import option may prevent the error.
ERROR: BASE Data Connector failed.
ERROR: The action stopped due to errors.
NOTE: The Cloud Analytic Services server processed the request in
0.921302 seconds.
89   quit;
NOTE: The SAS System stopped processing this step because of errors.
NOTE: PROCEDURE CASUTIL used (Total process time):
      real time           1.03 seconds
      cpu time            0.03 seconds
```

Again, the reason why the error message occurs is because the variable that contains the registered trademark symbol does not have enough bytes of storage to represent that character in UTF-8.

Note that this error message not only provides the exact cause of the error, but also recommends a method of correcting the error:

```
The charMultiplier= import option may prevent the error.
```

The CHARMULTIPLIER= Option

The CHARMULTIPLIER= option is available within the IMPORTOPTIONS= argument in PROC CASUTIL. The option specifies the number of bytes per character to use for storing the character data.

Here is the PROC CASUTIL code from the previous section. It has been updated to use the CHARMULTIPLER= option:

```
proc casutil;
load casdata='registered.sas7bdat' incaslib='mycas' outcaslib='casuser'
     promote
     casout='registered9'
     importoptions=(filetype='basesas',charmultiplier=2);
quit;
```

This option exists in multiple locations. In addition to PROC CASUTIL, this option can also be used with the CAS procedure. This option is also available as the NCHARMULTIPLIER= option in the LIBNAME statement using the CAS engine. Finally, this option is also available when you import a SAS data set into CAS by using the IMPORT procedure in SAS® Environment Manager.

When you use the CHARMULTIPLIER= option, it is important to know the characteristics of your data. Not all data sets require the CHARMULTIPLIER= option when loading the data into CAS. If you are sure that your data does not contain any extended ASCII characters, then you do not need to use this option. Using this option when it is not needed expands your data unnecessarily and takes up more space than is needed.

PARALLEL LOADING OF SAS DATA SETS

One of the primary benefits of CAS is its ability to provide high-speed analysis of big data. However, before you can take advantage of this ability, you must load the data into the memory space located within each of the worker nodes. CAS is capable of performing high-speed analysis because the analysis is spread across multiple worker nodes that are running in parallel. Parallel processing is also available when you load data into CAS. With a parallel load of the data, the data is transferred across multiple channels, which allows for a much faster load time. CAS supports parallel data transfer for multiple file types, but this paper discusses only the loading of SAS data sets (SAS7BDAT files).

SERIAL DATA TRANSFER

Before you learn about parallel data transfer, first consider serial data transfer, which is also available within CAS. Serial data transfer involves a sequential transfer of the data through a single channel from the source of the data to the destination. Serial data transfer is always available and is the most flexible method of data transfer. If your site has an SMP mode (CAS deployed on a single machine), serial data transfer is the only method available.

The more common deployment of CAS is an MPP mode, where the CAS deployment consists of a CAS controller and multiple workers. Even with multiple workers, a serial load of the data still might be the only option. The most common circumstance that would require a serial load of the data is when the data source is accessible only by the CAS controller. With this setup, the data is transferred via a serial data transfer to the controller, which then evenly distributes the data to the CAS workers. (See Figure 2.)

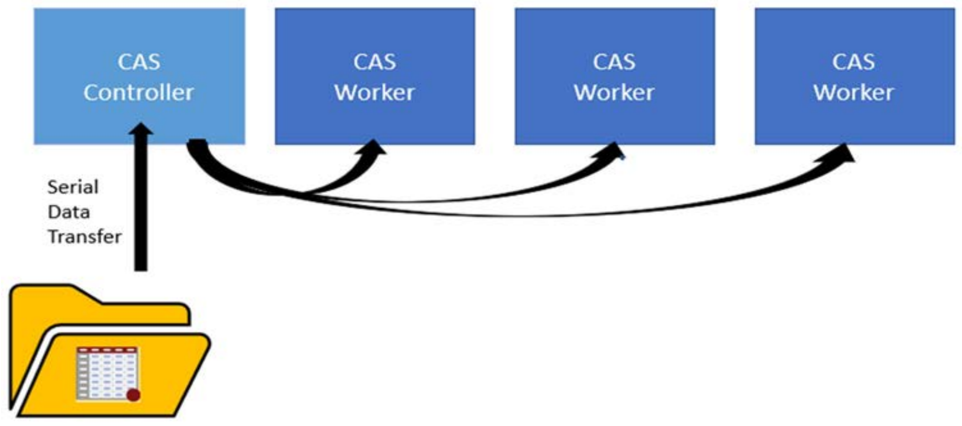

Figure 2. Serial Data Loading with MPP Mode in CAS

PARALLEL DATA TRANSFER

If a parallel data transfer is available, then it is the preferred method of loading data into CAS. A parallel data transfer allows the data to be concurrently transferred across multiple, independent connections between the data source and the destination. This method greatly increases the speed in which your data sets can be loaded into CAS.

There are two primary requirements for a parallel data transfer in CAS. The first is that your CAS deployment be an MPP deployment consisting of multiple CAS workers. The second is that each of the CAS workers has direct network access to the location of the data set. This access is provided by a shared file system that is mounted on each CAS worker. (See Figure 3.) If these requirements are met and a method that allows for parallel data transfer is used, your SAS data sets can be loaded into CAS very quickly.

When parallel data transfer is used, the CAS controller instructs each CAS worker to directly and simultaneously load the data set from the file location.

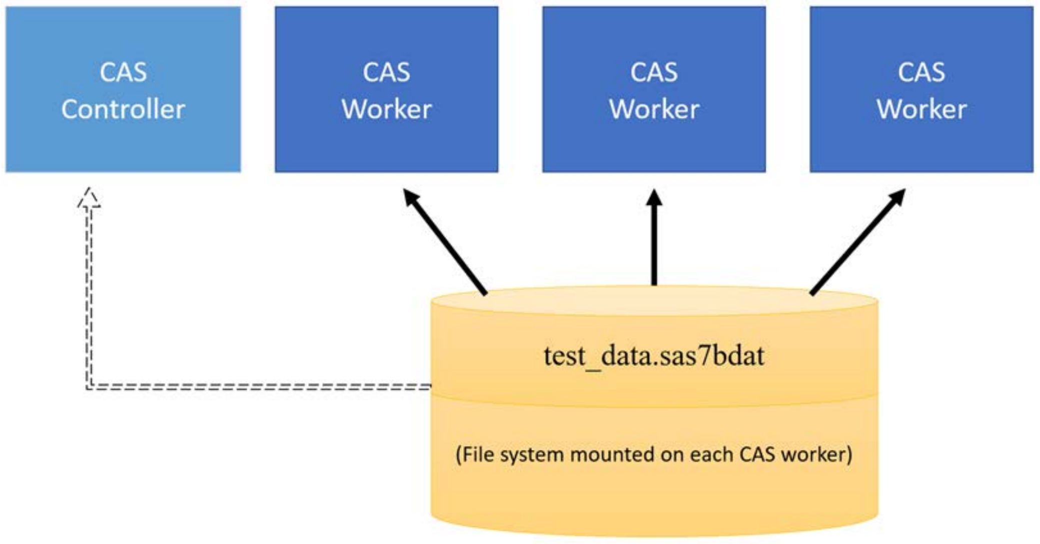

Figure 3. Parallel Data Loading with MPP Mode in CAS

EXAMPLE OF A PARALLEL DATA TRANSFER

This example uses PROC CASUTIL to execute a parallel data transfer of a SAS data set into CAS. In this code, the location of the SAS data set is provided in the CASLIB statement. The data set is then loaded into CAS using PROC CASUTIL:

```
caslib mycaslib path='/x/xxxxx.sas.com/vol/volxx/uxx/xxxxxx'
       datasource=(srctype='path’);
proc casutil;
load casdata='testdata.sas7bdat' incaslib='mycaslib'
     outcaslib='mycaslib' casout='paratest2' promote
     importoptions={filetype='basesas',dtm='parallel',debug='dmsglvli'};
quit;
```

The CASLIB statement adds the MYCASLIB caslib to the available caslibs in the current CAS session. The PATH= argument points to the shared file system mounted on each CAS worker. The SAS data set is stored in this location. The SRCTYPE= argument lists the data source type, which is PATH here, because a path-based caslib was created in the CASLIB statement.

The LOAD statement instructs PROC CASUTIL to load the TESTDATA data set into CAS and then name the resulting table as PARATEST2.

The IMPORTOPTIONS= options in this example are not technically necessary for this code to run. They are included here to show some of the options that are used when performing a parallel load of a data set into CAS. The FILETYPE= option is set to BASESAS to indicate that the file being loaded is a SAS data set. The DTM= option, for the data transfer method, is set to AUTO. That value instructs PROC CASUTIL to load the data in parallel if possible, or else load in serial. If you want to load the data only in parallel, you can set this DTM= option to PARALLEL and the data is then loaded in parallel if possible and the procedure generates an error if it is not able to do so. Finally, setting the DEBUG= option to DMSGLVLI writes a note to the log indicating whether the data was transferred in parallel or serial.

Here is the resulting log output after running this code:

```
82   proc casutil;
NOTE: The UUID '9bf4e0b9-a833-b14d-8ac8-12bd397ca9b8' is connected
using session CASAUTO.
83   load casdata='testdata.sas7bdat' incaslib='mycaslib'
outcaslib='mycaslib' casout='paratest2' promote
84
importoptions={filetype='basesas',dtm='parallel',debug='dmsglvli'};
NOTE: BASE Data Connector, Using PARALLEL mode
NOTE: Cloud Analytic Services made the file testdata.sas7bdat available
as table PARATEST2 in caslib mycaslib.
NOTE: The Cloud Analytic Services server processed the request in
190.049646 seconds.
85   quit;
NOTE: PROCEDURE CASUTIL used (Total process time):
      real time           3:10.06
      cpu time            0.52 seconds
```

You can see that the data set loaded successfully and the PARATEST2 table was created. Two key pieces of information are highlighted. First, the note indicates that the data set was loaded in parallel. Second, it took 3 minutes and 10 seconds to load this table.

Now look at the log after loading the exact same table in serial transfer mode:

```
82   proc casutil;
NOTE: The UUID '9bf4e0b9-a833-b14d-8ac8-12bd397ca9b8' is connected
using session CASAUTO.
83   load casdata='testdata.sas7bdat' incaslib='mycaslib'
outcaslib='mycaslib' casout='paratest1' promote
84   importoptions={filetype='basesas',dtm='serial',debug='dmsglvli'};
NOTE: BASE Data Connector, Using SERIAL mode
NOTE: BASE Data Connector, comm buffer size  : 524288
NOTE: BASE Data Connector, obs per buffer    : 173
NOTE: BASE Data Connector, reclen in buffer  : 3024
NOTE: BASE Data Connector, reclen on data set: 3024
NOTE: Cloud Analytic Services made the file testdata.sas7bdat available
as table PARATEST1 in caslib mycaslib.
NOTE: The Cloud Analytic Services server processed the request in
480.520151 seconds.
85   quit;
NOTE: PROCEDURE CASUTIL used (Total process time):
      real time           8:00.54
      cpu time            1.22 seconds
```

The first highlighted note in the log output confirms that the data was loaded using the serial transfer mode. The second highlighted note shows that it took 8 minutes to load the data. As you can see, loading data in parallel is a *lot* faster—almost 2.5 times faster than the serial load.

If possible, always try to load your data in parallel. This recommendation is true for all data sources, not just SAS data sets. When you are working with big data, you want to be efficient and load your data as fast as possible.

THE SIZE OF THE CAS TABLE VERSUS THE SIZE OF THE SOURCE SAS DATA SET

Something that you will notice when a data set is loaded into CAS is that the size of the CAS table is likely larger than the original data set. Depending on your data, the CAS table can sometimes be much larger. Although CAS is designed to run large-scale analytics on big data, you still have a finite amount of memory available on each worker. So, to maximize efficiency, you want to reduce the size of your CAS table and use the least amount of memory as possible. This section discusses why your CAS table can be larger than the original SAS data set and also how you can reduce the size of the table.

The reason your CAS table can be larger than your SAS data set is because of data structure alignment. Data structure alignment is the way data is arranged and accessed in computer memory. Data alignment means putting the data in memory at an address equal to some multiple of the word size. For example, numeric data is 8-byte-aligned in CAS, which means that numeric data is stored in a memory address that is a multiple of 8.

The purpose of data alignment is that aligned access is faster because the external bus to memory is not a single byte wide, but typically 4 or 8 bytes wide (or even wider). Therefore, the CPU does not fetch a single byte at a time, but rather 4 or 8 bytes starting at the requested address. For example, on a 64-bit machine, based on the number of banks, a DOUBLE variable (a numeric value in CAS) is allocated on an 8-byte boundary and requires only one memory read cycle.

Finally, in order to achieve data structure alignment, there also needs to be data structure padding. To properly align the data, it is necessary to insert some extra bytes between the end of the last data structure and the start of the next data structure as data is placed into memory. The padding is necessary to fill out the 4- or 8-byte boundary.

Within CAS there are three variable types: DOUBLE, CHAR, and VARCHAR. Table 2 describes each variable.

Variable Type	**Description**	**Range**
CHAR(*n*)	This variable stores a fixed-length character string. *n* is the maximum number of bytes to store. The maximum number of bytes is required to store each value, regardless of the actual size of the value.	1–32,767 bytes
VARCHAR(*n*)	This variable stores a varying-length character string. *n* is the maximum number of characters to store.	1–536,870,911 characters (in UTF-8 encoding)
DOUBLE	This variable stores a numeric value, including dates and times, as a floating-point number.	8 bytes (for the CAS engine)

Table 2. Variable Types That Are Available in CAS

All numeric data in CAS has a DOUBLE variable type, which is a numeric value that is stored as an 8-byte floating-point number. A numeric value that has fewer than 8 bytes must be padded to 8 bytes. In contrast, a numeric variable in Base SAS® can have a minimum length of 3 bytes in both PC and UNIX environments. This difference in the minimum size of a numeric value is the reason that a CAS table that contains numeric data is larger than the SAS data set. The DOUBLE variable type is the only numeric variable type currently available in CAS, so there is no way to reduce the size of a numeric variable.

Character data in CAS can be one of two types: CHAR or VARCHAR. The CHAR variable type is the same as it is in Base SAS. The value is stored as a fixed-length character string. The length of the variable is the maximum number of bytes needed to store the longest possible value for that variable. The maximum number of bytes is used to store each value, regardless of the actual number of bytes in that value.

VARCHAR is a new variable type for CAS. The VARCHAR variable type is a varying-length character value, whose length is the maximum number of characters needed to store the longest possible value for that variable. The VARCHAR variable type consists of two main characteristics:

- Its length is determined by the number of characters rather than bytes.
- Its length varies from row to row, depending on the characters in that string.

For example, if a variable is defined as VARCHAR(5), the variable can have a length up to 5 characters, but the actual storage used depends on the number of characters found in each individual value. In the following list, each character uses 1 byte of storage:

- abcde (5 bytes of storage used)
- fg (2 bytes of storage used)
- hijk (4 bytes of storage used)

The VARCHAR data type can store a varying-length value, and this capability can reduce the size of the CAS table that is created from a SAS data set. If your data set contains character data whose values vary greatly in length, then the VARCHAR variable type can prevent the data from being padded, thus reducing the size of the CAS table being created.

It is advantageous to use the VARCHAR variable type for most character variables. However, if a variable has a shorter length, like an ID variable, then using the fixed-length CHAR variable type is more efficient. However, the VARCHAR variable type has a 16-byte overhead, plus the memory needed to store the value. Because of that, it is recommended that only variables with values that consistently exceed 16 characters and that have values that vary greatly in length be defined as VARCHAR.

Here is a data set example with three numeric variables, 30 character variables, and 5.5 million observations. Fifteen variables have values with a length up to 14. The other 15 variables have values up to 100 characters in length. The following PROC CASUTIL code loads the data set into CAS. Within the IMPORTOPTIONS= option is the VARCHARCONVERSION= option. This option specifies the column length that determines when to convert from the CHAR variable type to the VARCHAR variable type. Because of the 16-byte overhead associated with using VARCHAR, you can set this option to 16:

```
proc casutil;
load casdata='testdata.sas7bdat' incaslib='mycaslib'
     outcaslib='mycaslib' casout='varchar4' promote
     importoptions={filetype='basesas', varcharconversion=16};
quit;
```

The following list shows how the CAS table is reduced in size after you use the VARCHAR variable type:

- Original data set size: 15.6 GB
- CAS table size, when using only CHAR variables: 16.1 GB
- CAS table size, when using VARCHAR variables: 7.2 GB

As you can see, the VARCHAR variable type can greatly reduce the size of a CAS table that is created from a SAS data set. You should always use the VARCHARCONVERSION=16 option to control which variables are converted to VARCHAR, so that you can maximize the benefit of this variable type.

PUTTING IT ALL TOGETHER

The last section of this paper combines all the information from the previous sections. In this example, a SAS 9.4 data set as well as a format catalog are loaded into CAS. The data set contains both numeric and character variables. There are character variables within the data set that can vary greatly in length from observation to observation, as well as character variables that contain extended ASCII characters. The format catalog also contains format ranges where extended ASCII characters are present.

The SAS 9.4 data set and format catalog are in a local directory. Because you cannot access this directory from CAS, you can copy the data set to a shared drive that is accessible both from your PC and directly from CAS. There is an extra step involved to move the format catalog, which will be discussed after the data set is loaded in.

When your data is in a location that can be accessed by CAS, open SAS Environment Manager and SAS® Studio to start to load the data and format catalog.

The first step you will take is to establish a CAS session using the CAS statement. The syntax for this statement cannot be any simpler:

```
cas;
```

There are a lot of options available for the CAS statement, but all you want to do here is to establish a CAS session. This statement creates a CAS session with the default name CASAUTO.

The next step is to create a path-based caslib that points to the directory where your data set and format are stored. Use a CASLIB statement as shown here:

```
caslib mycas path='/x/xxxxx.xxx.sas.com/vol/vol710/u71/xxxxxx'
       datasource=(srctype='path');
```

After you create the caslib successfully, load the data set into CAS by using PROC CASUTIL:

```
proc casutil;
load casdata='testdata.sas7bdat' incaslib='mycas' outcaslib='casuser'
     casout='alltogether' promote
     importoptions={filetype='basesas',varcharconversion=16,
     charmultiplier=2,dtm='auto',debug='dmsglvli'};
quit;
```

This paper already discussed the IMPORTOPTIONS= options, and they are used again in the above syntax:

- The VARCHARCONVERSION= option instructs PROC CASUTIL to convert all character variables to VARCHAR when they exceed 16 characters in length.
- The CHARMULTIPLIER= option instructs PROC CASUTIL to increase the width of fixed-byte-width character variables. This option converts extended ASCII characters to UTF-8. In this example, the character variables are expanded to 2 bytes.
- The DTM= option specifies the type of data transfer used when loading the data set into CAS. The value of AUTO instructs PROC CASUTIL to load the data in parallel if possible and resort to serial only when necessary.
- The DEBUG= option is set to DMSGLVLI. Although this option is not necessary for this code to run, it enables you to confirm whether the data set was loaded in parallel or not.

Now that the data set is loaded into CAS, it is time to load the format catalog. A SAS format catalog on the PC cannot be directly loaded into CAS when CAS is running on Linux. However, the process is very similar to loading a data set, but with a couple of additional steps. An easy way to move a format catalog to CAS is to write the format catalog out to a data set, load that data set into CAS, and then re-create the format in a CAS format library.

The first step is to write out the format catalog to a data set. Use the CNTLOUT= option in the PROC FORMAT statement. Note that this code is executed in SAS®9:

```
/* MYLIB is the current location of the format library */
libname mylib 'c:\dstep';

/* NEWLIB is the location of the shared drive that is accessible from
CAS */
libname newlib 'u:\';

/* The CNTLOUT= option points to the data set being created that will
contain all the data needed to re-create the format in CAS */
proc format lib=mylib cntlout=mylib.outfmt;
run;
```

Now, you can use the exact same PROC CASUTIL code from an earlier section to read in the OUTFMT data set into CAS:

```
proc casutil;
load casdata='outfmt.sas7bdat' incaslib='mycas' outcaslib='mycas'
     casout='outfmt'
```

(code continued)

```
      importoptions={filetype='basesas',varcharconversion=16,
      charmultiplier=2,dtm='auto',debug='dmsglvli'};
quit;
```

The only step that remains is to re-create the format:

```
/* This statement creates the CAS libref for all of the current
caslibs. It enables you to use MYCAS.OUTFMT in the next step */
caslib _all_ assign;

proc format cntlin=mycas.outfmt casfmtlib='mycasfmt';
run;
```

The CNTLIN= option in the PROC FORMAT statement points to the OUTFMT data set, which contains the data that is needed to re-create the format. The CASFMTLIB= option adds the format to the CAS session in a format library named MYCASFMT.

Here is the resulting log output:

```
93   proc format cntlin=mycas.outfmt casfmtlib='mycasfmt';
NOTE: Both CAS based formats and catalog-based formats will be written.
The CAS based formats will be written to the session CASAUTO.
NOTE: Format library MYCASFMT added. Format search update using
parameter APPEND completed.
NOTE: Format $CASFMT has been output.
94   run;
NOTE: PROCEDURE FORMAT used (Total process time):
      real time           0.39 seconds
      cpu time            0.04 seconds
```

There are two notes of interest in this log output. The first note lets you know that the formats were written to both your CAS session and your SAS session. When you are programming in SAS Studio running on SAS Viya, you have two programming environments. You can run code in CAS as well as on the SAS server. Therefore, if you have code (for example, a DATA step) that has syntax in it that prevents the code from running in CAS, the code runs in SAS instead. If the format did not exist in SAS, the code would generate an error and not execute.

The second note lets you know that the MYCASFMT format library was added to the CAS session. It also lets you know that the MYCASFMT format library was added to the FMTSEARCH= option. Therefore, the format is always found in your CAS session without having to manually update the FMTSEARCH= option.

You can use the following CAS statement to confirm that the format library was added to your CAS session:

```
cas casauto listfmtsearch;
```

The log output from this statement shows that all the format libraries are currently available in the CAS session:

```
97   cas casauto listfmtsearch;
NOTE: FmtLibName = SASSUPPLIEDFORMATS
         Scope = Both
NOTE: FmtLibName = MYCASFMT
         Scope = Session
```

The last step that you want to take is to save the format to a caslib. The following code enables you to persist your formats across CAS sessions. This code saves the MYCASFMT format library to the CASUSER caslib in a SASHDAT file that is named CASFMT:

```
cas casauto savefmtlib fmtlibname=mycasfmt caslib=casuser
    table=casfmt replace;
```

CONCLUSION

When you are moving your SAS data sets and formats from SAS 9.4 to CAS, there are a number of considerations to think about before, during, and after the move. This paper discussed different items that you need to consider, including the encoding for both the data set and format catalog, whether to load your data in parallel, and how to manage the difference between the size of a CAS table versus the size of the source data set. In addition to explaining these considerations, this paper provided steps you can take to accommodate the changes from SAS 9.4 to CAS. Finally, the last section showed you how to move your data set and format catalog into CAS.

After you successfully move your data and formats into CAS, you can start to take advantage of the high-performance analytic capabilities that are available in SAS Viya and CAS!

CONTACT INFORMATION

Your comments and questions are valued and encouraged. Contact the author at:

Kevin Russell
SAS Institute Inc.
SAS Campus Drive
Cary, NC 27513
Email: **support@sas.com**
Web: **support.sas.com/en/support-home.html**