Getting Your SAS® 9 Code to Run Multi-Threaded in SAS® Viya® 3.3

Phil Weiss
January 2018
Version 1.0
# Table of Contents

Introduction........................................................................................................................................... 2
Leveraging the CAS Server .......................................................................................................................... 2
Understanding SAS® 9 & SAS® Viya® architecture differences ................................................................. 3
  Single versus multi-threaded distinctions .................................................................................................. 3
  Persisted in-memory data ......................................................................................................................... 4
  The decision to multi-thread or single-thread ........................................................................................... 5
Syntax changes due to differences in architecture ....................................................................................... 6
  DATA step specific changes ....................................................................................................................... 6
    DATA statement data set options ............................................................................................................ 6
    INFILE, INPUT and DATALINES statements ......................................................................................... 7
    Some older non-supported functions ..................................................................................................... 7
    No ‘Descending’ Option on a BY statement (within DATA step) ............................................................ 8
  Macro (code generator) processing changes ............................................................................................... 8
    CALL EXECUTE, CALL SYMPUT and the SYMGET Function ................................................................. 8
  Procedure differences ............................................................................................................................... 8
    PROC SORTs are not needed and can be removed ............................................................................... 8
    Miscellaneous PROC similarities and differences ............................................................................... 8
  User defined formats ............................................................................................................................... 10
Miscellaneous potential syntax changes .................................................................................................... 10
  Unsupported SAS 9 formats .................................................................................................................... 10
  Statement and Functions supporting inter-row dependencies ................................................................ 10
  Temporary arrays ................................................................................................................................... 11
  Session encoding .................................................................................................................................... 11
  Unnecessary or obsolete statements (MODIFY, REMOVE, REPLACE) .................................................. 12
  WHERE clause for output tables .............................................................................................................. 12
Processing validation ................................................................................................................................. 12
  DATA step data processing differences .................................................................................................. 12
  Procedure-related processing changes .................................................................................................... 13
How to proceed ........................................................................................................................................... 14
Introduction

This document represents a compendium of the main factors affecting SAS® Viya® multi-processing as viewed from a SAS® 9 perspective. More importantly, it brings together information from various sources of SAS Viya technical documentation to provide a guidebook designed to assist with code transition efforts.

While this collection of topics represents most of the significant processing issues that one might encounter, it is not a complete list of all potential concerns or error-prone conditions, nor is it a manual on how to write good or efficient SAS Viya code. Additionally, no attempt is made to fully explore all possible replacements for some functionalities found in SAS 9, many of which are unique. As an initial reference document, it begins with a discussion of the fundamental architectural differences inherent in single-threaded as opposed to multi-threaded processing. That is because actual code translation must start with a firm architectural understanding before actual testing and syntax-checking work is initiated.

Leveraging the CAS Server

If you are a “dyed-in-the-wool” SAS programmer, you might be asking yourself, “what will it take to get my existing SAS 9 code to run multi-threaded in a SAS Viya environment?” Depending on how complicated or unconventional your existing SAS code is, it’s possible that it will run multi-threaded without any changes. But if it does not run faster and/or give you the same answers, then you need to know how to tune it to get the best possible performance using the pooled memory SAS Viya runtime engine called CAS (which stands for Cloud Analytic Services and is pronounced ‘caz’). This article attempts to provide you a high-level description of the most common things that might be affected and might need to change. It is not meant to be an exhaustive list. But with increased awareness of possible issues, you should be able to identify portions of your existing SAS 9 code that might need to change and experience fewer problems when testing it on a SAS Viya-enabled memory-cluster.

Leveraging the CAS server that is part of the new SAS Viya 3.3 release includes a whole host of tangible benefits. The main reason is represented by a simple three-word phrase: tremendous performance gains. Because processes run so much faster, you can complete your work faster, meaning you can complete more work, and even entire projects, in a significantly reduced time frame. As a result of the substantial SAS Viya redesign and consolidation, the interfaces are now more consistent and easy to use. SAS has also increased its integration between different tasks, and enabled a wide range of new capabilities. And finally, most future new development will be designed to run against CAS. These are just a few of the many reasons to use it.

The bottom line for running multi-threaded is to keep all data and code executing in CAS. If CAS encounters a code block or snippet that is not CAS-enabled, then it drops the data to disk and passes runtime execution over to a single-threaded compute engine automatically. This can dramatically slow processing down, so it behooves you to know which code segments do not qualify for multi-threaded execution and to make conscious decisions about how and where the code runs.

For those who do want to attempt to retro-fit their single-threaded code so that it runs in a multi-threaded environment, there are two basic things to consider: syntax changes and calculation modifications (due to processing differences). Regarding the latter, processing differences between single-threaded and multi-threaded architectures may lead to non-matching output and results. Care needs to be given to ensure that single-threaded processing results match multi-threaded ones – but more on that later.
Understanding SAS 9 & SAS Viya architecture differences

Single versus multi-threaded distinctions

It is important to step back for a moment and examine how both syntax and processing impacts are derived from architectural differences affecting computational processing. To begin, single threading is defined as the processing of one command or instruction at a time. In this processing paradigm, which typically runs on no more than a single CPU (and a single core), code is submitted for processing in an ordered sequence called a ‘thread.’ These instructions are executed as they are encountered in a program against a single set of data, usually one record or one row at a time. In other words, a thread represents a sequentially ordered queue of processing tasks. Multi-threading works similarly, but leverages multiple threads to spread the work out and complete work tasks concurrently. A limited number of procedures in SAS® were multithreaded as far back as Version 6 in the early 1990’s, and these could already take advantage of additional cores on the same machine, in a system that is referred to as Symmetric Multi-Processing (SMP).

Yet code blocks represented by SAS 9 PROCs and DATA steps still run sequentially, even though the work within them may be multi-threaded. Sequential processing means that subsequent blocks of code in an instruction stack cannot execute before earlier blocks have finished. In other words, if you have a series of PROCs and DATA steps in a single program, each individual step may be multi-threaded and run concurrent work tasks, but all the PROCs and DATA steps in a SAS program cannot run at the same time (simultaneously); they have to run in a sequential order. The exception to this is PROC TSMODEL in SAS® Visual Forecasting and some limited components in Model Studio, where object-oriented code enables special concurrent project or pipeline processing.

In contrast to single-threading, multi-threading tends to distribute the same instructions to other available threads for execution creating many different queues on many different cores using separate allocations or subsets of data. Most of the time, multiple threads perform operations on isolated collections of data that are independent of one another, but part of a larger table. For that reason, it is possible to have a counter (e.g. n+1;) operating on one thread to produce a result that may be different than a counter operating on another thread, since each thread is working on a different subset of the data. This is why results can be different from thread to thread unless and until the individual results from multiple threads are summed together. It’s not really as complicated as
it might sound. That is because SAS Viya automatically takes care of most collation and reassembly of processing results, with a few minor exceptions where the programmer must further specify how to combine results from multiple threads (see the 'Inter-row DATA step dependencies' section below).

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed, parallel processing?</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>In-memory data persistence?</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Common performance speed-up</td>
<td>1x</td>
<td>1x – 10x</td>
<td>10x – 20x</td>
<td>Up to 100x*</td>
</tr>
</tbody>
</table>

Table 1. Why move to multi-threading? Because of the tremendous performance gains!
*Increase depends on a number of factors including hardware allocation; performance could actually be higher.

While it is generally true that DATA step processing duplicates its instructions across all available CAS Server nodes, it is not necessarily the case with SAS Viya procedures. A lot of SAS Viya procs have been specially designed to allocate some specific analytic tasks to different processors so they can compute concurrently with other processing instructions. To reiterate, not all procedure related threads in a SAS Viya multi-threaded environment need to work on the same task because they can communicate with other running processes and are able to share data.

When multiple machines or nodes are involved, this thread communication ability is enabled through implementation of a SAS patented message protocol that allows different threads to “talk” to each other and provide data to other nodes across an interconnected cluster or network of CPUs. Called inter-node communication, the ability for different threads to communicate with one another mostly helps iterative analytics and concurrent processing when enabled. All SAS Viya procedures automatically collate and summarize data correctly when using multiple threads, meaning DATA step is the only place where output needs to be controlled and validated.

Even though both multi-threading and single-threading largely operate sequentially, the underlying software must nevertheless be intentionally configured to support parallel processes running on multiple cores or threads. This means that to support multi-threading, older single-threaded language constructs (i.e. statements, functions, informats/formats, etc.) had to be completely rewritten from the ground up.

Read more about single versus multi-threading in SAS Viya.

Persisted in-memory data

Another unique SAS Viya feature to consider is the shift away from using flat files and external data sets to a strategy of using persisted, pre-loaded data tables. In SAS Viya, all data typically goes through an I/O conversion
process only once and can be reused as many times as needed thereafter, without incurring the same expense of conversion into a binary, machine-level format. SAS Viya data is either stored within the RAM of a single machine (and runs in SMP mode) or within a shared pool of allocated memory created from several networked machines as part of a common memory grid (which enables Massively Parallel Processing, or MPP mode). That pooled memory array is an integral part of CAS. Once the data is loaded into CAS, all processing instructions execute very quickly against the pre-converted, in-memory data. Because data is always pre-loaded, it makes some prior language statements and functions irrelevant. Examples of obsoleted code due to the shift to persisted data consist of things like functions that open and close data sets, and using INFILE and INPUT statements to read records from disk.

Other main feature differences between SAS Viya and SAS 9 include the following:

- CAS cannot work with the older four-level catalogs and catalog entries.
- There is no ability to connect to the SAS 9 metadata server. In SAS Viya, metadata and security is handled through microservices instead.
- Sas7bdat files can be loaded into CAS memory, but .sashdat (SAS High-performance data) files are more efficient and are the preferred file type in SAS Viya.
- Single-threaded processes use a Workspace server, while multi-threaded processes use a CAS server.
- CAS libraries are locations identified in memory as well as pointers to the data source files, while SAS 9 libraries simply act as pointers to disk storage (not memory storage).
- With few exceptions, tables that are loaded into memory cannot be altered, as there is no row level access that is enabled. This means that changes to tables usually require creating new copies of the data.

The decision to multi-thread or single-thread

Now, just because we have been speaking earlier about the benefits of multi-threaded processing, that does not mean that every portion of your existing code needs to be altered to run multi-threaded. In fact, there may be valid reasons why you might want to leave certain portions of existing code to run as single-threaded processes. Examples of when you might want to continue to run single-threaded include: 1) running specialized SAS 9 procedures not available in SAS Viya; 2) executing sections of code that cannot be easily modified to support multi-threading; or 3) leaving certain DATA steps and PROCs that are needed to support other downstream single-threaded processing (like PROC SORT).

What makes things somewhat difficult to understand is that some PROCs in CAS actually run as single-threaded processes, for example PROC PRINT and PROC CONTENTS which run against a different compute engine (not CAS). Since multi-threaded CAS actions underlie all SAS Viya PROCs, and even the DATA step, this means that some CAS actions can be designated to run as single-threaded processes, for example when specifying SINGLE=YES on the DATA statement.

As mentioned above, any code block that is indicated to run single-threaded means that it is directed to an alternate compute engine called the SAS Program Run-time Environment. You will also hear this referred to as the SAS Viya Workspace server sometimes, as it is analogous to the Workspace server in SAS 9. Like the SAS 9 Workspace server, the SAS Program Run-time Environment compute engine reads data from disk, loads (pages) blocks of data into memory, and processes code instructions using a single thread. The CAS server determines which blocks of code will run in CAS versus what data and code needs to be passed to the SAS Program Run-time Environment. Note that not all existing SAS 9 procedures have been designed to run on the SAS Program Run-time Environment as of SAS Viya version 3.3 (see this link for the current list of SAS Viya and SAS Program Run-time Environment supported procedures). To run specialized SAS 9 procedures that are not supported in SAS Viya, they
will need to be passed to a SAS 9 Workspace server which will have to have the appropriate data for processing located on disk.

Returning to the main point, not everything has to run in CAS pooled memory; SAS Viya multi-threaded programs can be interspersed with single-threaded steps where necessary. However, program transitions from single to multi-threaded processing may need to have output data generated by single-threaded processes duplicated by loading them into memory to support multi-threaded steps that may follow, and vice versa. Single-threaded processes need data on disk (not in SAS Viya memory) to be available for processing on the Workspace server. If you make the decision to run a DATA step in single-threaded mode in SAS Viya, you can use the option ‘single=yes’ on the DATA statement, like in this example:

```sas
DATA MYCAS.NEWAIR / single=yes;
```

**Note:** Running multi-threaded is the default in SAS Viya, so there is no need to indicate that parallel processing needs to use all available threads.

As a reminder, if the input or **source table** in SAS Viya is **not** located in-memory, and the output or **target table** destination is **not** in-memory, then SAS Viya will default to single-threaded processing on its own (see Table 2). This is why it is important, **if you want to enable multi-threading**, to keep everything in CAS memory that you can keep in memory, until it is no longer needed and can be written to disk. Management of the shared memory space is up to the programmer, meaning that when in-memory files are no longer needed, they should be removed from memory to free up memory resources.

<table>
<thead>
<tr>
<th>Source (input) table is not in CAS</th>
<th>Single-threaded, multiple machines (SAS Grid)</th>
<th>Target table is in CAS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Target table is not in CAS (or if <em>NULL</em> table is defined)</td>
<td>Single-threaded</td>
<td>Single-threaded</td>
</tr>
<tr>
<td>Source table is in CAS</td>
<td>Single-threaded</td>
<td>Single-threaded</td>
</tr>
</tbody>
</table>

*Table 2 – Source (input) and target (output) table both need to be in CAS in order to run multi-threaded!*

**Syntax changes due to differences in architecture**

**DATA step specific changes**

Over the last 40+ years, the SAS DATA step has grown increasingly flexible with lots of useful statements, functions, features and options that have continued to enhance its utility. However, because SAS Viya enables multi-processing and use of persisted in-memory data, some of these features may no longer be valid. What follows is a review of the coding aspects that may need to be altered to accommodate the new processing architecture, noting that a majority of SAS 9 functionality operates as it always has.
DATA statement data set options

The lead statement that begins a DATA step supports different arguments for data set options. Data set options are used to rename variables; select only the first or last \( n \) observations for processing; drop variables from processing or from the output data set; or specify a password for a data set. SAS Viya supports a smaller list of arguments for output tables which can be saved either in CAS memory or to disk as sashdat (SAS high-performance data) files. They include:

COMPRESS=, DROP=, KEEP=, LABEL=, RENAME=, REPLACE=, WHERE=

As stated earlier, several popular DATA statement options have been obsoleted, including INDEX=, FIRSTOBS=, OBS=, POINTOBS=, since there are no more pointers available to indicate specific rows due to data allocated to different cores and threads.

View the most current list of what is supported on the CAS server.

INFILE, INPUT and DATALINES statements

For those of you not familiar with SAS Viya, the DATA step in SAS Viya has two primary uses: loading data into SAS Viya memory, and processing in-memory data sets. Most SAS 9 DATA steps will need to be altered, if they were originally designed to read non-SAS data from disk or other databases. To run multi-threaded, you’ll need to modify your SAS 9 code to support the reading, and possibly the loading of table data into CAS. To be more specific, if your SAS 9 program reads data from an external file using the INPUT statement, then you need to alter the code to point to specific in-memory CAS tables holding the same data. That means that INFILE, INPUT, and DATALINES statements need to be removed and replaced with SET statements, with the new SET statements pointing to the correct in-memory tables.

Be aware that reading SAS data from disk using SET will defeat your multi-threaded intentions, as the DATA step will then run single-threaded. For multi-threaded processing to occur, you have to read from in-memory tables (see Table 2 above). Therefore, it may be necessary to add additional DATA steps or PROC CASUTIL steps to existing code first, in order to load the proper data into memory. (The easiest way to verify multi-threaded processing is to look at the messages in the log file.)

Some older non-supported functions

Due to changes in architecture between SAS 9 and SAS Viya, dozens of SAS 9 functions are now superfluous. This is largely because of differences between processing and memory usage. The list of CAS supported functions that will run is subject to change as decisions are made to add new functions or replace older ones depending on architectural compatibility, development planning, and other considerations. In general, though, the more than 150 obsoleted functions basically fall into five main categories:

1) Current active data set pointer or observation controls (GETVARC, POINT, PUTC, PUTN, etc.).
2) External file or data set controls (CLOSE, EXIST, FETCH, FILENAME, LIBREF, etc.).
3) Functions that access SAS 9 Program Data Vector (PDV) metadata information (VARFMT, VARNAME, VARNUM, VARTYPE).
4) Macro interface support (RESOLVE, SYMEXIST, SYMGET, etc.).
5) Miscellaneous (GAMMA, NORMAL, NOTE, SOUNDEX, RANUNI, ZIPCITY, ZIPSTATE, etc.).

The last group, the ‘Miscellaneous’ category, really represents functions that have been rewritten, and thereby improved, in SAS Viya. This list, for the most part, represents functions that have been replaced through consolidation or enhanced in a significant way to work better in CAS. For example, the RANUNI and other “RAN”
functions have been replaced with the newer RAND function that can generate independent random number streams in a multi-threaded environment.

Of course, obsoleted functions naturally lead one to the question: “how do I mitigate SAS 9 functions that are no longer supported (and continue running multi-threaded)?” When encountering a function that produces an error (see the ‘How to Proceed’ section below), you have several possible alternatives. Your best option is to remove the function because it’s no longer needed. If it is still needed, you can search for a suitable replacement function from the list of functions that are currently supported, remembering that you can switch back to single-threaded processing if necessary.

Your last option is to architect a set of DATA step code that is designed to reproduce the processing result that the obsolete function would have produced. Remember that the main purpose of SAS functions are to save you time and effort and provide a short-cut in coding so you don’t have to write your own routines. Some DATA step functions (with the exception of those listed above) can be reproduced with some level of coding effort – that’s the good news. Still, we recommend this only as a last resort, done after researching the feasibility, and then executed only by a seasoned SAS programmer. Should you need help, the SAS Professional Services Division has a number of consultants trained to do this type of work (see the “How to Proceed” section below).

No ‘Descending’ Option on a BY statement (within DATA step)

If using a BY statement within the DATA step that has the option DESCENDING indicated, then that option needs to be removed, as it is not currently supported in the SAS Viya 3.3 release.

Macro (code generator) processing changes

CALL EXECUTE, CALL SYMPUT and the SYMGET Function

Most macro functionality will continue to work in SAS Viya as it has in the past, with the exception of macro variables that are populated from DATA step row/observation processing. This affects primarily macro-related functions (see category #4 in the section above) and some CALL routines. Because DATA step macro-related functions are no longer supported, it is also true that CALL routines like SYMPUT and EXECUTE will not work as they once did. Basically, CAS (or the multi-threaded SAS Viya environment) does not include storage for macro variables. The reason for this is that each thread would need its own set of variables, however when the DATA step completes, it would not be possible to determine which individual thread’s value to use as the value for the macro variable. Therefore, anytime you take a row/column value out of the current Program Data Vector in SAS 9, and use it to initialize or instantiate a macro variable for use later in the program, that code needs to be removed and/or replaced.

Assuming you need data set values to continue running your data processing program dynamically (using data-driven values), then you have two possible solutions. The first option is to leave your existing code intact and run it as a single-threaded process. Alternatively, you can write the values to a new, small, in-memory table in CAS, then fetch the values in SAS and set the macro variables you need to set.

Procedure differences

PROC SORTs Are Not Needed and Can Be Removed

Because sorting is done in-memory, PROC SORTs are no longer needed within CAS, hence they should be removed from existing SAS 9 code. The only exceptions are sorts that support downstream Data steps and procedures which need to run single-threaded. If your PROC SORT is used primarily for de-duping using the ‘nodupkey’ or
‘nouniquekey’ option, then read the section below entitled “Nodupkey/nouniquekey replacement” under ‘Procedure-related processing changes.’

Miscellaneous PROC Similarities and Differences

When trying to migrate existing SAS 9 code to SAS Viya 3.3, there are three basic procedure groupings of which you need to be aware. The first of these are CAS-enabled procedures that run multi-threaded; the second group consists of non-CAS-enabled procedures that run single-threaded in SAS Viya; and a third group which represents procedures that must run against a SAS 9 Workspace server. The easiest way to determine which of your existing SAS PROCs belong to which group is to focus on the SAS product to which they belong. In general, any procedure that falls outside of the products listed below needs to be routed to a SAS 9 Workspace server if working primarily in a SAS Viya coding environment.

CAS-enabled PROCs are included in the following SAS Viya products:

- SAS® Econometrics
- SAS® Optimization
- SAS® Visual Analytics (provides common foundation PROCs and action sets)
- SAS® Visual Data Mining and Machine Learning (VDMML)
- SAS® Visual Forecasting
- SAS® Visual Statistics
- SAS® Visual Text Analytics

SAS Program Run-time Environment enabled PROCs are included as part of the following products:

- Base SAS®
- SAS/CONNECT®
- SAS/ETS®
- SAS® Forecast Server procedures (HPF)
- SAS/GRAPH®
- SAS/IML®
- SAS/OR®
- SAS/QC®
- SAS/STAT®

If your existing PROC belongs to one of the SAS Program Run-time Environment enabled products, then it is possible that a CAS-enabled (multi-threaded) replacement exists. An example of a SAS 9 procedure that has a multi-threaded version is PROC LOGISTIC which is found in SAS/STAT. PROC LOGISTIC is a single-threaded procedure that will run against the SAS 9 Program Run-time Environment in SAS Viya. However, this procedure has been multi-threaded and renamed in SAS Visual Statistics to PROC LOGSELECT, as has PROC REG (found in SAS/STAT) been replaced by the multi-threaded PROC REGSELECT. Not all options and statements may be available when comparing the single-threaded versus multi-threaded versions, but it may be worthwhile to explore the procedure lists of the various SAS Viya products to determine which SAS 9 procedures may benefit from substitution, since the associated performance gains will be substantial.

In terms of SAS Viya procedure commonalities with older single-threaded PROCs, there are a limited set of SAS Viya 3.3 procedures that duplicate procedures found in SAS 9. These include the following: APPEND, CONTENTS, COPY, DATASETS, DELETE, DS2, EXPORT, FORMAT, IMPORT, MEANS, OPTIONS, PRINT, PRINTTO, REPORT, SUMMARY, TABULATE, and TRANSPose (most of which are found in Base SAS originally). While these PROCs will run in SAS Viya against in-memory tables, some of them will only support single-threaded processing, meaning they will only run in SAS Viya’s SAS Programming Run-time Environment. [Remember, the SAS Program Run-time Environment is separate from CAS (and separate from a SAS 9 Workspace server) and runs all code identified to run single-threaded].
The following listed procedures will run multi-threaded: PROC COPY, DS2, MEANS, REPORT, SUMMARY, TABULATE and TRANSPOSE. Conversely, there are procedures that will only run single-threaded. These include: PROC APPEND, CONTENTS, DATASETS, DELETE, EXPORT, FORMAT, IMPORT, PRINT and PRINTTO. You may also find that not all options are present on some of the multi-threaded procedures, so check the latest SAS Viya documentation before proceeding.

User defined formats

One big area of change for SAS Viya is how formats are created, assigned to variables, and stored in CAS format libraries rather than SAS 9 catalogs. In general, any user-defined formats associated with specific variables have to be added to a data set before it is loaded into memory. The exception to this is using an ‘alterTable’ action to add formats to a variable after the table has been created. PROC DATASETS and PROC CASUTIL do not yet use the alterTable action for adding formats after table creation as of version 3.3, therefore the more common usage will be to add formats to tables during the table build process.

If you use a PROC FORMAT within your pre-existing SAS 9 program run-stream, and for multi-threaded processing to work, you need to be explicit about where the custom format will be stored, namely in a CAS format library. This means adding a new CASFMTLIB= option to your PROC FORMAT code. You can also add formats to a CAS format library if they are stored as .sashdat files (a new CAS feature). If you are using a format that was created in SAS 9 and you want to move it to CAS, the macro %UDFSEL can assist with that movement and copying. The CAS statement is used to list, manage, and update formats that have been loaded into a CAS format library.

Learn more about information on how CAS user-defined formats.

As mentioned above, there exists an interesting distinction between single-threaded and multi-threaded processing of SAS formats. If you want to have some of your DATA steps run in single-threaded mode, while other DATA steps run in multi-threaded mode, but you want both to use the same format, then you need to ensure that the format exists in both places (the CAS format library and the SAS format catalog). This is because single-threaded SAS Viya programs cannot read from the CAS format library, only from a SAS format catalog. Similarly, the CAS server can only read formats loaded into a CAS format library. Using the CASFMTLIB= in a PROC Format in CAS means that the formats are also created in the SAS Viya Workspace Server so you do not have to copy the formats manually from one place to the other.

For an excellent code example of how formats have changed in relation to their deployment and use in CAS, refer to this link.

Miscellaneous potential syntax changes

Among other possible impediments to running existing SAS 9 code in a multi-threaded environment, a few other syntax specifics need to be examined.

Unsupported SAS 9 formats

Only a few SAS 9 formats have been eliminated in SAS Viya and hence can no longer be used in CAS. Those formats include the following: WORDS, WORDSF, WORDDATE, WEEKDATE

Statement and Functions Supporting Inter-row Dependencies

An inter-row dependency is defined as a processing functionality that allows temporary information to be exchanged between different rows of a SAS table. It is important to acknowledge that in SAS Viya, as opposed to
SAS 9, the behavior of the DATA step is different depending on whether one of several inter-row functions or statements are invoked or utilized within your in-stream code. (This is not the case in SAS 9; only SAS Viya is affected.) Unlike default DATA step processing, which is automatically designed to reassemble and reorganize data back from each of the worker nodes, when some inter-row dependency commands execute in SAS Viya, the data comes back from the worker nodes and is simply appended together and not summed. This difference in terms of functionality is defined as partition or ‘by-group operations’ as opposed to what are called ‘full table operations.’

If your SAS 9 program needs full-table operations then you must process your data in two distinct steps, one that performs a local operation, followed by a second step that sums all local operations into a final result set. Examples of inter-row functions and statements include RETAIN, DIF and LAG functions; some processing using first. and last. automatic variables; and use of the SUM statement or SUM function (the latter of which both invoke an implicit RETAIN). More details are listed below.

**RETAI N** - the RETAIN statement has inter-row dependencies because it retains a value from one row to the next. In a SAS Viya multi-threaded DATA step, the rows of a table are processed using different threads, so their values and sort order is not the same when comparing data from node to node. For example, in SAS Viya multi-threaded DATA step, the rows of a table are processed using different threads, so a thread retains the values that it reads or computes independent of other threads. As a result of this disparate thread processing, all SAS 9 RETAIN statements need to be explicitly evaluated to determine whether or not they will produce the same DATA step output when running the same code in SAS Viya. Someone who is familiar with a specific SAS 9 DATA step program needs to examine the code and attempt to determine how having multiple processors working on the same problem might affect potential output or desired processing outcomes. Unfortunately, this examination is not strictly a mental exercise; in other words, it cannot be automatically determined without some sort of testing on at least a portion of the data.

**DIF and LAG functions** – Like the RETAIN statement, the DIF and LAG functions take data variable values from the current record (observation) and put them into a memory buffer so that their values can be retrieved when the next row or record is read. Since their intended effect in single-threaded code may not be reproduced exactly due to SAS Viya’s multi-threaded architecture, care needs to be taken to validate the end result – that single-threaded and multi-threaded run-streams both produce identical output. PROC COMPARE is a useful tool to compare different output files to see if they are the same or to understand how they might differ.

**Sum statement** – the Sum statement introduces an implicit RETAIN, and hence can produce non-matching results when comparing multi-threaded processing to prior single-threaded calculations. (See the section above related to RETAIN statement processing and the section below under Processing Validation called ‘Sum A Variable Across An Entire Table.’)

Temporary arrays

In the case of temporary arrays, where the DATA step statement is of the form,

```
Array arrayname {#of elements} _temporary_;
```

the array elements are only in existence for the duration of the DATA step; temporary array elements are not written out unless they are assigned to output table columns. In any SAS array, when any or all elements have been initialized (set to a specific value), all elements behave as if they have been named as part of a RETAIN statement. A potential issue arises if conditional logic is designed to operate on the entire table, but instead values of some rows are retained to others that were not originally intended because of partitioning to support multi-threading. As with other inter-row dependent statements and functions, you should check SAS Viya output for alignment and consistency in relation to SAS 9 processing.
Session encoding

The default encoding for CAS is UTF-8. SAS 9 used ASCII encoding where all characters were represented by a single byte. UTF-8, through multiple byte encoding, extends the range of possible character and special symbol representation beyond what was previously available providing better national character support through a wider range of options. Yet SAS 9, with its ASCII focus, relied on what are called 'byte semantic' functions which were created to work with ASCII data primarily. Traditionally, SAS programmers have used byte semantic functions to parse character strings (e.g. SUBSTR), determine string lengths (like LENGTH), or perform other operations that return positional information when specific strings are encountered (like INDEX, INDEXC, etc.). All of these functions treat strings as units of bytes, so care must be taken when using these on UTF-8 encoded data.

For the most part it should not matter if using traditional ASCII files as input, but if your SAS 9 code will run against new UTF-8 compatible data sets, it may be appropriate to consider switching to ‘K functions’ which are specifically designed to handle multi-byte data. SAS has supported a series of ‘K functions’ for years already in SAS 9.

Go here to read more about SAS K functions for processing UTF-8 encoded strings.

Unnecessary or obsolete statements (MODIFY, REMOVE, REPLACE)

The statements MODIFY, REMOVE and REPLACE are typically used to update a SAS data set in place without making another copy of the data set. Since SAS Viya does not allow rows to be updated once the data has been loaded up into shared memory, these statements are no longer valid for use in SAS Viya multi-threaded programs. If the SAS 9 code cannot be altered to accommodate these SAS Viya restrictions, then the code can run as a single-threaded process and the resulting set uploaded into SAS Viya memory.

WHERE clause for output tables

As with the prior discussion of the source (input) table and target (output) table needing to reside in CAS memory, if you use a WHERE clause on the target table specification, the process will automatically run as a single-threaded process (see Table 2 above). So, if you intend to alter your existing SAS 9 code so that it runs multi-threaded, you need to remove the WHERE= clause. WHERE= is okay if part of the input (source) file SET statement option.

Processing validation

As mentioned earlier, tasks like counting, do-loop processing, and random number generation execute on physically different threads, threads that process data sequentially and independently from other parallel threads. Because of the independent nature of individual threads, counts and derived output may be different when comparing results from one thread to another. For this reason, a lot of older processing features simply do not function as they once did. They have either been re-designed or simply restricted to run in SAS 9 only. That is the main result of the new architecture, namely the ability to support many simultaneous, but parallel processes that are working toward running a procedure or processing a DATA step.

DATA step data processing differences

Sum a variable across an entire table

Here is an example of a SUM statement,
which produces individual sums on every node or thread on which the DATA step ran. The SAS Viya documentation provides a nice example of this process. 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 full table, which has been distributed on a multi-node architecture. In the second step, the DATA step runs in a single thread on the smaller output data set to gather the final sum. The data collected in the final sum is the total number of homeowners in the data set mycas.purchase.

_N_ Usage

In SAS Viya, where there are multiple processors and threads, using _N_ for counting may not work the way it once did. In SAS 9, _N_ represented how many passes through the DATA step were made, most of the time equaling a count of row information. That’s how a lot of programmers have used it for single-threaded computations. Likewise in CAS, _N_ starts at 1 for each thread and then is incremented for each pass through the DATA step. However, in a multi-threaded environment, the counter can represent different record counts on separate threads, especially if there exist unique row counts due to the presence of a BY statement. Counts may end up being different on separate threads depending on how the data has been automatically distributed and how the code handles BY-group processing. So in SAS Viya, where there are multiple processors and threads, using _N_ for counting may return results that are not expected depending on how the data is structured and processed. Care needs to be taken in checking results to insure consistency.

BY group processing

When processing in multi-threaded mode, BY statements used in DATA step or procedure processing have the effect of dividing the data up into different groups of rows, which share the same values of the BY group variable. By groups are the basis for how CAS automatic shuffling determines how many threads to use. In general, the fewer the BY groups on the first variable listed, the fewer number of threads that are used. CAS tries to match the number of threads to the number of unique by groups so each group gets its own processor. Therefore, you may need to alter an existing BY-group sequence or order in SAS 9 code streams if high-cardinality variables are listed first as part of a BY statement. In other words, cardinality of BY groups can dramatically affect the performance of the CAS system and may need some tuning.

Procedure-related processing changes

Nodupkey/nouniquekey replacement

As I stated in the blog “Top 12 Advantages of SAS Viya,” one of the main reasons for running SAS Viya is the elimination of all the PROC SORTs in the code. You can save a significant portion of processing run-time cycles by not having to sort your data before using it. SAS Viya has new internal capabilities that perform optimized shuffling, as a replacement for formalized sorting. In the latter scenario, data is organized according to natural groupings or partitions in the data - that is until you specify a specific order to the data (you do that by introducing BY statements in PROCs or DATA steps).

Even though you might be tempted to remove all sorts, some serve the useful purpose of dedupping, or eliminating duplicate records in a SAS data set. If you have code that performs this type of work, then those uses of PROC SORT need to be replaced with a DATA step using first. and last. processing to replicate that processing and produce the same output. Here is an example of a PROC SORT that was replaced using a DATA step:
**Original SAS 9 (single-threaded) code:**

PROC SORT DATA=myCASlib.source NODUPKEY out=myCASlib.target;
   By VAR1 VAR2 VAR3;
run;

**Replace with SAS Viya multi-threaded DATA step:**

DATA myCASlib.target;
   Set myCASlib.source;
   By VAR1 VAR2 VAR3;
   IF FIRST.VAR3 than output;
run;

**Unsupported procedures**

Obviously, any procedure that is not SAS Program Run-time Environment enabled, or a component of High Performance Analytics, will not run in SAS Viya. For example, you cannot execute statistical procedures to run multi-threaded against CAS from SAS/STAT in SAS 9, the procedures associated with SAS Viya Data Mining and Machine Learning must be used instead. Many of the common statistical procedures have new SAS Viya analogs and more are coming on-line with every update or major release. In terms of common PROCs, all of the typical procedures, like PROC MEANS and PROC SUMMARY, can run as-is in SAS Viya. The exceptions to this are PROC FREQ and PROC UNIVARIATE, both of which are not CAS enabled (meaning no support for multi-threading), therefore they will run as single-threaded processes in the SAS Program Run-time Environment.

**How to Proceed**

It may seem like there is a lot of stuff to consider, and possibly a lot of re-engineering work necessary, to convert a SAS 9 run-stream so that it runs multi-threaded in SAS Viya. It is true that more complex programs utilizing special processing routines not supported in SAS Viya, may take too much effort to alter and hence should remain as single-threaded processes running on a SAS 9 Workspace server. However, for the bulk of SAS programs, the changes could be very minor, hence you should take advantage of the benefits that SAS Viya provides. Unfortunately, it is not easy to assess levels of effort without looking at how prior SAS 9 jobs were coded, especially since SAS DATA step code is so flexible and used to address a wide range of data processing problems.

Regardless, all migration efforts need to start with testing, an endeavor that includes assessment of syntax changes, as well as likely processing differences that might potentially lead to discontinuities in output when comparing SAS 9 with SAS Viya processing. One of the easiest ways to validate existing SAS 9 code is to do a syntax check on the code in SAS Viya before proceeding. Submitting an OPTIONS statement with the option ‘SYNTAXCHECK’ in SAS Viya can turn the syntax-checking facility on. It is the equivalent of running ‘options obs=0’ in SAS 9. The only risk in doing this is that the syntax checker will not process any data until it is overridden (with ‘NOSYNTAXCHECK’) and turned back to default data processing again by using another OPTIONS statement.
In order for the syntax checker to work in SAS Viya, you need to have at least one variable written to the target or output file. The syntax checker will only produce an error condition when it encounters problems, otherwise it will appear as though nothing actually ran or was processed. Some other potentially helpful options are ERRORCHECK='STRICT' or by using the option ERRORABEND, which terminates the processing after encountering the first error.

Especially when debugging possible changes in output, you may need to examine the automatic variable _nthreads_ to validate that the DATA step is running multi-threaded. To use _nthreads_ you can put the value out to the log file like this:

```
put 'The number of threads is ' _nthreads_;
```

_threadid_ is another useful automatic variable to tell where the by-group processing occurred.

If it seems that a particular SAS 9 task will not work in SAS Viya there may be a surrogate procedure or a different CAS statement or function that performs a similar set of work. There is also a rich set of SAS Viya actions available, which you can use as potential code replacements when no SAS Viya supported statement can be found. If you need assistance, SAS has excellent consultants who can assess the amount of effort involved in changing and altering your existing SAS code so that it can run in SAS Viya. Contact SAS Consulting at [http://sas.com/consulting](http://sas.com/consulting).