Paper SAS3376-2019

# Tips and Tricks for Building Business Applications with SAS® Visual Investigator

Gordon Robinson, SAS Institute Inc.

## ABSTRACT

SAS® Visual Investigator provides users with the flexibility to build applications that are customized to their business needs. This flexibility includes the ability to define how data is analyzed to generate alerts, how data is indexed to enable users to search, how data is presented to the users, what data users can enter into the system, and what workflows are in place with respect to data entry. As a result, SAS Visual Investigator can be used for applications in many different markets. This paper covers some of the best practices for building applications on top of SAS Visual Investigator to ensure maximum performance and utilization of available resources.

## INTRODUCTION

One of the key strengths of SAS Visual Investigator is the power that it puts in the hands of administrators to build solutions that can solve complex business problems. From being able to define the entity types that are available to users through to controlling how alerts are generated, there are a lot of controls available.

To quote a Spiderman movie (can't believe I managed to fit a quote from a superhero movie into a SAS Global Forum Paper), "With great power comes great responsibility." This is true of building a solution on top of SAS Visual Investigator. You need to think closely about what you are doing to ensure that you deliver a solution that meets the needs of the end users and performs efficiently.

When I try to explain this idea to people, I tend to compare it to a database management system. Out of the box it doesn't do anything. When you build a database, you must think carefully about how you structure your data within schemas. Consideration needs to be given to how the data will be accessed and which indexes will be needed to ensure it performs. SAS Visual Investigator requires the solution designer to not only think about all of the same things they would need to if they were designing a database, but also much more.

This paper will cover some of my thoughts and tips on best practices for building solutions using SAS Visual Investigator. If you are reading this before building a solution, I hope it helps you to deliver something that will amaze your end users.

## BUILDING A SOLUTION FOR GDR BANK

For the purposes of explaining the concepts in this paper, we are going to use the example of building a solution for a corporation called GDR Bank. In a lot of the areas covered in this document we will give examples of decisions we may make when building a solution for this bank.

### ENTITY TYPES

Every SAS Visual Investigator solution starts with data. Without data, the product is not going to do very much. GDR Bank has multiple external systems containing data that they want to be able to expose within the solution that we are building. This data will also be analyzed to identify threats that we need analysts to look at.

An entity type within SAS Visual Investigator is a group of database tables that all relate to a single object. For example, a solution might have a customer entity type that gathers information such as first name, last name and date of birth. Each customer can also have one or more addresses or telephone numbers.

Creating an entity relationship diagram of the entities you want in your solution is a critical first step. For example, GDR bank has customers, accounts, and investigations. This is a very simple example. However, we regularly see SAS Visual Investigator solutions with substantially more than 30 entity types, so putting thought into this step is critical.

## Internal Versus External Entity Types

For each entity type, you need to consider whether the end user should be able to create or edit the data. This consideration is crucial in deciding which of these entity types you need to use:

- **Internal Entity** – An internal entity type is one that is managed by SAS Visual Investigator. When you define this type within the administration section of the application, tables are automatically created within Postgres to host the data.
- **External Entity** – An external entity type is one that is designed to host data managed externally from SAS Visual Investigator. End users are not able to edit the contents of an external entity.

The choice between internal and external entities will influence how you get the data into the system. External entities allow for data to be loaded directly from the ETL job into the corresponding tables. Internal entities require that data be loaded through the SAS Visual Investigator API because more validation is required.

For GDR Bank, the Customer and Account data will be modeled as external entities, because the system of record for this data is external to SAS Visual Investigator. The Investigation entity will be modeled as an internal entity type, because we will want to be able to create and edit this data.

These are some tips for using Internal and External entities.

- Loading data through the API is slower than loading it directly to the tables.
- If you do need to load data into an internal entity, consider using the bulk API. This performs significantly better than loading data into each entity individually.
- Workflows are only available on internal entity types.

## Child Entity Versus Relationships

The next thing to consider is how we want to model the contents of the entities. For example, let's look at the GDR bank Customer entity type.

When we present a view of this data to a user, we will want to expose data points such as first name, last name and date of birth. We will also want to expose some data points that could have multiple values, such as accounts and telephone numbers.

At this point we need to consider how to model this content. SAS Visual Investigator has two ways to do this modeling:

- **Child Entities** – Child entities enable us to add repeating data to an entity type. You can have multiple different types of child entities within a single entity type. Child entities cannot have child entities of their own, which means only a single level of hierarchy is allowed.

- **Relationships** – Relationships enable a system administrator to define how entity types are related to each other.

When considering how to model the solution, these are some of the key factors are:

- Do we want the data to be searchable as part of the entity? if so, then we need to use child entities.
- When a user looks at a network diagram to visualize the relationships within the system, what entity types are they interested in seeing? If we need to expose something on a network diagram, then it needs to exist as a top-level entity.

Going back to our example, we would likely model Telephones as a child entity. Each customer can have zero or more telephone numbers that we want to store. We will want to be able to search this information as part of the Customer entity. For the purposes of this example, we can assume that the customer does not need to see the telephones on the network diagram.

We already discussed that we are going to model the Account entity as an external entity. We will also define a relationship between the Account entity and the Customer entity in order to allow the Account entity to be visualized on the network diagram. After this relationship is created, we can refer to the Account entity as being a related object of the Customer entity.

Defining the relationship between the Customer entity and the Account entity means that we can present lists of accounts when we view a customer's record. An end user would likely be unaware of the difference in how the Telephones and Account entities are modeled, because they would just see lists of data.

These are some key considerations about using child entities and relationships:

- The number of child tables the system must join on in order to index the data will impact performance. Carefully consider every child entity you add, especially if you have more than 10 child entities for one top-level entity.
- Related objects can be indexed independently. If you have data that does not frequently change, consider using related objects instead of child objects. This allows the entity to be indexed without having to include all the infrequently changing data, which makes indexing quicker.
- Using related objects allows security to be set independently on parts of the data.
- SAS Visual Investigator 10.5 can exclude certain entity types from the network diagram, which means you can have related objects that will not clutter what the user sees.

## INDEXING

The search index used by SAS Visual Investigator is critical, because it drives much of the functionality of the system, such as searching, maps, and network diagrams.

SAS Visual Investigator supports the ability to fully reindex an entity type. It can also perform incremental indexing if the entity type contains the relevant columns that provide details of what changed since the last indexing job was executed.

The indexing process involves looking at the metadata for an entity and performing the relevant SQL joins to pull together data to form JSON. This JSON is then sent to ElasticSearch for indexing.

As mentioned in the previous section, the more child objects an entity has, the more work the application must do in pulling the data together to form JSON. This is a critical factor in determining how long the indexing process will take.

The other aspect that must be considered is the amount of memory being used by the indexing process. The indexing job works in batches. It attempts to build up the JSON for a configurable batch size. While the job is doing this, it is also holding the data in memory. If the batch size is high and the amount of data in each instance is also high, the process might run out of heap space and start to produce errors in the logs. If you find this happening, you have several options:

1. Increase the heap size for the Datahub service. More memory is made available for the service to use while processing a batch.
2. Decrease the number of worker threads that are processing messages at one time.
3. Decrease the batch size for the indexing process. The service will then perform smaller batches and as a result, use less memory for each.

This is a summary of the various settings for controlling indexing:

- **config/svi-datahub/sas.svi-datahub.documentBatchSize**

  This controls the number of records processed in one batch. It defaults to 500. If you are creating large objects or objects with large numbers of child entities, then you might need to reduce this value. However, if you are creating small objects, then you might be able to speed up indexing by increasing this size.

- **config/svi-datahub/sas.svi-datahub.loader.topLevelDataLoadingThreads**

  This setting controls the total number of worker threads that can be run at any one time. This setting combined with documentBatchSize will largely control the amount of resources you will need at any one time whilst indexing data.

- **config/svi-datahub/sas.svi-datahub.loader.topLevelSplits**

  This setting controls the level of parallelism that will be used for processing each entity by setting the number of concurrent worker threads that will be used. Each of these workers then use the documentBatchSize setting to control how many objects they process at once. Setting this to 5 would mean that 5 threads would run concurrently at any one time to index an entity.

- **config/svi-datahub/sas.svi-datahub.loader.childDataLoadingThreads**

  This setting controls the number of threads that are used to load data for child entities, workspaces, attachments, and insights. This does not influence the total amount of memory used, because each worker still loads all the data for the batch size set. However, it does impact the speed of loading the data.

  In our example where we have ten parent worker threads, we would need this setting to be a minimum of 10 to ensure each parent thread could load data. However, it would ideally need to be higher than 10, depending on the number of child entities that your entity types have. For example, if you have an entity with five child entity types, then you would likely want to use a value of 8 for childDataLoadingThreads. This value allows for the five child entity types to be loaded at the same time as the workspaces, attachments, and insights.

For an example of using these settings, let's consider GDR Bank. We already know that we want to have the Customer and Account parent entity types, and that Telephones are going to be modeled as child entities.  Because this is a bank, we obviously want to store far more information than this, so suppose we had the following criteria:

- The Customer entity is the largest entity, and it contains ten child entity types, each containing significant amounts of text.
- Overall, we have 15 entity types.
- The entities other than Customer have an average of three child entity types.

Using the above criteria, we might want to consider the following settings:

- **sas.svi-datahub.documentBatchSize** – The Customer entity is fairly large, so we will leave this at 500.  Most solutions would be able to increase this to about 1000, because they have smaller entity types.
- **sas.svi-datahub.loader.topLevelSplits** – Setting this to 3 will mean that we have 3 worker threads available to concurrently process the data for each entity.
- **sas.svi-datahub.loader.topLevelDataLoadingThreads** – Setting this to 45 would allow the 3 worker threads for each entity to be run concurrently.  It’s likely that we may not have the resources available for that though, so we would choose a setting that limits the resource usage to what is available.  For example, we may choose to set this to 20.
- **sas.svi-datahub.loader.childDataLoadingThreads** – In our example, we know we need 13 threads for the Customer entity type (child entities, workspaces, insights, and attachments).  We have 14 other entity types with an average of three child entities each.  This means we would need 14 * 6 = 84 threads to be able to simultaneously load all of them.  Therefore, we need a total of 97 threads to enable us to get the best performance from indexing.  Again, we may need to limit this based on the specification of the hardware as having too many threads may be counterproductive.

The key to adjusting these settings is keeping a close eye on the heap usage of the data hub server.  If the heap usage is getting too close to the limit set for this service or if it runs out of memory, then you need to control how many things the process is doing at once by adjusting these settings.

## RDBMS INDEXES

The first question I ask anyone who tells me that indexing is slow is “*Do you have indexes on all foreign key columns in your RDBMS schema?*”.  This is one of the most common mistakes I see being made by solution designers.

In a lot of RBDMS systems, creating a foreign key constraint does **NOT** create an index. Rather, it is recommended that an index is created manually.

Indexing works by using the key for the primary object to identify the rows within child tables that are used to construct the JSON to be sent to ElasticSearch.  If there is no index on the foreign key columns, then a table scan will be performed on each child table for each object being indexed.  This will be slow.

Going back to the GDR Bank example, we know that Telephones are being modeled as child entities of the Customer entity type.  We must ensure that an index is created on the column containing the customer primary key within the Telephones table.

## ELASTICSEARCH

SAS Visual Investigator indexes entity data within ElasticSearch. Recent versions of the application give the user the ability to control parameters for how an index is created.

ElasticSearch works by breaking an index up into multiple chunks. These chunks are called shards. ElasticSearch manages these shards to keep them balanced and to move them between nodes when needed.

The number of shards used for an index is specified at creation time and cannot subsequently be changed without fully reindexing. This means that you need to give some thought to how the data for your entity will grow over the coming years.

Most environments will also want the ElasticSearch indexes to be highly available. To enable this, an administrator can specify the number of replica shards that will be created for the index. A replica is an exact copy of a shard that is stored on a different node within a cluster. If a node is lost, having a replica means we still have a copy of the shard. An additional benefit of a replica is that there are more copies of the shards that can be used for searching. Obviously, the more replicas we have, the more work ElasticSearch must do to keep these in sync and the more resources are used.

These are some recommendations for working with ElasticSearch and shards:

- When indexing data in ElasticSearch, the size of the resulting index on disk is about 4x the input data size.
- The maximum size on disk of an individual shard should not exceed 50GB. Thought should be given to the current size of an entity's data as well as future growth of the entity. By using the 4x multiplication factor, we can determine the maximum size on disk that we want to plan for, and then divide that value by 50GB to determine the number of shards that we need to allocate.
- Avoid creating shards that are too small. Each shard is essentially a Lucene index, which consumes file handles, memory, and CPU resources.
- We strongly recommend that there is one primary shard per index per node in your cluster. Each search will access each shard for an index. If they are all on the same node, there could be contention issues that will slow performance.
- The number of data nodes required in a cluster will be influenced by the largest number of shards needed for an entity as well as the total number of entities. The number of data nodes should be at a minimum 1 larger than the highest number of required primary shards. If the total number of entities is large, you may need more data nodes as you need space for all the shards.
- For best performance, the amount of heap space allocated to ElasticSearch on each node should be limited to around 26GB. The rest of the memory on the node should be available for the disk cache to ensure fast access to files.
- For each 1GB of heap, we have found that ElasticSearch can handle around 30GB of indexed data. Therefore, a single node should not have more than 780GB of indexed data. If you have more data than this, you should use multiple nodes.
- Version 10.5 of SAS Visual Investigator introduced the ability for administrators to control which columns within an entity are indexed for search. Data that is not indexed for search can still be displayed within pages. This is an important capability, because reducing what we send to ElasticSearch can speed up the indexing process and reduce the size of the resulting index.

For example, suppose that the total size in Postgres of the Customer data for GDR Bank is expected to be 20GB in 4 years. The resulting index we create would be around four times this size, making it around 200GB. Dividing this value by 50GB means that we would need 4 primary shards. As this is the largest entity in our system

## PAGE DESIGN

Critical to the success of a SAS Visual Investigator solution is how easily an end user can access the data that they need. SAS Visual Investigator is shipped with a custom page builder to provide solution designers with the ability to build out the user interface. Taking time to design interfaces well is critical to all applications.

These are some tips for designing pages within SAS Visual Investigator:

- Top-level tabs support lazy loading of controls, which means that the contents of the tab are only loaded when a user selects the tab. This helps with the page loading performance and means that the browser uses less memory. Use top-level tabs rather than the page tab control whenever possible.
- Design pages for to match their intended use:
  - Use slim and tall pages for inspector pages.
  - Use short and wide pages for summary views.
  - Do not use top-level tabs on pages that will be used in Inspector, Summary views, or Related Page controls.
- Use a consistent design across all pages. A key component of this is ensuring a consistent layout of toolbar buttons.
- Page contexts allow administrators to set which page will be displayed based on rules. These rules evaluate the groups to which a user belongs as well as the values within the object itself. This allows different pages to be used for different stages of a workflow for an entity.

## SCENARIO ADMINISTRATOR

SAS Visual Investigator provides users with the ability to analyze their data to detect threats and generate alerts. This is done through the Scenario Administration module.

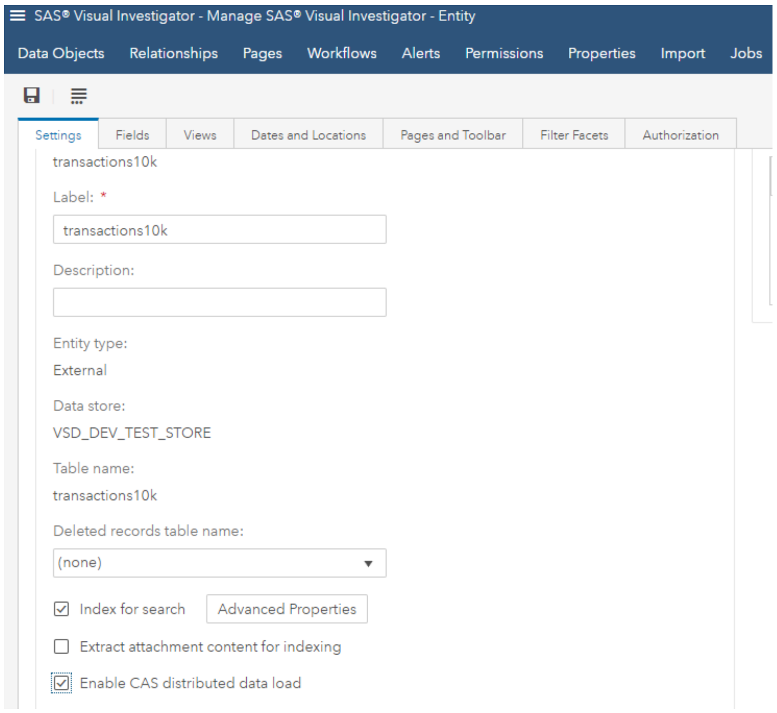
The alert generation process works by loading data into CAS, running analytics on the data to make some decisions, and then pushing any alerts into queues for SAS Visual Investigator to process.

### DISTRIBUTED DATA LOADS

In many scenarios, the amount of data you want to load into CAS to analyze can be substantial. CAS provides the ability to perform distributed loads of data, in which multiple worker nodes are present within a cluster.

When this mode is selected, CAS allows each of the worker nodes to run a query to load its section of the data. CAS performs this action by using mod values on the first integer column it finds in the table that it is loading. It is critical that this first column has an even distribution of values, otherwise too much data will be loaded on one node. In future releases of CAS, it will be possible to choose the column to use, rather than automatically choosing the first column.

Distributed data loading can be configured on the first tab for the entity within the Administration section of SAS Visual Investigator. See Figure 1.



transactions10k

Label: *

transactions10k

Description:

Entity type:
External

Data store:
VSD_DEV_TEST_STORE

Table name:
transactions10k

Deleted records table name:
(none)

☑ Index for search

☐ Extract attachment content for indexing

☑ Enable CAS distributed data load

**Figure 1. Enabling CAS Distributed Data Load**

## AGGREGATION BATCHES

Scenario Administrator uses the Aggregate action within CAS. If SAS Visual Investigator is trying to record details of the rows in the data that might have caused a rule to trigger, the Aggregate action can consume a lot of memory.

Because of this issue, you might want to organize your data into batches so that it is not all loaded into CAS at once. This obviously means that any scenarios and flows will take longer to run, but it also means that we can lower the RAM required by CAS to run them.

To configure aggregation batches, you need to configure two keys in Consul. In future releases we will build this into the Configuration section of SAS® Environment Manager.

**/config/svi-vsd-service/vsd/isActionInputTableSubsetting** – Set this to true to use the aggregation batch functionality.

**/config/svi-vsd-service/vsd/casMemorySoftCeilingGB** – Set this to the size, in GB, of the batches to load. The Scenario Administration service uses this setting to determine how to load the data into CAS.

## HEAP SPACE

Heap space is the memory allocated to the Java Virtual Machine (JVM) by the operating system.

The SAS Viya architecture uses lots of services. Each of these services is assigned an amount of heap space with which to work. This value controls how much memory the service can use to do its job.

The business problem for the solution you are building controls how users interact with the system. The design of the solution together with the usage patterns influence how much heap space you need to assign to different services.

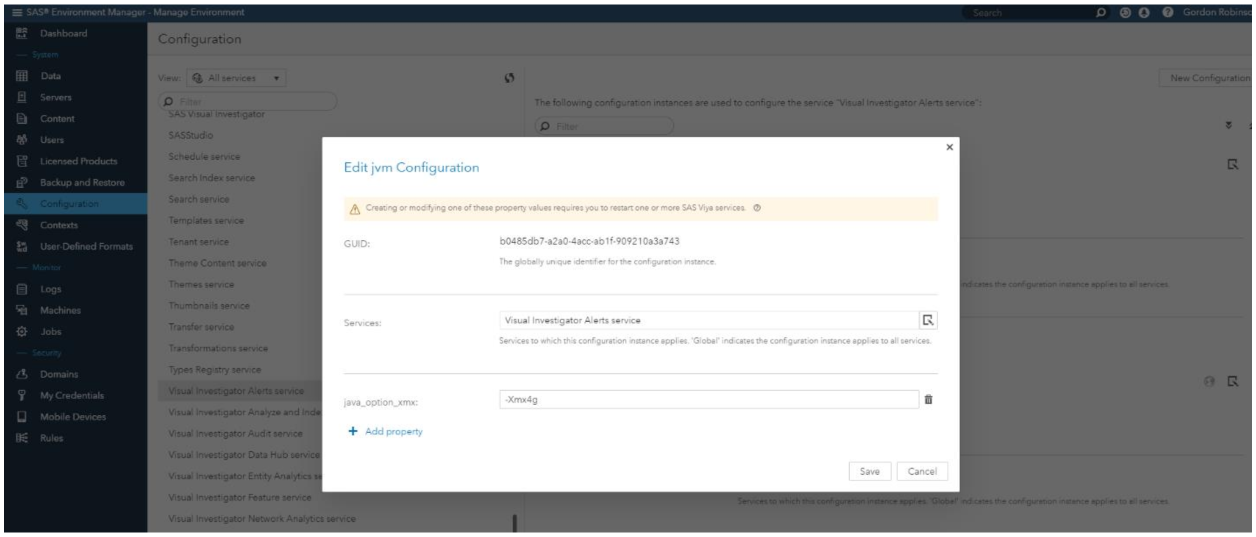
For example, if the GDR Bank solution was largely focused on triaging alerts, then we might want to increase the heap space allocated to the svi-alert service. If your solution is largely focused on searching, you might want to increase the heap space allocated to the svi-sand service.

The heap space for a service is configured through the Configuration section of SAS Environment Manager, as shown in Figure 2. In this section, you can select the service that you want to change. For example, the JVM section contains the "java_options_xmx" setting. This setting uses the following format:

*-xmx<size>*

For example, "-xmx4g" sets the heap space to 4GB. "-xmx256m" sets the heap space to 256MB.

If you see any errors with SAS Visual Investigator and you see "Out of memory" errors within the log for a service, then you might want to consider changing the heap allocation. Changing the heap space requires the service to be restarted.


Edit jvm Configuration

Creating or modifying one of these property values requires you to restart one or more SAS Viya services.

GUID: b0485db7-a2a0-4acc-ab1f-909210a3a743
The globally unique identifier for the configuration instance.

Services: Visual Investigator Alerts service
Services to which this configuration instance applies. 'Global' indicates the configuration instance applies to all services.

java_option_xmx: -Xmx4g

**Figure 2. Configuring the Heap Space for the Alert Service**

## DATABASE CONNECTIONS

We've discussed already that the SAS Viya architecture uses many services. Almost all these services require a connection to a database to be able to do their work.

To increase performance, the service uses connection pools. When a connection is established to the database, it is held in the pool. This means that the next time a connection is required, it can use the existing connection rather than trying to connect again. This improves performance and can reduce the overall number of connections used to the database.

If connections are not used, they become idle. The pool can be configured to determine how many idle connections to keep and how long they should remain idle before they are closed.

If all the connections in a pool are being used, then a service is blocked from waiting for one to become available. Because of this behavior, the usage patterns of the system might indicate you should adjust the number of connections held in the pool.

The following settings are available to be configured for each service:

- **datasource.tomcat.initialSize** – Specifies how many connections are established on creation of the pool. The defaults value is 2.
- **datasource.tomcat.maxActive** – Specifies the maximum number of connections the pool can hold. The default value is 10.
- **datasource.tomcat.maxIdle** – Specifies the maximum number of idle connections to hold onto within the pool. If the pool has more idle connections than this value, then the pool releases some of them. The default value is 2.
- **datasource.tomcat.minIdle** – Specifies the minimum number of idle connections to hold onto within the pool. This value is the minimum number of total connections for the pool. The default value is 2.

*SAS Viya Administration: Tuning* provides details about changing these settings for services in the topic "Tuning: JDBC Connection Pool."

If you encounter any issues while under load and find errors in the service logs due to not being able to get database connections, then you might want to change these connection settings.

## CONCLUSION

SAS Visual Investigator provides a flexible framework to enable complex business problems to be solved. Understanding how to build a good solution and to ensure it performs well is critical to being successful.

This paper gives details of some of the levers and switches that can be used to modify the system. My hope is that it will be used to help guide the development of several new solutions and result in some extremely happy users.

## REFERENCES

Hundley, Ben. "Optimizing Elasticsearch: How Many Shards per Index?" Available https://qbox.io/blog/optimizing-elasticsearch-how-many-shards-per-index. July 29, 2015. Accessed on March 22, 2019.

Elastic Inc. 2019 "Setting the heap size." In Elasticsearch Reference 6.6. Mountain View, CA: Elastic Inc. Available https://www.elastic.co/guide/en/elasticsearch/reference/current/heap-size.html (accessed March 22, 2019)

SAS Institute Inc. 2018. "Tuning: JDBC Connection Pool." In SAS Viya 3.4 Administration: Tuning. Cary, NC: SAS Institute Inc. Available https://documentation.sas.com/?cdcId=calcdc&cdcVersion=3.4&docsetId=caltuning&docsetTarget=p19ysl1cbx92nqn13gs5hp2ewatk.htm&locale=en (accessed March 22, 2019).

## CONTACT INFORMATION

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

Gordon Robinson
SAS Institute Inc.
Cary NC 27513
919-531-3038
gordon.robinson@sas.com