Paper 3179-2019

Can't Find The Right CAS Action? Create Your Own Action Using CASL

Brian Kinnebrew, SAS Institute Inc.

ABSTRACT

SAS® Cloud Analytic Services language (CASL) is a language specification in SAS® Cloud Analytic Services (CAS) that provides a programming environment to manage the execution of CAS actions. The results of those CAS actions are processed with functions and expressions to prepare the parameters for subsequent CAS actions. CASL and the CAS actions provide the most control, flexibility, and options when interacting with CAS. In addition to the many CAS actions supplied by SAS®, as of SAS® Viya® 3.4, you can create your own CAS actions using CASL. You specify the interface to your new CAS action, provide a name and parameters, and supply the CASL code that implements your new CAS action. Once created, your CAS action is available for use by any user that has permission. Your CAS actions look and feel just like CAS actions provided by SAS. Jump on board and find out how to create your own CAS actions that are tailor-made for your needs!

INTRODUCTION

CASL is a language specification that can be used by the SAS client and other clients to interact with and provide easy access to Cloud Analytic Services (CAS). CASL is a statement-based scripting language with many uses and strengths including:

- Specifying CAS actions to submit requests to the CAS server to perform work and return results.
- Evaluating and manipulating the results returned by a CAS action.
- Creating user-defined CAS actions and functions and creating the arguments to a CAS action.
- Developing analytic pipelines.

CASL uses the CAS procedure, which enables you to program and execute CAS actions from the SAS client and use the results to prepare the parameters for a subsequent CAS action. A single PROC CAS statement can contain several CASL programs. With the CAS procedure you can run any CAS action supported by the server, load new CAS action sets into the server, use multiple sessions to perform asynchronous execution and operate on parameters and results as variables using the function expression parser.

CASL, and the CAS actions, provide the most control, flexibility, and options when interacting with CAS. Combining DATA Step, CAS-enabled PROCS and CASL provides optimal flexibility and control. CASL works well with traditional SAS interfaces and the Base SAS language.

Each CAS action belongs to a CAS action set. Each CAS action set is further categorized by product including Visual Analytics, Visual Statistics, and Visual Data Mining and Machine Learning. Please refer to the documentation for a list of each action set by product. In addition to the many CAS actions supplied by SAS, as of SAS® Viya® 3.4, you can create your own CAS actions using CASL. Developing and using your own CAS actions allows you to further customize your code and increase your ability to work with CAS in a manner that best suits you and your organization.

ABOUT USER-DEFINED CAS ACTION SETS

Developing a CASL program that is stored on the CAS server for processing is defined as a user-defined CAS action set. Since the CAS action set is stored on the CAS server, the CASL statements can be written once and executed by many users. This can help reduce the exchange of programs between users that execute common code. **Note**: You cannot add, remove, or modify a single user-defined CAS action. The entire user-defined CAS action set must be redefined.

As a best practice, prior to creating any user-defined CAS actions, test your routines and functions first to ensure they execute successfully in CAS when submitted from the programming client.

DEVELOPING USER-DEFINED CAS ACTIONS

To create user-defined CAS actions, use the defineActionSet CAS action in the builtins CAS action set and add your code. You also need to modify your code to use CASL functions such as SEND_RESPONSE, so the resulting objects on the server are returned to the client.

COMBINE SAS-PROVIDED CAS ACTIONS

One technique for creating your own CAS actions is to combine one or more SAS provided CAS actions into a single user-defined CAS action. This method allows you to execute one CAS procedure and call one CAS action which subsequently calls all the SAS provided CAS actions contained within. This is extremely beneficial if you repeatedly run many of the same CAS actions against a CAS table. Suppose your organization has data on which it needs to regularly generate summary statistics, a correlation analysis, and a Random Forest model. Lastly, the model generated is used to provide scoring and probability information. This entire routine can be rolled up into a single user-defined CAS action. An example of this is illustrated below:

```
proc cas;
   builtins.defineActionSet / name="udActionSet"
      actions={
         {
    name="statmodel"
    desc="
         1. Generate multi-dimensional descriptive statistics of numeric
         variables
         2. Generate Pearson Product-Moment correlation coefficients
         3. Generate HTML output for summary and correlation results
         4. Run a Random Forest model with training and scoring - generate
         OOB errors and variable importance analytics;
         generate prediction errors and scoring information analytics
         "
    parms={
         {name="caslib" type="string" required=TRUE}
         {name="intable" type="string" required=TRUE}
         {name="grpvar1" type="string" required=TRUE}
         {name="grpvar2" type="string" required=TRUE}
         {name="avar1" type="string" required=TRUE}
         {name="avar2" type="string" required=TRUE}
         {name="avar3" type="string" required=TRUE}
         {name="avar4" type="string" required=TRUE}
         {name="outtable" type="string" required=TRUE}
          }
    definition="
```

```
            simple.summary result=a /
            table={caslib=caslib, name=intable, groupby={name={name=grpvar1},
            name={name=grpvar2}}},
            inputs={{name=avar1}},
            casout={caslib=caslib, name=outtable, replace=true};
            send_response(a);

            simple.correlation result=b /
            inputs={avar1, avar2}
            pairWithInput={avar3, avar4}
            table={caslib=caslib name=intable};
            send_response(b);

            table.fetch result=c /
            table={caslib=caslib, name=outtable,
            vars={{name=grpvar1}, {name=grpvar2}, {name='_Min_'},
            {name='_Max_'}, {name='_NObs_'},
            {name='_Mean_', format='6.0'}, {name='_Std_', format='6.0'}}},
            index=false;
            send_response(c);

            decisionTree.forestTrain result=d / table={name=intable,
            caslib=caslib},
            target='BAD',
            inputs={{name='LOAN'}, {name='MORTDUE'}, {name='VALUE'},
            {name='REASON'}, {name='JOB'}, {name='DELINQ'}, {name='NINQ'}},
            nominals={{name='LOAN'}, {name='MORTDUE'}, {name='VALUE'},
            {name='REASON'}, {name='JOB'}, {name='DELINQ'}, {name='NINQ'}},
            nBins=20, maxLevel=21, maxBranch=2, leafSize=5, crit='VARIANCE',
            missing='USEINSEARCH', vote='PROB', minUseInSearch=1,
            binOrder=true, varImp=true,
            casOut={name='mymodel', caslib=caslib, replace=true},
            mergeBin=true, encodeName=true, nTree=100, seed=17,
            bootstrap=0.6, oob=true;
            send_response(d);

            decisionTree.forestScore result=e / table={name=intable,
            caslib=caslib},
            modelTable={name='mymodel', caslib=caslib}, copyVars={'id'},
            vote='PROB',
            treeError=true, encodeName=true;
            send_response(e);
                  "
            }
                  };
   quit;
```

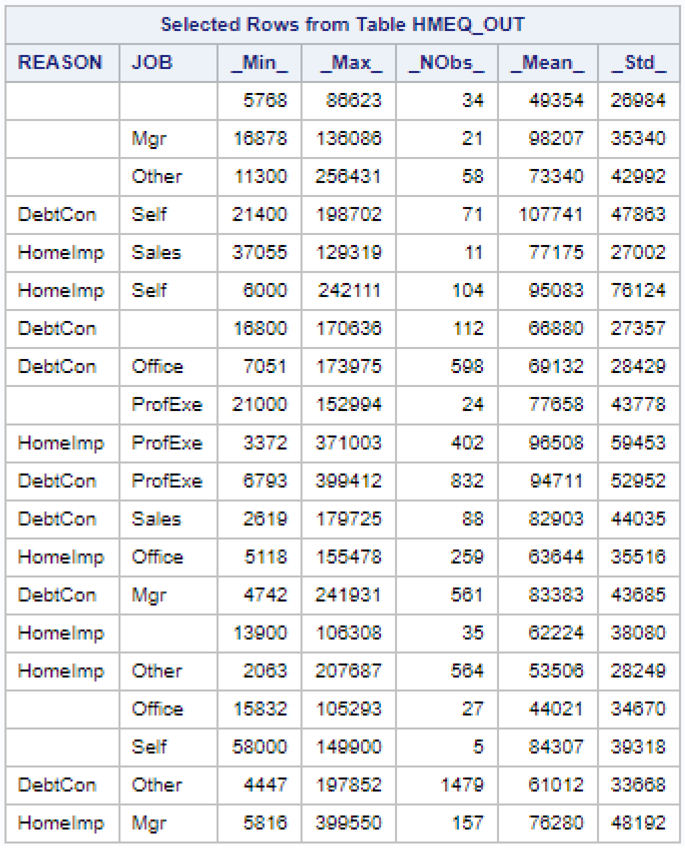
In this example, a single user-defined CAS action named ***statmodel*** has been created by using the SAS-provided CAS actions summary, correlation, fetch, forestTrain, and forestScore. Each of these CAS actions return the information outlined in the description section of the PROC CAS statement above. The new user-defined CAS action is also part of the new user-defined CAS action set ***udActionSet***. When the CASL code is executed, the log will display a note that the new CAS action set has been added.

```
NOTE: Added action set 'udActionSet'.
{actionset=udActionSet}
```

Once the new CAS action set and CAS action have been created and added, call them via a PROC CAS statement. Specify the user-defined CAS action set, user-defined CAS action(s), and parameters for each:

```
proc cas;
   session mysess;
   udActionSet.statmodel /
         caslib="public"
         intable="hmeq"
         grpvar1="reason"
         grpvar2="job"
         avar1="mortdue"
         avar2="derog"
         avar3="delinq"
         avar4="bad"
         outtable="hmeq_out";
quit;
```

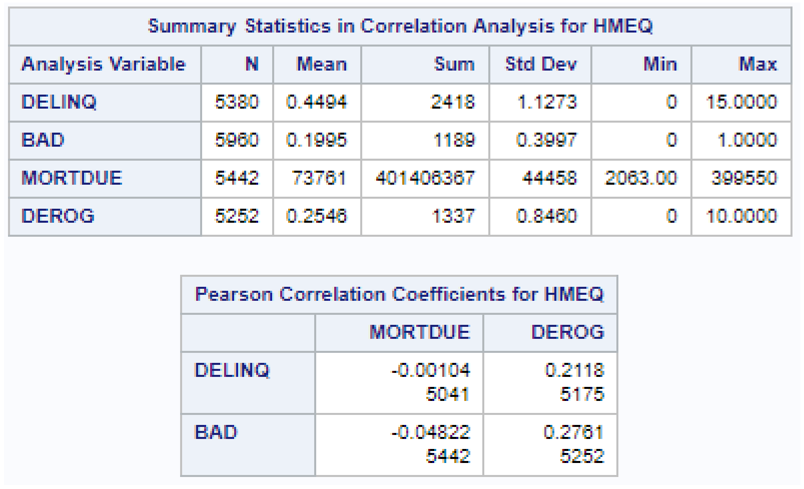
The parameters that are passed when calling the ***statmodel*** CAS action are defined in the parms section above. Calling the ***statmodel*** CAS action produces the output shown in the following tables:



Selected Rows from Table HMEQ_OUT

REASON	JOB	_Min_	_Max_	_NObs_	_Mean_	_Std_
		5768	86623	34	49354	26984
	Mgr	16878	136086	21	98207	35340
	Other	11300	256431	58	73340	42992
DebtCon	Self	21400	198702	71	107741	47863
HomeImp	Sales	37055	129319	11	77175	27002
HomeImp	Self	6000	242111	104	95083	76124
DebtCon		16800	170636	112	66880	27357
DebtCon	Office	7051	173975	598	69132	28429
	ProfExe	21000	152994	24	77658	43778
HomeImp	ProfExe	3372	371003	402	96508	59453
DebtCon	ProfExe	6793	399412	832	94711	52952
DebtCon	Sales	2619	179725	88	82903	44035
HomeImp	Office	5118	155478	259	63644	35516
DebtCon	Mgr	4742	241931	561	83383	43685
HomeImp		13900	106308	35	62224	38080
HomeImp	Other	2063	207687	564	53506	28249
	Office	15832	105293	27	44021	34670
	Self	58000	149900	5	84307	39318
DebtCon	Other	4447	197852	1479	61012	33668
HomeImp	Mgr	5816	399550	157	76280	48192

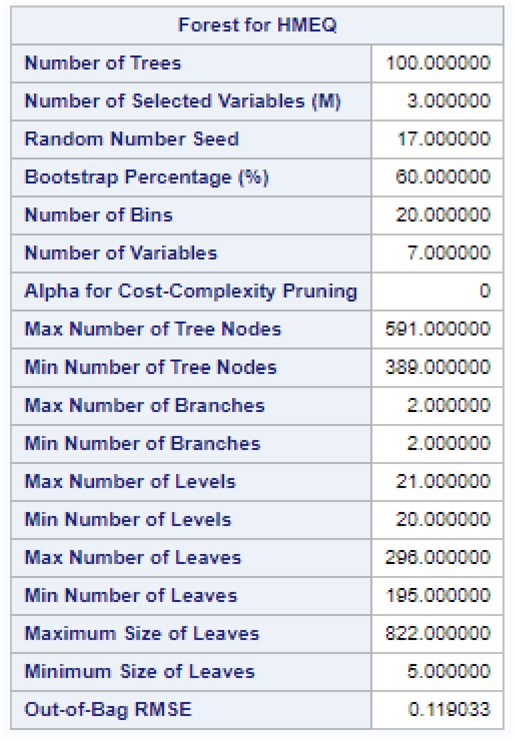
Table 1. Multi-Dimensional Descriptive Statistics Generated By The SUMMARY Action



Summary Statistics in Correlation Analysis for HMEQ						
Analysis Variable	N	Mean	Sum	Std Dev	Min	Max
DELINQ	5380	0.4494	2418	1.1273	0	15.0000
BAD	5960	0.1995	1189	0.3997	0	1.0000
MORTDUE	5442	73761	401406367	44458	2063.00	399550
DEROG	5252	0.2546	1337	0.8460	0	10.0000

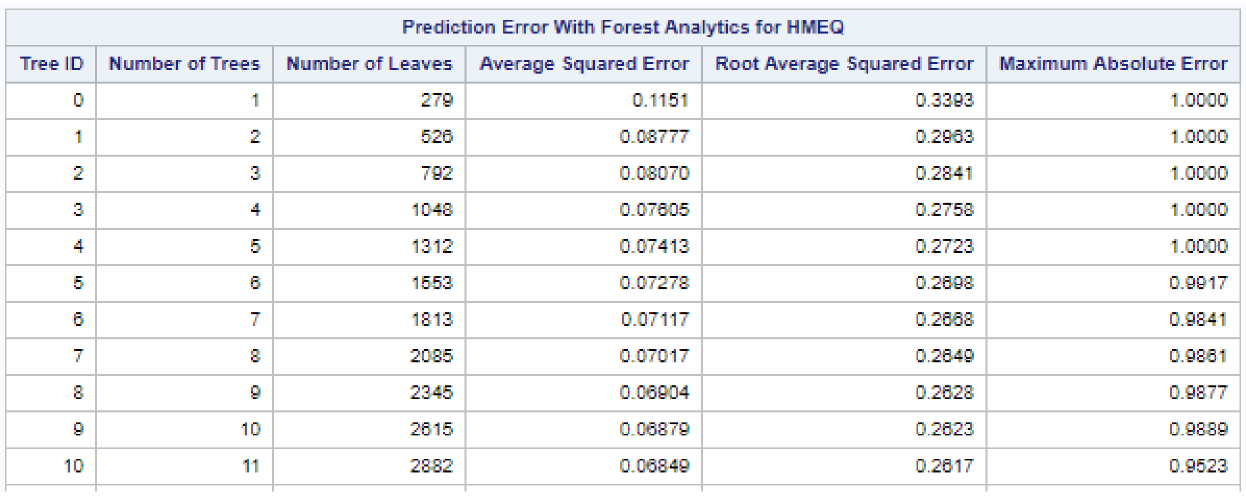
Pearson Correlation Coefficients for HMEQ		
	MORTDUE	DEROG
DELINQ	-0.00104 5041	0.2118 5175
BAD	-0.04822 5442	0.2761 5252

Table 2. Summary Statistics and Correlation Coefficients Generated By The CORRELATION Action



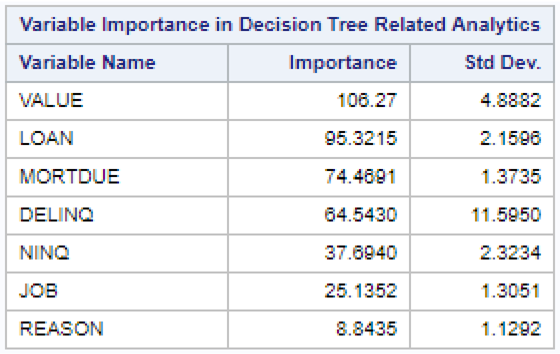
Forest for HMEQ	
Number of Trees	100.000000
Number of Selected Variables (M)	3.000000
Random Number Seed	17.000000
Bootstrap Percentage (%)	60.000000
Number of Bins	20.000000
Number of Variables	7.000000
Alpha for Cost-Complexity Pruning	0
Max Number of Tree Nodes	591.000000
Min Number of Tree Nodes	389.000000
Max Number of Branches	2.000000
Min Number of Branches	2.000000
Max Number of Levels	21.000000
Min Number of Levels	20.000000
Max Number of Leaves	296.000000
Min Number of Leaves	195.000000
Maximum Size of Leaves	822.000000
Minimum Size of Leaves	5.000000
Out-of-Bag RMSE	0.119033

Table 3. Decision Tree Specifications Generated By The FORESTTRAIN Action



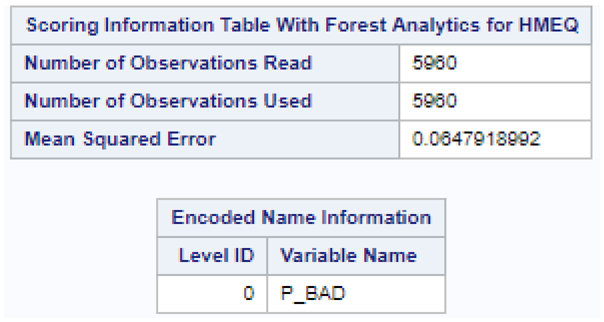
Prediction Error With Forest Analytics for HMEQ					
Tree ID	Number of Trees	Number of Leaves	Average Squared Error	Root Average Squared Error	Maximum Absolute Error
0	1	279	0.1151	0.3393	1.0000
1	2	526	0.08777	0.2963	1.0000
2	3	792	0.08070	0.2841	1.0000
3	4	1048	0.07605	0.2758	1.0000
4	5	1312	0.07413	0.2723	1.0000
5	6	1553	0.07278	0.2698	0.9917
6	7	1813	0.07117	0.2668	0.9841
7	8	2085	0.07017	0.2649	0.9861
8	9	2345	0.06904	0.2628	0.9877
9	10	2615	0.06879	0.2623	0.9889
10	11	2882	0.06849	0.2617	0.9523

Table 4. Partial Result: Prediction Error Results Generated By The FORESTTRAIN Action



Variable Importance in Decision Tree Related Analytics		
Variable Name	Importance	Std Dev.
VALUE	106.27	4.8882
LOAN	95.3215	2.1596
MORTDUE	74.4691	1.3735
DELINQ	64.5430	11.5950
NINQ	37.6940	2.3234
JOB	25.1352	1.3051
REASON	8.8435	1.1292

Table 5. Variable Importance Results Generated By The FORESTTRAIN Action



Scoring Information Table With Forest Analytics for HMEQ	
Number of Observations Read	5960
Number of Observations Used	5960
Mean Squared Error	0.0647918992

Encoded Name Information	
Level ID	Variable Name
0	P_BAD

Table 6. Scoring Information Results Generated By The FORESTSCORE Action

APPLY YOUR OWN CODE

Another method for creating user-defined CAS actions is to apply your own code, functions, and statements instead of SAS-provided CAS actions. This gives you the flexibility to truly customize your code. Suppose you want to calculate the speed of sound in meters per second given a certain air temperature.

```
proc cas;
   builtins.defineActionSet / name="speed"
      actions={
         {
   name="tc"
   desc="Convert temperature in Fahrenheit (tf) to Celsius (tc)"
   parms={
         {name="tf" type="int64" required=TRUE}
         }
   definition="
         tc = (tf - 32) * (5/9);
         send_response({temp=tc});
              "
          }
          {
   name="sound"
   desc="Calculate Speed of Sound in Meters per Second at a Certain
         Air Temperature in Degrees Celsius
        "
   parms={
         {name="tc" type="double" required=TRUE}
         }
   definition="
         s = 331 + (.6 * tc);
         send_response({spd=s});
              "
         }
              };
quit;
```

In this example, two user-defined CAS actions named ***tc*** and ***sound*** have been created by incorporating user-defined code. Similar to the first example, these CAS actions return the information outlined in the description sections of the PROC CAS statement above. The new user-defined CAS actions are now part of the new user-defined CAS action set ***speed***. When the CASL code is executed, the log will display a note that the new CAS action set has been added.

```
NOTE: Added action set 'speed'.
{actionset=speed}
```

Once the new CAS action set and CAS actions have been created and added, they can be called individually or all at once via PROC CAS. Specify the user-defined CAS action set, user-defined CAS action(s), and parameters for each:

```
proc cas;
   session mysess;
   do n over {0 32 60 68 80 100};
      speed.tc result=a / tf = n;
        temp=a.temp;
        print "At " put(n, best2.) " degrees Fahrenheit the
        temperature in Celsius is " compress(put(temp, best3.));
    end;
```

```
         print " ";

      do n over {-18 0 16 20 27 38};
          speed.sound result=z / tc = n;
            spd=z.spd;
            print "At " put(n, best2.) " degrees Celsius the speed of
            sound in meters per second is " compress(put(spd, best3.));
      end;
   quit;
```

This example uses a range of values for the temperature conversion. Subsequently, the range of converted temperatures are used as input for the speed of sound calculation. The output from this example is written to the SAS log and is shown below:

```
At  0 degrees Fahrenheit the temperature in Celsius is -18
At 32 degrees Fahrenheit the temperature in Celsius is 0
At 60 degrees Fahrenheit the temperature in Celsius is 16
At 68 degrees Fahrenheit the temperature in Celsius is 20
At 80 degrees Fahrenheit the temperature in Celsius is 27
At 100 degrees Fahrenheit the temperature in Celsius is 38

At -18 degrees Celsius the speed of sound in meters per second is 320
At 0 degrees Celsius the speed of sound in meters per second is 331
At 16 degrees Celsius the speed of sound in meters per second is 341
At 20 degrees Celsius the speed of sound in meters per second is 343
At 27 degrees Celsius the speed of sound in meters per second is 347
At 38 degrees Celsius the speed of sound in meters per second is 354
```

Output 1. Output from PROC CAS calling user-defined CAS actions *tc* and *sound*

SAVING AND LOADING USER-DEFINED CAS ACTIONS

User-defined CAS action sets only exist in the current CAS session. If the current CAS session is terminated, the program used to create the user-defined CAS action set must be executed again unless an in-memory table is created from the CAS action set and the in-memory table is subsequently persisted to a SASHDAT file. **Note:** SASHDAT files can only be saved to path-based caslibs such as Path, DNFS, and HDFS. To create an in-memory table and persist it to a SASHDAT file, use the actionSetToTable and save CAS actions.

```
   proc cas;
      builtins.actionSetToTable /
      actionSet="speed"
      casOut={caslib="casuser" name="speed" replace=True};
   run;

      table.save /
      caslib="casuser"
      table="speed"
      name="speed.sashdat"
      replace=True;
   quit;
```

The actionSetToTable CAS action creates an in-memory table from the user-defined CAS action set ***speed*** and places it in the Casuser caslib. The save CAS action subsequently saves this in-memory table to the caslib's data source as a SASHDAT file. In this case, speed.sashdat has been saved to the Casuser caslib. After executing this code, the SAS log will display a note that the new SASHDAT file has been saved.

```
NOTE: Cloud Analytic Services saved the file speed.sashdat in
caslib CASUSER(userid).
{caslib=CASUSER(userid),name=speed.sashdat}
```

Now that the CAS action set is saved, it can be used whenever necessary. To use the user-defined CAS action set, it needs to be restored from the saved SASHDAT file. This is done with the actionSetFromTable CAS action.

```
proc cas;
   builtins.actionSetFromTable /
   table="speed.sashdat"
   name="speed";
quit;
```

After executing this code, the SAS log will display a note that the CAS action set has been restored and added. You are now free to use the CAS action set and the CAS actions within it.

```
NOTE: Added action set 'speed'.
{actionset=speed}
```

CONCLUSION

CASL provides the most flexibility and functionality when interacting with Cloud Analytic Services (CAS). This paper has demonstrated one of the many versatile features in CASL, the ability to create your own CAS actions. The many SAS provided CAS actions deliver an array of capabilities for programming in CAS. CASL takes the next step by offering the means to customize your code by creating custom CAS actions and using them in different clients and with different languages such as Python, R, Lua, Java, and REST APIs. Both SAS provided and user-defined CAS actions can be used alone or in conjunction with other CAS enabled PROCS and Base SAS code to maximize the CAS programming experience.

RECOMMENDED READING

- *SAS Global Forum 2019 Session: 3177 - Got Results? Let CASL Help You Turn them into Superior Reports*
- *SAS Global Forum 2019 Session: 3040 – What is CASL?*
- *Getting Started with CASL Programming 3.4*
- *SAS® Cloud Analytic Services 3.4: CASL Reference*
- *SAS® Cloud Analytic Services 3.4: CASL Programmer's Guide*
- *Writing User-Defined Actions*

CONTACT INFORMATION

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

Brian Kinnebrew
SAS Institute Inc.
469.801.7461
brian.kinnebrew@sas.com
www.sas.com