

A New Gateway to Open Source in SAS Viya

Jim Box, SAS Institute, Cary, NC, USA

ABSTRACT

PROC GATEWAY is a new procedure in SAS® Viya® that allows you to run code written in other programming languages directly from within SAS. We will show how GATEWAY can be used to integrate R and Python code in your SAS Programs, allowing you to take advantage of new open-source procedures to solve more complex coding tasks. We'll also show you how to take advantage of Viya's cloud capabilities to submit open-source programs in a way to take advantage of multi-threading, allowing you to really leverage large data or complex analyses.

INTRODUCTION

There have been a variety of ways to execute Open Source code languages in SAS. PROC IML has allowed programmers to submit R code in SAS V9 since version 9.2, which came out at the latter part of 2010. PROC Python has existed in the SAS Viya environment since version 4. Both of these methods still work fine and will submit the code in a single-threaded execution engine. The CAS Gateway Action set was added to Viya at the end of 2023 and allows you to submit code written in Python or R to the CAS serve and to be able to execute in a multi-threaded environment. PROC Gateway was added to make it a bit simpler to interact with these CAS Action sets. For this paper, we'll start out looking at single threaded R environments and executing code without PROC IML. All of the code we use will be available on a GitHub page, see end of paper for location.

EXECUTING SINGLE THREADED R CODE

UNDERSTANDING THE R ENVIRONMENT

SAS allows you to have multiple R environments to which to submit code, and you can switch between them by changing the R_HOME option (provided to you by your administrator). The code will look something like this:

```
options set=R_HOME="/opt/sas/viya/home/sas-pyconfig/default_r/lib64/R";
```

Changing that path to a different one will point your code to a different R installation, and you can switch back and forth in a program. When getting ready to run R code, it's helpful to get an understanding of what version and what packages you have installed. To do that, you can run code (Figure 1) to get a printout of versions and packages.

```

1  proc iml;
2      submit / R;
3          ver = R.version.string
4          vdf = data.frame(version = ver)
5          pkg = installed.packages()[,c(1,3)]
6      endsubmit;
7      call ImportDataSetFromR("R_pkg", "pkg");
8      call ImportDataSetFromR("R_ver", "vdf");
9      quit;
10
11  data _null_;
12      set R_ver;
13      CALL SYMPUTX("RVer",version);
14      RUN;
15
16
17      TITLE "R Version: &Rver";
18      proc print data = R_pkg; run;

```

Figure 1: R Version and Package Checker

The version on my environment is 4.3.3 and has 189 packages installed (Figure 2).

R Version: R version 4.3.3 (2024-02-29)

Obs	Package	Version
1	arrow	18.1.0
2	askpass	1.2.1
3	assertthat	0.2.1
4	backports	1.5.0
5	base	4.3.3
6	base64enc	0.1-3
7	BH	1.87.0-1
8	BiocManager	1.30.25
9	bit	4.5.0.1
10	bit64	4.5.2
11	blob	1.2.4
12	boot	1.3-29
13	broom	1.0.7
14	bslib	0.8.0
15	cachem	1.1.0
16	callr	3.7.6
17	caret	7.0-1
18	cellranger	1.1.0
19	chk	0.9.2
20	class	7.3-22
21	cli	3.6.3

Figure 2: Version and Packages Output

That code is handy, and it can be kept as a code snippet so it can easily be used to see what we are working with.

RUNNING R CODE IN PROC CAS WITH GATEWAY ACTIONS

First up, we will call some R code that lives in an external file, using PROC CAS. The code is just framework to run a specified model on data specified by the macro calls. Figure 3 shows how to call the CAS Gateway action. It is important to note that all data must be in CAS libraries; the gateway action cannot point to SAS 9 libraries on disk.

```
1 %LET caslib = PUBLIC;
2 %LET castbl = HEART;
3 %let modelCode = glm(as.factor(Status) ~ AgeAtStart + Systolic + Diastolic + Weight + Cholesterol + Sex, data=df,fami
4
5 proc cas;
6     filename rcode '/nfsshare/sashls2/home/jimbox/model.r';
7     code=readfile('rcode');
8     action gateway.runlang /
9         args={caslib="&caslib", castbl = "&castbl", mod="&modelCode"},
10         code=code,
11         lang="R",
12         log_level="INFO",
13         single=true;
14 run;
15
16 quit;
```

Figure 3: Calling R from a remote program via CAS gateway action

Line 9 is how we pass values to the R program. Line 11 identifies what type of code is being run, and line 13 is where we are specifying that we are running this single threaded. We'll revisit that later, but for smaller datasets and less complicated analytics, this is fine and works just like submitting this directly via PROC IML. There are some configuration steps that would need to be done by your admin to use something other than single here, so we will not worry about that for now. The output of the model will show up in the log file, as it would with PROC IML (Figure 4).

```

NOTE: Call:
NOTE: glm(formula = as.factor(Status) ~ AgeAtStart + Systolic + Diastolic +
NOTE:       Weight + Cholesterol + Sex, family = binomial, data = df)
NOTE: Coefficients:
NOTE:             Estimate Std. Error z value Pr(>|z|)
NOTE: (Intercept) -8.6616892  0.3393746 -25.523  < 2e-16 ***
NOTE: AgeAtStart   0.1068458  0.0044927  23.782  < 2e-16 ***
NOTE: Systolic     0.0151530  0.0024647   6.148 7.85e-10 ***
NOTE: Diastolic    0.0065554  0.0043500   1.507  0.132
NOTE: Weight       -0.0010776  0.0013689  -0.787  0.431
NOTE: Cholesterol  0.0019217  0.0007645   2.514  0.012 *
NOTE: SexMale      0.9025547  0.0764243  11.810  < 2e-16 ***
NOTE: ---
NOTE: Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
NOTE: (Dispersion parameter for binomial family taken to be 1)
NOTE: Null deviance: 6708.0 on 5050 degrees of freedom
NOTE: Residual deviance: 5368.7 on 5044 degrees of freedom
NOTE: (158 observations deleted due to missingness)
NOTE: AIC: 5382.7
NOTE: Number of Fisher Scoring iterations: 4
94

```

Figure 4: Model Output is written to the SAS Log.

The approach of having the R code live in a separate file is fine with long R programs, but during development or with smaller batches of code, it is helpful to have the R code right in the PROC CAS statement. Figure 5 shows how to accomplish this.

```

24  PROC CAS;
25  externalsource Rprog1;
26
27  df <- read_table(list(name= "HEART",caslib = "PUBLIC"))
28
29  model <- glm(as.factor(Status) ~ AgeAtStart + Systolic + Diastolic + Weight + Cholesterol + Sex,
30             data=df,
31             family = binomial)
32
33  print(summary(model))
34  mdf = as.data.frame(summary(model)$coefficients)
35  mdf$Vars = rownames(mdf)
36
37  mdf <- mdf[, c("Vars", names(mdf)[-ncol(mdf)])]
38  mdf
39
40  write_table(mdf, list(name = 'Heart_Model', caslib = 'casuser', replace = TRUE))
41
42  endexternalsource;
43
44  action gateway.runLang /
45
46      code=Rprog1,
47      lang="R",
48      log_level="INFO",
49      single=true;
50  run;
51
52  PROC PRINT data = casuser.heart_model; run;
53

```

Figure 5: Including the R code in the PROC CAS call.

Lines 25 and 42 serve as the wrappers to the code block. In line 25 we had to give the program a name, which is then referenced back in line 46 with the code= setting. Also here, we added in a step in line 40 to write the model output to a table, so we could print it out with line 52 instead of having to examine the log file to see the results (Figure 6).

Obs	Vars	Estimate	Std. Error	z value	Pr(> z)
1	(Intercept)	-8.661689167	0.3393746156	-25.5225016	1.10932E-143
2	AgeAtStart	0.1068458465	0.004492739	23.781895035	5.14176E-125
3	Systolic	0.015152978	0.0024647219	6.1479464042	7.849255E-10
4	Diastolic	0.0065554329	0.0043499851	1.5070012315	0.1318103415
5	Weight	-0.001077557	0.001368858	-0.787194096	0.4311682528
6	Cholesterol	0.001921728	0.0007645438	2.513561558	0.0119518917
7	SexMale	0.9025547243	0.0764243497	11.809779576	3.47471E-32

Figure 6: Model results saved to a CAS data table and printed.

RUNNING R CODE IN PROC GATEWAY

PROC CAS is a very general way of interacting with CAS actions and can be a little hard to read. Fortunately, we have Viya PROCs that make it easier to work with these actions, and PROC GATEWAY is set up to help us do just that. Figure 7 shows how we would submit that code in a way that is easier to follow.

```

57 ❷ PROC GATEWAY single lang = R;
58      submit;
59
60      df <- read_table(list(name= "HEART",caslib = "PUBLIC"))
61
62      model <- glm(as.factor(Status) ~ AgeAtStart + Systolic + Diastolic + Weight + Cholesterol + Sex,
63                  data=df,
64                  family = binomial)
65
66
67      mdf = as.data.frame(summary(model)$coefficients)
68      mdf$Vars = rownames(mdf)
69
70      mdf <- mdf[, c("Vars", names(mdf)[-ncol(mdf)])]
71
72      write_table(mdf, list(name = 'Heart_Model', caslib = 'casuser', replace = TRUE))
73
74      endsubmit;
75
76      run;
77

```

Figure 7: Submitting R Code via PROC GATEWAY

The language used and threading type is right up in the PROC invocation on line 57. Like PROC IML and PROC PYTHON, the code is encased in a submit block (lines 58 and 74). This is a very clean way of submitting R code to the CAS server. As with PROC CAS, all data must be read in from and written out to CAS datasets to execute properly.

EXECUTING MULTI-THREADED R CODE

CONFIGURATION CHANGES TO MOVE FROM SINGLE TO MULTI-THREAD

The benefits of using CAS to multi thread your open-source programs is that the modifications to move from single threading to multi-threading are relatively simple. However, there are some mental hurdles to get over to ensure that the code executes as expected. This is because we are moving from linear code execution to the execution of the same code, at the same time, across multiple threads.

There are two primary configurations for moving code from running in single thread to multiple threads; the change of SINGLE to NTHREADS and updating the dataset to have groupby and groupbyMode options set. Both changes can be seen below in Figures 8, 9, and 10, for both PROC GATEWAY and the CAS GATEWAY action.

The NTHREADS configuration sets how many threads are created per node (Figure 8). So for the environment we ran this code in, there were 4 available nodes, and we set there to be 2 threads per node, so we had 8 total threads.

`PROC GATEWAY single lang = R;` → `PROC GATEWAY NTHREADS=2 lang=R;`

Figure 8: Threading Configuration Change for Single to Multi-Threading for PROC GATEWAY

`action gateway.runLang /`
`code=Rprog1,`
`lang="R",`
`log_level="INFO",`
`single=true;`

→

`action gateway.runLang /`
`code=Rprog1,`
`lang="R",`
`log_level="INFO",`
`NTHREADS=2;`

Figure 9: Threading Configuration Change for Single to Multi-Threading for GATEWAY Action

The addition of the groupby and groupbyMode options (Figure 10) sets what data is going to be allocated to each thread. If no groupby variable is set then the data will still run in a single thread, regardless of if NTHREADS is set. When the groupby is set the data will only be distributed across as many threads as there are groups. In the example used in Figure 10, there are only two sex groups (Male and Female) in the Heart dataset, therefore there will only be two threads that will run with data; the rest of the available threads will run without data.

`df <- read_table(list(name= "HEART",caslib = "PUBLIC"))`

↓

`heart <- read_table(list(caslib= 'public', name= 'heart', groupby='Sex', groupbyMode = 'redistribute'))`

Figure 10: PROC GATEWAY Data Configuration Change for Single to Multi-Threading

CODE CHANGES WHEN MOVING FROM SINGLE TO MULTI-THREAD

While the code Figure 8 could be run with the configuration changes for multi-threading there would be errors produced in the log and so we have had to make a few modifications when running it across threads. As noted in the previous section, when multi-threading there is a chance that there will be threads without data. By adding the if statement, seen in Figure 11 on line 64, these threads do not run code if the data passed to them is missing. The next change we made to the code was to deal with the fact that since the code is running concurrently, the variable names need to be distinct or else we run the risk of overwriting the data from one thread to another. A solution for this can be seen in Figure 11 on lines 72 and 78. There we have added on a name change following the execution of the GLM model; the new name needed to be something dynamic and in this case we used the groupby variable names, Male and Female.

```

57 ① PROC GATEWAY NTHREADS=2 lang=R;
58 submit;
59
60     df <- read_table(list(caslib= 'public', name= 'heart', groupby='Sex', groupbyMode = 'redistribute
61
62     print(gw$num_threads)
63
64     if(nrow(df)>0){
65
66         cat("Thread", gw$thread_id)
67
68         model <- glm(as.factor(Status) ~ AgeAtStart + Systolic + Diastolic + Weight + Cholesterol,
69                     data=df,
70                     family = binomial)
71
72         assign(paste("model_",df$Sex[1], sep = ""),model)
73
74         mdf = as.data.frame(summary(assign(paste("model_",df$Sex[1], sep = ""),model))$coefficients)
75         mdf$Vars = rownames(mdf)
76         mdf <- mdf[, c("Vars", names(mdf)[-ncol(mdf)])]
77
78         assign(paste("mdf_",df$Sex[1], sep = ""),mdf)
79
80         print(df$Sex[1])
81         head(assign(paste("mdf_",df$Sex[1], sep = ""),mdf), 10)
82     }
83
84 endsubmit;
85 RUN;

```

Figure 11: Submitting R Code via PROC GATEWAY using Multi-Threading

The results from the code in Figure can be seen in Figure 12 below. Here we can see that we have been able to output which thread the code ran on, utilizing the built in variable `gw$num_threads`, which groupby variable is running in that thread (Male or Female) and the resulting model details. Since this is a small dataset the speed efficiencies of multi-threading are less pronounced, but as data sizes increase their benefit would be more noticeable.

```

NOTE: Thread 4[1] "Female"
NOTE:
NOTE:      Vars      Estimate Std. Error    z value    Pr(>|z|)
NOTE: (Intercept) (Intercept) -7.5979906517 0.414528775 -18.3292237 4.837596e-75
NOTE: AgeAtStart   AgeAtStart  0.0939128686 0.006724172  13.9664594 2.497369e-44
NOTE: Systolic     Systolic    0.0112491877 0.003205295   3.5095637 4.488425e-04
NOTE: Diastolic    Diastolic    0.0077942557 0.006067017   1.2846932 1.988995e-01
NOTE: Weight       Weight     -0.0006085457 0.001855482  -0.3279717 7.429330e-01
NOTE: Cholesterol  Cholesterol  0.0015808878 0.001049167   1.5068035 1.318610e-01
NOTE: Thread 5[1] "Male"
NOTE:
NOTE:      Vars      Estimate Std. Error    z value    Pr(>|z|)
NOTE: (Intercept) (Intercept) -1.011163e+01 0.605165457 -16.7088742 1.129478e-62
NOTE: AgeAtStart   AgeAtStart  1.244938e-01 0.006471886  19.2360864 1.846336e-82
NOTE: Systolic     Systolic    2.646623e-02 0.004062030   6.5155193 7.243861e-11
NOTE: Diastolic    Diastolic    1.322024e-03 0.006372544   0.2074562 8.356536e-01
NOTE: Weight       Weight     -4.192031e-04 0.002077309  -0.2018010 8.400723e-01
NOTE: Cholesterol  Cholesterol  3.565731e-03 0.001175513   3.0333397 2.418632e-03
NOTE: WARNING: ignoring environment value of R_HOME
NOTE: WARNING: ignoring environment value of R_HOME
NOTE: [1] 8
NOTE: [1] 8
NOTE: The Cloud Analytic Services server processed the request in 3.019463 seconds.
NOTE: PROCEDURE GATEWAY used (Total process time):
NOTE:      real time      3.05 seconds
NOTE:      cpu time      0.03 seconds

```

Figure 12: Log from Submitting R Code via PROC GATEWAY using Multi-Threading

CONCLUSION

The gateway CAS action is a new and powerful way to submit Open-Source code in SAS Viya. The easiest way to interact with this action is via PROC GATEWAY, which allows you to specify the language you are using, and if you want to run the code via single-threaded or multi-threaded processing.

REFERENCES

All code for this paper is available here:

https://github.com/jwbox/phuse/blob/main/EU2025_CT07.sas

ACKNOWLEDGMENTS

The author would like to thank Eduardo Hellas, the creator of PROC GATEWAY, for his guidance on this endeavor, Samiul Haque for his contributions in general and Mary Dolegowski for her contributions on multi0threading.

RECOMMENDED READING

PROC GATEWAY documentation:

<https://documentation.sas.com/doc/da/pgmsascdc/default/proc/n1e4tu4779qvq4n1mqpgy1op58gg.htm>

CONTACT INFORMATION

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

Jim Box

SAS Institute

Jim.Box@SAS.com

Brand and product names are trademarks of their respective companies.