SAS® Viya® Platform Programming: Getting Started with Lua for CAS

2020.1 - 2025.06

This document might apply to additional versions of the software. Open this document in SAS Help Center and click on the version in the banner to see all available versions.

Requirements . . . 1

Connect and Start a Session . . . 2
- How To . . . 2
- SSL Troubleshooting . . . 3

How to Run Actions . . . 3
- Action Syntax . . . 3
- Running an Action and Viewing Results . . . 4
- Working with Result Tables . . . 5

Example: Load All Files from a Caslib . . . 7

Example: View Descriptive Statistics . . . 8

Requirements

To use Lua with SAS Cloud Analytic Services, the client machine that runs Lua must meet the following requirements:

- Use 64-bit Linux.
- Use a 64-bit version of Lua.
- Use Lua 5.2 or later.
- Install the middleclass (4.0+), csv, and ee5_base64 Lua packages.

You must also download and install the SAS Scripting Wrapper for Analytics Transfer (SWAT) package. The package is available for download from http://support.sas.com/downloads/package.htm?pid=1975. Installation information is available from a README that is included in the download.

There is additional information that is common with other programming languages and authenticating with the server.

See Also

- "Create an Authinfo File" in *Client Authentication Using an Authinfo File*
- *SAS Viya Platform: System Programming Guide*

Connect and Start a Session

How To

To enable a Lua program to work with SAS Cloud Analytic Services, you must establish a connection with the server. You need the host name and port that the CAS controller is listening on.

Example Code 1 *Connection with User Name and Password*

```
swat = require 'swat'
s = swat.CAS("cloud.example.com", 5570, "username", "password")
```

While it is possible to specify a user name and password when connecting, the best practice is to use an authinfo file to supply credentials. Using an authinfo file avoids including credentials in Lua programs.

Example Code 2 *Connection with an Authinfo File*

```
swat = require 'swat'
s = swat.CAS("cloud.example.com", 5570)
```

- SWAT is the name for the package that is used for connecting to the server.
- If a server is listening on the host name and port that are specified, and you authenticate, then the SWAT package makes a connection to the server. The package starts a session on the same hosts as the server, and returns the connection object. As a documentation convention, the variable "s" is used to represent your CAS session.

- The package attempts to locate an authinfo file in ~/.authinfo. You can override the path by specifying:

```
s = swat.CAS("cloud.example.com", 5570, {authinfo="path/.authinfo"})
```

- If you do not specify user name and password, and do not have an authinfo file, Kerberos authentication is attempted.

SSL Troubleshooting

```
ERROR: The TCP/IP negClientSSL support routine failed with status 807ff008.
ERROR: Failed to connect to host 'cloud.example.com', port 5570.
```

- Contact the server administrator for either the trustedcerts.pem or vault-ca.crt file. Specify the path to the file in the CAS_CLIENT_SSL_CA_LIST environment variable. The trustedcerts.pem file is needed for a programming-only deployment. The vault-ca.crt file is needed for a full deployment.
- For more information about CAS and TLS, see *SAS Viya Platform Encryption: Data in Motion*.

How to Run Actions

Action Syntax

After you connect to the server and have a session, you use the session to run actions. The following code is an example of the syntax for running a simple action.

```
results, info = s:builtins_serverStatus{}
```

The parts of that syntax are as follows:

`results`
: This is a named Lua table that is used to store the results of an action.

`info`
: This is another named Lua table that is used to store metadata about the action run such as the status, printed messages, and performance information.

`s`
: This is a named variable that represents your CAS session. You can use any name that you prefer. As a documentation convention, "s" is used.

`builtins`
: This is the name of the action set that includes the action to run. Specifying the action set name is optional, but is a best practice.

4

`serverStatus{}`
: This is the name of the action to run. If an action accepts parameters, such as the name of table to analyze, then the parameters are specified within the braces.

Running an Action and Viewing Results

You can run the action, store the results in a variable, and then view the results:

```
results, info = s:builtins_serverStatus{}
print(results)
```

Output 1 *Results of the ServerStatus Action*

```
[server]

      Server Status
Node Count  Total Actions
         8              2

[nodestatus]

                          Node Status
Node Name              Role        Uptime (Sec)  Running  Stalled
cloud1.example.com  backup ctl       118.125        0        0
cloud1.example.com  worker           118.126        0        0
cloud1.example.com  worker           118.125        0        0
cloud1.example.com  worker           118.125        0        0
cloud1.example.com  worker           118.126        0        0
cloud1.example.com  worker           118.126        0        0
cloud1.example.com  worker           118.126        0        0
cloud.example.com   controller        118.16        0        0

[About]

table: 0x1a62950
```

1 The results are shown for a distributed server that uses eight machines: a controller, a backup controller, and 6 worker nodes. For a single-machine server, the results include the line for the controller node only.

The result of the builtins_serverStatus action has three keys, as shown. As with all Lua tables, you can access the results with the key:

```
print(results.server)
```

```
      Server Status
Node Count  Total Actions
         8              2
```

The metadata about the action run is also in a table:

```
for k, v in pairs(info) do
   print(k, type(v))
end
```

```
events  table
severity        number
statusCode      number
performance     table
updateflags     table
messages        table
```

You can check the info.severity value to determine whether an action ran without error. Zero indicates success. For more information, see "Severity, Status, and Reason Codes" in *SAS Viya Platform: System Programming Guide*.

Working with Result Tables

Many actions produce tables as all or part of the results.

Result tables support the # syntax that is part of Lua to determine the number of rows in a table:

Example Code 3 *Determine the Number of Rows in a Result Table*

```
nodeTable = s:serverStatus{}.nodestatus
```

```
NOTE: Grid node action status report: 8 nodes, 31 total actions executed.
```

```
print(type(nodeTable))
```

```
table
```

```
print(#nodeTable)
```

```
8
```

You can use the .columns attribute to determine the number of columns and the column names in a result table:

```
print(#nodeTable.columns)
```

```
5
```

```
for k,v in pairs(nodeTable.columns) do
   print(k)
end
```

```
name
role
uptime
running
stalled
```

The column names are the same as the column names that are shown in Output 1 on page 4.

You can specify the names of columns to access from a result table:

Example Code 4 *Accessing Selected Columns in a Result Table*

```
print(nodeTable{'name', 'role'})
```

```
           Node Status
Node Name              Role
cloud1.example.com  backup ctl
cloud2.example.com  worker
cloud3.example.com  worker
cloud4.example.com  worker
cloud5.example.com  worker
cloud6.example.com  worker
cloud7.example.com  worker
cloud.example.com   controller
```

Similarly, you can access a range of columns by enclosing the start-column and the end-column in double braces:

```
print(nodeTable{{'role', 'running'}})
```

```
           Node Status
Role        Uptime (Sec)  Running
backup ctl       407.089        0
worker           407.089        0
worker           407.089        0
worker           407.089        0
worker           407.089        0
worker           407.089        0
worker           407.089        0
controller       407.123        0
```

You can access rows by index as well:

Example Code 5 *For the First and Eighth Row, Access the Name and Role Columns*

```
print(nodeTable{1, 8}{'name', 'role'})
```

```
           Node Status
Node Name              Role
rdcgrd075.unx.sas.com  backup ctl
rdcgrd066.unx.sas.com  controller
```

Example Code 6 *For the First and Eighth Row, Access Name Column and the Columns between Role and Running*

```
print(nodeTable{1,8}{'name', {'role', 'running'}})
```

```
                    Node Status
Node Name              Role        Uptime (Sec)  Running
cloud1.example.com  backup ctl       407.089        0
cloud.example.com   controller       407.123        0
```

Example: Load All Files from a Caslib

This example shows how to create a caslib, use the table_fileInfo action to get a list of the SASHDAT files in the caslib, and then load the SASHDAT files in the list into memory. Here is the example code.

```
swat = require 'swat'
s = swat.CAS("cloud.example.com", 5570)

s:table_addCaslib{                                    -- 1
   name="casdata",
   dataSource={srcType="path"},
   path="path-to-server-side-files"
}

files = s:table_fileInfo{path="%.sashdat"}             --2

print("\nList of SASHDAT files in caslib Casdata:")   --3
print(files.FileInfo.columns['Name'])


print("\nLoad the tables in caslib Casdata.")        -- 4
for i=1,#files.FileInfo do
   s:table_loadTable{path=files.FileInfo[i]['Name'],
   promote=false}                                     -- 5
end

print("\n")                                           -- 6
tables = s:table_tableInfo{}
print(tables.TableInfo{'Name', 'Rows', 'Columns','Label'})
```

1 After the Casdata caslib is added, it becomes the active caslib. Unless another caslib name is specified in a subsequent action for this session, the active caslib is used.

2 The % filename wildcard is used to list all files that have a .sashdat suffix. The results of the table_fileInfo action are stored in the Files variable.

3 The results table from the table_fileInfo action is named FileInfo. The columns attribute is used to display the values from the Name column only.

4 Iterate over them and run the table_loadTable action on each file.

5 Use the promote parameter to load the table as a global-scope table. This enables other sessions and other users to access the in-memory tables, if they have permission to access data in the caslib.

6 Use the table_tableInfo action to list the in-memory tables.

Here is an example of the result.

```
List of SASHDAT files in caslib Casdata:
cars.sashdat
class.sashdat
iris.sashdat
prdsale.sashdat
Name: Name, dtype: varchar

Load the tables in caslib Casdata.
NOTE: Cloud Analytic Services made the file cars.sashdat available as table CARS in caslib casdata.
NOTE: Cloud Analytic Services made the file class.sashdat available as table CLASS in caslib
casdata.
NOTE: Cloud Analytic Services made the file iris.sashdat available as table IRIS in caslib casdata.
NOTE: Cloud Analytic Services made the file prdsale.sashdat available as table PRDSALE in caslib
casdata.

     Table Information for Caslib casdata
Name     Rows  Columns  Label
CARS      428       16  4-cylinder car data
CLASS      19        6  Student data
IRIS      150        6  Iris data
PRDSALE  1440       11  Regional product sales
```

Example: View Descriptive Statistics

This example shows how to upload a CSV file to a table and how to summarize the table data. The table data is stored in file iris.csv. You can download this file from the following URL:

https://support.sas.com/documentation/onlinedoc/viya/exampledatasets/iris.csv

Here is the example code.

```
swat = require 'swat'
swat.setOption('display.width', 120)
s = swat.CAS("cloud.example.com", 5570)

                                                                    --1
result = s:upload{"path-to-data/iris.csv",
   casout={name="iris", caslib="casuser", replace=true}}

irisTbl = {}                                                        --2
irisTbl.name = result.tableName
irisTbl.caslib = result.caslib
irisTbl.groupBy = {"species"}

stats = {"min", "max", "n", "nmiss", "mean", "std", "stderr"}

results, info = s:simple_summary{table=irisTbl, subset=stats}       --3

bgi = results.ByGroupInfo                                           --4

print(bgi)
print()

-- so we don't print this again
```

```
results.ByGroupInfo = nil

for k,v in pairs(results) do
   print(k, v)
   print()
end
```

1 Perform a client-side load of the Iris.csv file from a path that Lua can access. The upload method transfers the CSV file from Lua to the server, and then the server loads the data into memory.

2 A variable with the name IrisTbl is created to represent the in-memory table. The groupBy value, Species, is set as a key in the variable.

3 The results from the simple_summary action include the descriptive statistics that are listed in the Stats variable.

4 When you perform BY-group processing, the results include one table that is named ByGroupInfo and a table for each BY-group. The ByGroupInfo table is printed first and then a loop is used to print the remaining tables.

Here is the result.

```
           ByGroupInfo
Species     Species_f   _key_
setosa      setosa      setosa
versicolor  versicolor  versicolor
virginica   virginica   virginica

ByGroup3.Summary                                  Descriptive Statistics for IRIS
Species    Analysis Variable  Minimum  Maximum   N  N Miss    Mean  Std Dev  Std Error
virginica  Sepal Length        4.9000   7.9000  50       0  6.5880   0.6359    0.08993
virginica  Sepal Width         2.2000   3.8000  50       0  2.9740   0.3225    0.04561
virginica  Petal Length        4.5000   6.9000  50       0  5.5520   0.5519    0.07805
virginica  Petal Width         1.4000   2.5000  50       0  2.0260   0.2747    0.03884
virginica  Index               101.00   150.00  50       0  125.50  14.5774     2.0616

ByGroup2.Summary                                    Descriptive Statistics for IRIS
Species     Analysis Variable  Minimum  Maximum   N  N Miss     Mean  Std Dev  Std Error
versicolor  Sepal Length        4.9000   7.0000  50       0   5.9360   0.5162    0.07300
versicolor  Sepal Width         2.0000   3.4000  50       0   2.7700   0.3138    0.04438
versicolor  Petal Length        3.0000   5.1000  50       0   4.2600   0.4699    0.06646
versicolor  Petal Width         1.0000   1.8000  50       0   1.3260   0.1978    0.02797
versicolor  Index              51.0000   100.00  50       0  75.5000  14.5774     2.0616

ByGroup1.Summary                                  Descriptive Statistics for IRIS
Species  Analysis Variable  Minimum  Maximum   N  N Miss    Mean  Std Dev  Std Error
setosa   Sepal Length        4.3000   5.8000  50       0  5.0060   0.3525    0.04985
setosa   Sepal Width         2.3000   4.4000  50       0  3.4180   0.3810    0.05388
setosa   Petal Length        1.0000   1.9000  50       0  1.4640   0.1735    0.02454
setosa   Petal Width         0.1000   0.6000  50       0  0.2440   0.1072    0.01516
setosa   Index               1.0000  50.0000  50       0 25.5000  14.5774     2.0616
```