



Getting Started with SAS[®] Viya[®] 3.5 for Lua

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: 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 {authinfo="/path/to/file"}.
- 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 [Encryption in SAS Viya: 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.

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

Output 1 Results of the ServerStatus Action

```
[server]

      Server Status
Node Count  Total Actions
           8             4

[About]

table: 0x1638b80

[nodestatus]

      Node Status
Node Name      Role      Uptime (Sec)  Running  Stalled
cloud1.example.com  worker      1076.941      0        0
cloud2.example.com  worker      1076.942      0        0
cloud3.example.com  worker      1076.932      0        0
cloud4.example.com  worker      1076.941      0        0
cloud5.example.com  worker      1076.942      0        0
cloud6.example.com  worker      1076.944      0        0
cloud7.example.com  worker      1076.942      0        0
cloud.example.com   controller  1076.928      0        0
```

- 1 The results are shown for a distributed server that uses eight machines. 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             4
```

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

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

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

You can check the info.severity value to determine whether an action ran without error. Zero indicates success. For more information, see [“Severity and Reason Codes” in SAS Viya: 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	worker
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
worker	5189.845	0
worker	5189.863	0
worker	5189.858	0
worker	5189.84	0
worker	5189.565	0
worker	5189.829	0
worker	5189.857	0
controller	5189.844	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
cloud1.example.com	worker
cloud.example.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	worker	5189.845	0
cloud.example.com	controller	5189.844	0

Example: Load All Files from a Caslib

Example Code 7 Add a Caslib and Store the Results of the FileInfo Action

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

-- add a caslib that uses a file system path -- 1
s:table_addCaslib{
  name="casdata",
  dataSource={srcType="path"},
  path="/path/to/server-side-files"
}

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

-- print only the file names
print(files.FileInfo.columns['Name']) -- 3
```

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

```
historicalcpi.sashdat
car_prices.sashdat
cars.sashdat
iris.sashdat
Name: Name, dtype: varchar
```

Now that you have the filenames available, you can iterate over them and run the table_loadTable action.

Example Code 8 Iterate Over the Results and Load the Files

```
for i=1,#files.FileInfo do
  s:table_loadTable{path=files.FileInfo[i]['Name'], promote=true} -- 1
end
```

- 1 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.

NOTE: Cloud Analytic Services made the file historicalcpi.sashdat available as table HISTORICALCPI in caslib casdata.
 NOTE: Cloud Analytic Services made the file car_prices.sashdat available as table CAR_PRICES 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 iris.sashdat available as table IRIS in caslib casdata.

Finally, you can use the table_tableInfo action to list the in-memory tables.

```
tables = s:table_tableInfo{}
print(tables.TableInfo{'Name', 'Label', 'Rows'})
```

Table Information for Caslib casdata		
Name	Label	Rows
HISTORICALCPI	Historical CPI data, 1974 through 2014; updated 3/3/2015	242
CAR_PRICES		76
CARS	2004 Car Data	428
IRIS	Fisher's Iris Data (1936)	150

Example: View Descriptive Statistics

Example Code 9 Add a Caslib and Store the Results of the FileInfo Action

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

8

```
result = s:upload{"/path/to/iris.csv", casout={name="iris", replace=true}}

irisTbl = {} -- 2
irisTbl.name = result.tableName
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.

```

ByGroupInfo
Species    Species_f  _key_
Setosa     Setosa     Setosa
Versicolor Versicolor  Versicolor
Virginica  Virginica  Virginica

ByGroup1.Summary                               Descriptive Statistics for IRIS
Species Analysis Variable      Min      Max  N  Number Missing      Mean  Std Dev.  Std Error
Setosa  SepalLength              43.0000  58.0000  50      0  50.0600    3.5249    0.4985
Setosa  SepalWidth                23.0000  44.0000  50      0  34.2800    3.7906    0.5361
Setosa  PetalLength                10.0000  19.0000  50      0  14.6200    1.7366    0.2456
Setosa  PetalWidth                   1.0000   6.0000  50      0   2.4600    1.0539    0.1490

ByGroup2.Summary                               Descriptive Statistics for IRIS
Species Analysis Variable      Min      Max  N  Number Missing      Mean  Std Dev.  Std Error
Versicolor SepalLength              49.0000  70.0000  50      0  59.3600    5.1617    0.7300
Versicolor SepalWidth                20.0000  34.0000  50      0  27.7000    3.1380    0.4438
Versicolor PetalLength                30.0000  51.0000  50      0  42.6000    4.6991    0.6646
Versicolor PetalWidth                10.0000  18.0000  50      0  13.2600    1.9775    0.2797

ByGroup3.Summary                               Descriptive Statistics for IRIS
Species Analysis Variable      Min      Max  N  Number Missing      Mean  Std Dev.  Std Error
Virginica SepalLength              49.0000  79.0000  50      0  65.8800    6.3588    0.8993
Virginica SepalWidth                22.0000  38.0000  50      0  29.7400    3.2250    0.4561
Virginica PetalLength                45.0000  69.0000  50      0  55.5200    5.5189    0.7805
Virginica PetalWidth                14.0000  25.0000  50      0  20.2600    2.7465    0.3884

```

