SAS® GLOBAL FORUM 2020 #SASGF

Paper SAS4576-2020

Making Administration Easier by Scripting the SAS® Viya® Command-Line Interfaces

James Holman, SAS Institute Inc., Cary, NC

ABSTRACT

SAS® Viya® offers a set of rich command-line interfaces (CLIs) to directly interact with SAS Viya without using a graphical interface. This paper describes how these CLIs can be scripted and chained together to accomplish administrative tasks. Examples include backing up and then changing the logging levels for SAS® Cloud Analytic Services (CAS), auditing failed logins, updating themes across a set of SAS Viya reports, batch creating caslibs and adding access controls to them, as well as many other tasks. All of these tasks can be done remotely as well, without having to physically be on a SAS Viya system.

INTRODUCTION

SAS Viya contains various command-line interfaces (CLIs) that are available to SAS Viya administrators. The main entry point is an application called sas-admin. For SAS Viya 3.X deployments, sas-admin is installed on SAS Viya systems. sas-admin can also be downloaded to your environment and runs on Linux, Microsoft Windows, and Mac OS systems. The download method is the recommended approach for SAS Viya 4.0.1 as well. Commands executed with sas-admin are calling plug-ins and will be referred to as such.

This paper will discuss topics SAS administrators will use to improve productivity – automation of everyday tasks and building a toolbox of the various tasks scripted. Most examples shown will be in Unix shell script language. Understanding of SAS Viya authentication and authorization is needed but will be discussed.

WHAT IF?

It's Sunday night and you're just settling down to watch the NFL divisional championships. You're draped in your team's colors and you have your favorite beverage in hand. Why is the TV ringing? Wait, that's your phone and it's your manager calling. This can't be good. Your manager is excited and tells you a new project is green lit and the department doing it needs to be onboarded. You agree it sounds exciting, secretly hoping to still see the football game. But then it happens – the onboarding must happen tonight and be ready by 9am Monday morning. Thankfully, your manager does have the requirements of what's needed. You tell your manager you'll get right on this and mournfully consider abandoning football. But you know there's a chance for the second half, maybe more as you have a toolbox of scripts already in place that can do a lot of what's needed. And since the SAS Viya command-line interfaces are so robust and powerful, you sit down and script the rest. After some testing, you onboard the department and send mail to your manager. Back to the game now – oh no, there's only 7 minutes left!? Wait, there's 7 minutes left in the first quarter. Glad I had that toolbox and it has even more content now.

While the above story may never happen to you, knowing that you can handle it is one of the reasons that automation is so important.

GETTING STARTED

In order to use sas-admin in an automated fashion, the first thing to do is set up your authentication paradigm. The SAS Viya command-line interface uses the same authentication scheme as the SAS Viya web applications do. Authentication tokens need to be renewed every 12 hours by default and manually authenticating to them defeat the purpose of scripting. By default, the SAS Viya command-line interface saves its configuration to $HOME/.sas.

Consider creating a file called credentials.sh in $HOME/.sas:

```
#!/bin/sh

SAS_ADMIN_USER=demoadmin
SAS_ADMIN_PASS=demopass
```

Creating the file in the same location as the supporting files used by the SAS Viya command-line interface allows for enforcing desired permissions and protecting passwords.

Once this file is in place, it's usable in any script you write. For example, this script (auth.sh) sources the file and then authenticates to SAS Viya:

```
#!/bin/sh

# The location of sas-admin
SAS_ADMIN=/opt/sas/viya/home/bin/sas-admin

# The profile to use
export SAS_CLI_PROFILE=”$1”

# Include our credentials
source ~/.sas/credentials.sh

# Authenticate
$SAS_ADMIN auth login --u $SAS_ADMIN_USER --p $SAS_ADMIN_PASS > /dev/null

# Success?
if [ $? != 0 ]; then
    echo "Authentication failure"
else
    echo "Authentication success"
fi
```

As you can see in the above code, a “profile” is referenced and is passed in as an argument. In this context, a profile is a construct used to define and characterize the deployment you'll be working with. Profiles are created with the sas-admin profile init command and can be named. See "Create at Least One Profile" in SAS Viya 3.5 Administration for a more detailed explanation. Each named and persisted profile is contained in the file credentials.json, located in $HOME/.sas.

This is the first tool in our toolbox.

WHAT TO AUTOMATE

Everyday tasks are the best choices for automation, but so are complex tasks you may need to repeat. The challenges a SAS administrator faces are varied and numerous. Maybe a new department needs its infrastructure created or perhaps you need to whitelist 200 new mobile devices, allowing them to connect to your environment. The following sections will discuss several different situations and what can be done with them.

CAS LOGGERS

One thing an administrator must deal with is log files. They have the potential to grow out of control. Suppose your CAS server has been doing quite a lot lately and you're seeing more logs than you need. The cas plug-in to sas-admin can deal with CAS loggers, so let's see how we can back up the current logging levels and then set them to something more manageable.

The first thing we need to do is obtain the list of CAS servers in the deployment. The following examples assume that sas-admin is in the current OS path and a profile is defined and authenticated to.

This command returns a list of all the servers in a deployment:

```
sas-admin cas servers list
```

And now to get the loggers set to info for one of our servers, cas-shared-default:

```
sas-admin cas servers loggers list --server cas-shared-default --level info
```

Consider the following script:

```
#!/bin/sh

# Authenticate and define our environment
source ./auth.sh demoprofile

# Set the sas-admin options we need (json)
export SAS_OUTPUT=json

# Obtain the list of CAS servers
SERVERS=`$SAS_ADMIN cas servers list | grep \”name\”: | cut -f4 -d\”`

# Any errors? If so, exit
if [ $? != 0 ]; then
    echo "ERROR: Unable to determine the list of CAS servers"
    exit 1
fi

# Loop through the list of servers, getting the required loggers
for server in $SERVERS ; do

    # Obtain the trace loggers for the current server
    TRACELOGGERS=`$SAS_ADMIN servers loggers list --server $server --level
trace --all | grep \"name\": | cut -f4 -d\"`

    # Any errors? If so, exit
    if [ $? != 0 ]; then
        echo ERROR: Unable to obtain list of TRACE loggers
        exit 1
    fi

    # Obtain the info loggers for the current server
    INFOLOGGERS=`$SAS_ADMIN servers loggers list --server $server --level
info --all | grep \"name\": | cut -f4 -d\"`

    # Any errors? If so, exit
    if [ $? != 0 ]; then
        echo ERROR: Unable to obtain list of INFO loggers
        exit 1
    fi

    # Remove any existing default files if they exist as we're making them
with this execution
    rm -f $server-trace-cas-loggers.txt
    rm -f $server-info-cas-loggers.txt

    echo Recording all CAS loggers with a level of TRACE...

    # For each trace logger, write it to the backup file
    for logger in $TRACELOGGERS ; do
        echo $logger >> $server-trace-cas-loggers.txt
        echo $logger
    done
    echo

    echo Recording all CAS loggers with a level of INFO...

    # For each info logger, write it to the backup file
    for logger in $INFOLOGGERS ; do
        echo $logger >> $server-info-cas-loggers.txt
        echo $logger
    done
done
```

The above script does several things and contains many pieces we can add to our toolbox. Be sure to take note of the process. A plug-in command executes, and its results are stored in a variable. JSON output (as opposed to text output) allows for more targeted parsing. Being able to loop through a list of items we found is invaluable. In the code above, we basically found all the CAS servers, and then for each CAS server, found all the CAS loggers set to `trace` or `info`. See the *Appendix* for the rest of the code that resets the `trace` and `info` loggers to the warn level as well as restores them to their original settings.

RETHEMING REPORTS

SAS Viya has powerful reporting features that use cutting edge analytics to solve business problems. Reports have cosmetic attributes as well and as a SAS administrator, you may find yourself having to retheme a given department's reports with the company accepted 2020 theme. Opening each report in SAS® Visual Analytics would take some time, so let's look at how we would script this issue.

The `reports` plug-in offers the ability to find reports, learn details about them, delete them, retheme them, etc. Based on our experience with CAS servers above, we need to find all the reports that meet a certain criterion and then retheme them.

Consider the following (again we'll use our authentication script):

```
#!/bin/sh

# Authenticate and define our environment
source ./auth.sh demoprofile

# Set the sas-admin options we need (json)
export SAS_OUTPUT=json

# Obtain the list of "Demo Dept" reports
```

```
REPORTS=`$SAS_ADMIN reports list --name-starts-with "Demo Dept" | grep
\"id\" | cut -d\" -f4`

# Any errors? If so, exit
if [ $? != 0 ]; then
    echo ERROR: Unable to find any reports beginning with \"Demo Dept\"
    exit 1
fi

# Loop through the reports we found, setting each one's theme to 2020Theme
for report in $REPORTS ; do
    $SAS_ADMIN reports themes update --force --report-id $report --theme-id
2020Theme

    # Any errors?
    if [ $? != 0 ]; then
        echo Unable to set the theme for the report with ID $report
        echo
    fi
done
```

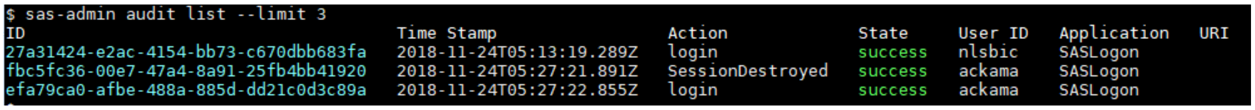
As we did with CAS servers, we loop through the applicable reports and in this case, retheme them. Note though, a filter was used (`--name-starts-with`) in locating the reports we wanted. Most of the SAS Viya command-line interfaces offer a rich filtering mechanism. Being able to search for specific patterns and conventions is an important tool to have.

AUDITING FAILED LOGONS

I was talking to one of our consultants the other day about a customer's request to audit login activity in SAS Viya. Out of the box, several administrative reports detailing system stability and activity are available in SAS Viya. There is an audit report as well but in the current releases, login information is not included.

Here are some actual commands using the audit plug-in and its subcommands.

The `list` subcommand will show a series of records and has many filters to help refine results. Display 1 is a typical audit record listing.



```
$ sas-admin audit list --limit 3
ID                                    Time Stamp                Action            State     User ID   Application   URI
27a31424-e2ac-4154-bb73-c670dbb683fa  2018-11-24T05:13:19.289Z  login             success   nlsbic    SASLogon
fbc5fc36-00e7-47a4-8a91-25fb4bb41920  2018-11-24T05:27:21.891Z  SessionDestroyed  success   ackama    SASLogon
efa79ca0-afbe-488a-885d-dd21c0d3c89a  2018-11-24T05:27:22.855Z  login             success   ackama    SASLogon
```

Display 1. A Simple Listing of Audit Records

Most of the column headers can be used as filters. Since the consultant I spoke to was interested in login records, I gave him the following command:

```
sas-admin audit list --action login
```

That returns a list of audit records (50 by default) where the action was "login". In the case of more precise tracking, additional filters can be added.

For example:

```
sas-admin audit list --action login --state failure
```

This returns a set of audit records where logins failed.

```
sas-admin audit list --action SessionDestroyed --state success
```

This command is looking for audit records where login sessions were destroyed, e.g. a user logged out.

Of course, these can be filtered by user.

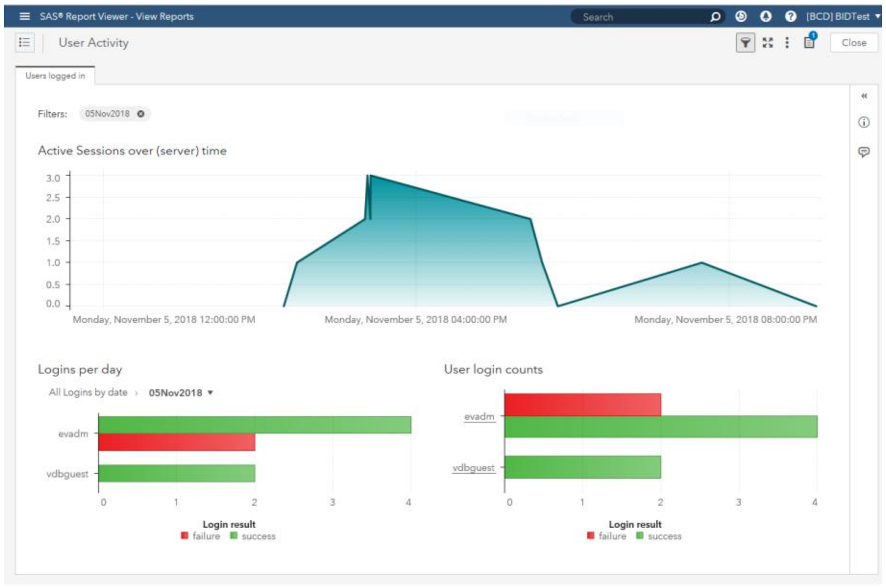
```
sas-admin audit list --action login --state success --user-id james
```

This command shows audit records where I logged in.

Once you’ve gotten the filters in place for the desired results, the command’s output can be directed to a CSV file for further processing. The --csv option to the list subcommand takes the fully qualified path and filename as the location to write the file. For example:

```
sas-admin audit list --action login --csv /tmp/records.csv
```

At this point, I took the CSV file and imported it into CAS using SAS Data Explorer for SAS Viya. Then using SAS Visual Analytics on SAS Viya, I created myself a report detailing login information. Display 2 shows the report.



Display 2. Reporting Audit Failures

ADDING TRANSLATIONS TO SAS VIYA REPORTS

As seen earlier, we were able to find a specific set of reports and retheme them. The next thing we’re asked to do is to support a group of translators that want to translate all strings in the report into French, Spanish, German, Japanese, and Korean. With the reports plug-in, it’s easy to automate the process of extracting the English strings as well as importing the newly created translations.

Consider the following script:

```
#!/bin/bash

# Authenticate and define our environment
source ./auth.sh demoprofile

# Set the sas-admin options we need (json)
export SAS_OUTPUT=json

# Obtain the list of “Demo Dept” reports
REPORTS=`$SAS_ADMIN reports list --name-starts-with "Demo Dept" | grep
\"id\" | cut -d\" -f4`

# Any errors? If so, exit
if [ $? != 0 ]; then
```

```
    echo ERROR: Unable to find any reports beginning with \"Demo Dept\"
    exit 1
fi

# Define the languages we need
LANGUAGES="fr es de ja ko"

# Now loop over the reports and for each language, extract the strings
for report in $REPORTS; do
    for language in $LANGUAGES; do
        $SAS_ADMIN reports translations export --report-id $report --
report-locale $language --output-location .
    done
done
```

The idea here is to extract all the languages you're interested in initially even though the only language that likely is in the report is the one it was written in. By extracting them all now, what's produced is easy for the translators to understand. When the script is complete, the following files now exist in the directory from which the script was executed:

```
Demo Dept - Report 1_de.reports       Demo Dept - Report 3_ja.reports
Demo Dept - Report 1_es.reports       Demo Dept - Report 3_ko.reports
Demo Dept - Report 1_fr.reports       Demo Dept - Report 4_de.reports
Demo Dept - Report 1_ja.reports       Demo Dept - Report 4_es.reports
Demo Dept - Report 1_ko.reports       Demo Dept - Report 4_fr.reports
Demo Dept - Report 2_de.reports       Demo Dept - Report 4_ja.reports
Demo Dept - Report 2_es.reports       Demo Dept - Report 4_ko.reports
Demo Dept - Report 2_fr.reports       Demo Dept - Report 5_de.reports
Demo Dept - Report 2_ja.reports       Demo Dept - Report 5_es.reports
Demo Dept - Report 2_ko.reports       Demo Dept - Report 5_fr.reports
Demo Dept - Report 3_de.reports       Demo Dept - Report 5_ja.reports
Demo Dept - Report 3_es.reports       Demo Dept - Report 5_ko.reports
Demo Dept - Report 3_fr.reports
```

As you can see, each report now has five language files, denoted by the 2-letter country code. These files can be handed off to the translators now through whatever process is desired.

When translation is complete, the files can be returned to you the administrator for importing. Now consider the following script:

```
#!/bin/bash

# Authenticate and define our environment
source ./auth.sh demoprofile

# Define the languages we need
LANGUAGES="fr es de ja ko"

# Now loop over the translations and import each language for the 5 reports
for language in $LANGUAGES; do
    for i in 1 2 3 4 5 ; do
```

```
        $SAS_ADMIN reports translations import --source-file "Demo Dept --
Report ${i}_$language.reports"
    done
done
```

For each language we have translations for, we loop over the "Demo Dept" reports and insert each language into them. This approach can be modified to whatever paradigm is appropriate, but the tools the `reports` plug-in offers are great additions to our toolbox.

CLONING CASLIBS

Caslibs are integral for SAS Viya users and something an administrator must deal with. From disk space allocation to database schema settings, there are things that will need to be done correctly. This kind of work sometimes is best done via a graphical user interface, such as SAS Data Explorer. There, a user can specify whatever settings are appropriate. But once that's done, if there is a need to continue creating these kinds of caslibs, it sounds like a job for our toolbox.

Assume for this example that an Oracle caslib was created named `oralib` in the CAS server `cas-shared-default`. The following command will extract its definition in a JSON format:

```
  sas-admin --output json cas caslibs show-info --server cas-shared-default -
-caslib oralib > oracletmpl.json
```

The JSON definition of the caslib is now in the file `oracletmpl.json`. It has all the settings specified during the creation of the caslib in SAS Data Explorer. But what can be done with it?

Suppose you needed to create another Oracle caslib, `oralib2`, again in `cas-shared-default`. All that's different between it and `oralib` are its name and schema. Using your newly created JSON file, you can do this creation from the command-line as well:

```
  sas-admin cas caslibs create oracle --name oralib2 --schema ORALIB2S --
source-file oracletmpl.json
```

All settings from the JSON file are maintained save the new name and schema.

This approach lends itself to quickly creating caslibs in batch. The cas plug-in also offers a command that generates sample templates:

```
  sas-admin cas --generate-cas-samples --output-location <path>
```

At this point, we can create caslibs in batch whenever we'd like. But what about the permissions on the caslibs? Can they be cloned as well? Yes, they can.

Consider the following command:

```
  sas-admin cas --output json caslibs list-controls --server cas-shared-
default --caslib oralib > oraclectrlstmpl.json
```

This command creates a JSON file containing all direct access controls set on the caslib `oralib`. Assuming that those controls are what you want any Oracle caslib to have, you could use this command after creating `oralib2`, as seen above, to achieve this:

```
  sas-admin replace-controls --server cas-shared-default --caslib oralib2 --
superuser --force --source-file oraclectrlstmpl.json
```

The `--force` option causes the command to execute without any prompting and the `--superuser` option runs the command as a CAS super user.

At this point, our toolbox has grown a good bit.

THE FOOTBALL PROJECT

As mentioned in the beginning, phone calls can pull one away from various things. Although the story turned out all right with beverages consumed and football watched, let's look at what onboarding a new department could mean.

The following is probably a subset of what really would be needed.

- Create 4 new groups: `NewDeptUsers`, `NewDeptMgrs`, `NewDeptAdmins`, `NewDeptDataAdmins`
  - The `identities` plug-in can help here.
- Add the `NewDeptAdmins` group to the CAS admin users list.
  - The `cas` plug-in can help here.
- Create a folder for `NewDeptDataAdmins` to store SAS Viya reports in.
  - The `folders` plug-in can help here.
- Grant the `NewDeptDataAdmins` relevant permissions to their folder.
  - The `authorization` plug-in can help here.
- Whitelist the mobile devices for the 4 newly hired managers.
  - The `devices` plug-in can help here.
- Create 4 new path-based caslibs: `NewDeptUsers`, `NewDeptMgrs`, `NewDeptAdmins`, `NewDeptDataAdmins`
  - The `cas` plug-in can help here.
- Apply access controls to each of the 4 new caslibs.
  - The `cas` plug-in can help here.

See the *Appendix* for a sample script that does the above tasks.

CONCLUSION

The SAS Viya command-line interfaces are numerous and robust. They are designed to allow an administrator to remotely manage their environments, but their very nature lends themselves to scripting for automation. The very first computer science teacher I had talked about programming as nothing but a collection of tools in your toolbox. I think it's an accurate and fair metaphor applicable to the use of command-line interfaces. Whether you are batch updating themes in SAS Viya reports or creating 20 caslibs, the atomic and scriptable nature of the commands should make life easier. I hope this paper is the start of your own SAS Viya command-line interfaces toolbox and that it helps make your life easier as a SAS administrator.

REFERENCES

SAS Institute Inc. 2019. "Create at Least One Profile." In *SAS Viya 3.5 Administration*. Cary, NC: SAS Institute Inc. Available https://go.documentation.sas.com/?cdcId=calcdc&cdcVersion=3.5&docsetId=calcli&docsetTarget=n1e2dehluji7jon1gk69yggc6i28.htm&locale=en#p17rei99cguhqdn13m8vgg81nlax (accessed February 10, 2020).

RECOMMENDED READING

- *SAS® Viya® 3.5 Administration*
- *SAS® Viya® 3.5 Administration: CAS Authorization*
- *SAS® Viya® 3.5 Administration: Using the Command-Line Interfaces*

APPENDIX

The following scripts are more complete versions of a couple of the techniques described in the main paper. They all make use of the auth.sh paradigm described in the *Getting Started* section.

CAS LOGGERS REDUX

Three scripts will be presented – one that makes a record of all CAS loggers with either info or trace level logging, one that changes all CAS loggers with either info or trace level logging to warn level logging, and one that restores the affected CAS loggers to their original info and trace levels.

First up, the backup script:

```
#!/bin/bash

# Define our commands and profile for authentication; we don't need to see
# successes, just errors
source ./auth.sh demoprofile > /dev/null

# For this script, we want JSON output
export SAS_OUTPUT=json

# Obtain the list of servers...
SERVERS=`$SAS_ADMIN cas servers list | grep \"name\": | cut -f4 -d\"`

# Any errors, if so exit
if [ $? != 0 ]; then
   tput setaf 1
   echo "ERROR: Unable to get list of CAS servers"
   tput sgr0
   exit 1
fi

# Loop through each server, getting the required loggers
for server in $SERVERS ; do

    # Obtain the loggers with a level of trace in this server
    TRACELOGGERS=`$SAS_ADMIN cas servers loggers list --server $server --
level trace --all | grep \"name\": | cut -f4 -d\"`

    # Any errors, if so exit
    if [ $? != 0 ]; then
       tput setaf 1
       echo "ERROR: Unable to get list of CAS loggers with a level of
TRACE"
       tput sgr0
       exit 1
    fi

    # Obtain the loggers with a level of info in this server
    INFOLOGGERS=`$SAS_ADMIN cas servers loggers list --server $server --
level info --all | grep \"name\": | cut -f4 -d\"`

    # Any errors, if so exit
    if [ $? != 0 ]; then
       tput setaf 1
       echo "ERROR: Unable to get list of CAS loggers with a level of INFO"
       tput sgr0
       exit 1
    fi
    # Remove any existing the default files if they exist as we're making
them with this execution
    rm -f $server-trace-cas-loggers.txt
    rm -f $server-info-cas-loggers.txt

    tput setaf 4
    echo "Recording all CAS loggers with a level of TRACE..."
    tput sgr0

    # For each trace logger, write it to the backup file
    for logger in $TRACELOGGERS ; do
        echo $logger >> $server-trace-cas-loggers.txt
        tput setaf 2
        echo $logger
        tput sgr0
    done

    echo

    tput setaf 4
    echo "Recording all CAS loggers with a level of INFO..."
    tput sgr0

    # For each info logger, write it to the backup file
    for logger in $INFOLOGGERS ; do
        echo $logger >> $server-info-cas-loggers.txt
        tput setaf 2
        echo $logger
        tput sgr0
    done
done
```

The following modifies the CAS loggers with levels of info or trace.

```
#!/bin/bash

# Define our commands and profile for authentication; we don't need to see
# successes, just errors
source ./auth.sh demoprofile > /dev/null

# For this script, we want JSON output
export SAS_OUTPUT=json

# Obtain the list of servers...
SERVERS=`$SAS_ADMIN cas servers list | grep \"name\": | cut -f4 -d\"`

# Any errors, if so exit
if [ $? != 0 ]; then
   tput setaf 1
```

```
      echo "ERROR: Unable to get list of CAS servers"
      tput sgr0
      exit 1
   fi

   # Loop through each server, getting the required loggers
   for server in $SERVERS ; do

       # Obtain the loggers with a level of trace in this server
       TRACELOGGERS=`$SAS_ADMIN cas servers loggers list --server $server --
   level trace --all | grep \"name\": | cut -f4 -d\"`

       # Any errors, if so exit
       if [ $? != 0 ]; then
          tput setaf 1
          echo "ERROR: Unable to get list of CAS loggers with a level of
   TRACE"
          tput sgr0
          exit 1
       fi

       # Obtain the loggers with a level of info in this server
       INFOLOGGERS=`$SAS_ADMIN cas servers loggers list --server $server --
   level info --all | grep \"name\": | cut -f4 -d\"`

       # Any errors, if so exit
       if [ $? != 0 ]; then
          tput setaf 1
          echo "ERROR: Unable to get list of CAS loggers with a level of INFO"
          tput sgr0
          exit 1
       fi

       tput setaf 4
       echo "Updating all CAS loggers with a level of TRACE to WARN..."
       tput sgr0

       # For each trace logger, change its level to warn
       for logger in $TRACELOGGERS ; do
           $SAS_ADMIN cas servers loggers update --server $server --logger
   $logger --level warn
       done

       echo

       tput setaf 4
       echo "Updating all CAS loggers with a level of INFO to WARN..."
       tput sgr0

       # For each info logger, change its level to warn
       for logger in $INFOLOGGERS ; do
           $SAS_ADMIN cas servers loggers update --server $server --logger
   $logger --level warn
       done
   done
```

The final script resets the changed CAS loggers back to their original levels:

```
#!/bin/bash

# Define our commands and profile for authentication; we don't need to see
# successes, just errors
source ./auth.sh demoprofile > /dev/null

# For this script, we want JSON output
export SAS_OUTPUT=json

# Obtain the list of servers...
SERVERS=`$SAS_ADMIN cas servers list | grep \"name\": | cut -f4 -d\"`

# Any errors, if so exit
if [ $? != 0 ]; then
   tput setaf 1
   echo "ERROR: Unable to get list of CAS servers"
   tput sgr0
   exit 1
fi

# Loop through each server, reading and restoring the default loggers
for server in $SERVERS ; do

    # Obtain the default loggers with a level of trace in this server by
reading the backup file
    if [ -e $server-trace-cas-loggers.txt ]; then
       TRACELOGGERS=`cat $server-trace-cas-loggers.txt`
    fi

    # Obtain the default loggers with a level of info in this server by
reading the backup file
    if [ -e $server-info-cas-loggers.txt ]; then
       INFOLOGGERS=`cat $server-info-cas-loggers.txt`
    fi

    tput setaf 4
    echo "Restoring all CAS loggers with a default level of TRACE..."
    tput sgr0

    # For each trace logger, restore its level to trace
    for logger in $TRACELOGGERS ; do
        $SAS_ADMIN cas servers loggers update --server $server --logger
$logger --level trace
    done

    echo

    tput setaf 4
    echo "Restoring all CAS loggers with a default level of INFO"
    tput sgr0

    # For each info logger, restore its level to info
    for logger in $INFOLOGGERS ; do
        $SAS_ADMIN cas servers loggers update --server $server --logger
$logger --level info
    done

done
```

THE FOOTBALL PROJECT REDUX

This section is one script that automates the onboarding example found earlier in the paper. Each task is treated separately but data structures are kept and reused where applicable. Some advanced manipulation of CAS artifacts is done and will be explained. Such sections will be highlighted in order to draw attention to them. The JSON files referenced are assumed to have already been created.

```
#!/bin/bash

# Define our commands and profile for authentication; we don't need to see
# successes, just errors
source ./auth.sh demoprofile > /dev/null

# Create the groups

# NewDeptUsers
$SAS_ADMIN identities create-group --name "New Department Users" --id
NewDeptUsers --description "Users in the new department"

# Any errors?
if [ $? != 0 ] ; then

   # Not being able to create groups at this point means we should fail the
   # entire process
   echo Unable to create the group NewDeptUsers
   exit 1
fi
echo

# NewDeptMgrs
$SAS_ADMIN identities create-group --name "New Department Managers" --id
NewDeptMgrs --description "Managers in the new department"

# Any errors?
if [ $? != 0 ] ; then

   # Not being able to create groups at this point means we should fail the
   # entire process
   echo Unable to create the group NewDeptMgrs
   exit 1
fi
echo

# NewDeptAdmins
$SAS_ADMIN identities create-group --name "New Department Administrators" -
-id NewDeptAdmins --description "Administrators in the new department"

# Any errors?
if [ $? != 0 ] ; then

   # Not being able to create groups at this point means we should fail the
   # entire process
   echo Unable to create the group NewDeptAdmins
   exit 1
fi
echo
```

```
# NewDeptDataAdmins
$SAS_ADMIN identities create-group --name "New Department Data
Administrators" --id NewDeptDataAdmins --description "Data administrators
in the new department"

# Any errors?
if [ $? != 0 ] ; then

   # Not being able to create groups at this point means we should fail the
   # entire process
   echo Unable to create the group NewDeptDataAdmins
   exit 1
fi
echo

# The groups are created

# Add NewDeptAdmins to the CAS Admin Users list
$SAS_ADMIN cas admin-users add --server cas-shared-default --group
NewDeptAdmins

# Any errors?
if [ $? != 0 ] ; then

   # Not being able to add NewDeptAdmins to the CAS Admin Users list is a
   # fatal error
   echo Unable to add NewDeptAdmins to the CAS Admin Users list
   exit 1
fi
echo

# Create a folder for the NewDeptDataAdmins group to use
FOLDER_ID=`$SAS_ADMIN -output json folders create --name "New Department
Data Administration" --description "Folder to contains resulting data
analysis reports" | grep \"id\": | cut -f4 -d\"`

if [ "$FOLDER_ID" != "" ] ; then
   echo The requested folder was created with the ID $folderID
else
   echo Could not create the requested folder
   exit 1
fi
echo

# Grant permissions to the NewDeptDataAdmins folder
$SAS_ADMIN authorization create-rule --permissions "read,update,add" --
object-uri /folders/folders/$folderID --group NewDeptDataAdmins
echo

# Now, whitelist the four devices for the new managers
DEVICE_IDS="119569f6-144a-436c-b949-4994e10e838c bb3997d6-73ce-4960-9e14-
a74e4d51469e 97c836ab-9fc6-43a9-8fe0-745c473457bc c0c0ef3d-e209-405c-9c39-
8f65ee7a667e"

# Loop through the devices and add each to the white list
for device in $DEVICE_IDS ; do
```

```
   # Add the device
   $SAS_ADMIN devices whitelist add --device-id $device

   # Any errors?
   if [ $? != 0 ]; then

      # Yes, note them
      echo Could not add the device $device to the white list
   fi
   echo
done

# The following code is for CAS artifacts in the cas-shared-default server

# First, create a super user session for use later
SESSION_ID=`$SAS_ADMIN --quiet cas sessions create --name "Onboarding" --
superuser --server cas-shared-default`

# Success?
if [ "SESSION_ID" == "" ] ; then

   # No, stop here as this could be problematic for the CAS artifacts
   echo Could not create super user session
   exit 1
else

   # Success
   echo Created a super user session with id $SESSION_ID
fi
echo

# Create four path-based caslibs for use by each group created earlier
CASLIBS="NewDeptUsers NewDeptMgrs NewDeptAdmins NewDeptDataAdmins"

# Base file system directory for the caslibs; set accordingly
BASEDIR=/tmp

# Loop through the caslib list and create them, applying the controls if a
successful creation occurs
for caslib in $CASLIBS ; do

   # Create the caslib
   sas-admin cas caslibs create path --server cas-shared-default --caslib
$caslib --path $BASEDIR/$caslib --create-dir --description "caslib for the
group $caslib"

   # Any errors?
   if [ $? != 0 ] ; then

      # Inform the user and stop processing this caslib
      echo Unable to create the caslib $caslib
   else

      echo
      # There are no errors so apply the controls; before doing so, lock
      # the caslib
      $SAS_ADMIN cas transactions checkout --server cas-shared-default --
caslib $caslib --session-id $SESSION_ID
```

```
      # Any errors?
      if [ $? != 0 ] ; then

         # Inform the user and stop processing this caslib
         echo Cannot lock the caslib $caslib; controls won’t be applied
      else

         # Replace the controls
         $SAS_ADMIN cas caslibs replace-controls --server cas-shared-
default --caslib $caslib --session-id $SESSION_ID --source-file
./caslib_controls.json --force

         # Any errors?
         if [ $? != 0 ] ; then

            # Inform the user and stop processing this caslib
            echo Cannot lock the caslib $caslib; controls won’t be applied
         else

            # Commit the changes
            $SAS_ADMIN cas transactions commit --server cas-shared-default
--session-id $SESSION_ID
         fi
      fi
   fi
   echo
done

# We're done with CAS artifacts now so destroy the session we created
$SAS_ADMIN cas sessions delete --server cas-shared-default --session-id
$SESSION_ID --force
echo Delete the CAS session $SESSION_ID
```

The first highlighted section deals with suppressing output. sas-admin supports a “quiet” mode where most of the output is suppressed. The cas plug-in uses this when creating the session and only writes the actual session ID to the screen. This allows for an easier scripting experience.

Another important item to note demonstrated here is that the cas plug-in can use values established in a previous execution. In general, the SAS Viya command-line interfaces are stateless and atomic. However, as seen in this block of code, a CAS session is created and then used in later cas plug-in commands. This approach allows users to maintain a consistent state and conserve CAS resources.

The second highlighted area illustrates the use of CAS access control transactions. Using transactions is a valuable tool to develop if you need to administer CAS artifacts. Being able to lock the artifact in question to a CAS session means that only CAS actions taken in that session can affect the artifact. The technique is used here mainly for illustrative purposes as more than likely, creating a caslib, and then applying access controls to it immediately is low risk for race conditions or collisions.

CONTACT INFORMATION

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

James Holman
SAS Visualization Division ■ Content and System Lifecycle
James.Holman@sas.com