____READ THIS DOCUMENT FIRST____

Cluster System Management (CSM):

Version 1.0.0

User Guide

April 2018

Author:
Fernando Pizzano

Co-Authors:

Nathan Besaw, Nicholas Buonarota, John Dunham, Patrick Lundgren, William Morrison, Alda Ohmacht, Sandy Kaur, and Lars Schneidenbach.

Table of Contents

1 Introduction

The purpose of this document is to familiarize the system administrator with CSM, and act as a reference for when CSM Developers can not be reached.

CSM is developed for use on POWER systems to provide a test bed for the CORAL specific software on non-production systems. This will allow the national laboratories to provide early feedback on the software. This feedback will be used to guide further software development. These POWER systems consist of:

- Witherspoon as compute and utility (Login, Launch and Workload manager) nodes
- Boston as management node and Big Data Store

This document assumes POWER systems are fully operational and running Red Hat Pegas 1.0 or higher.

1.1 Functions and Features

CSM includes support for the following functions and features:

- CSM API support for Burst Buffer, diagnostics and health check, job launch, node inventory, and RAS.
- CSM DB support for CSM APIs, diagnostics and health check, job launch, node inventory, and RAS.
- CSM Infrastructure support for CSM APIs, job launch, node inventory, and RAS.
- RAS event generation, actions, and query.
- Diagnostics and Health Check framework support for CSM prolog/epilog, HTX exercisers.

1.2 Restrictions, Limitations, and Known Issues

CSM does not include support for the following items:

- Shared allocation
- Performance counters and HW statistics (like SSD wear) collection
- Processor inventory collection

Known Issues:

- Whenever CSM is upgraded to a new version, all software that is dependent on CSM must be restarted. LSF and Burst Buffer have daemon processes that must be restarted. JSM is also dependent on CSM, but does not include any daemon processes that need to be explicitly restarted.
- CSM does not support incremental upgrades, fixes, and patches (for example, Alpha 2 to Beta 1, or Beta 1 to PRPQ). System administrator should completely uninstall CSM and its sub components (for example, the CSM DB), then freshly install the new version.
- CSM does not support rpm upgrade (rpm –U), uninstall and re-install is required.
- CSM API responses failing with the error: "`recvmsg timed out. rc=-1`" when the response message to the API grows beyond the maximum supported message size. To work around this issue reduce the number of returned records adding addition filters and/or using the limit option on the API.
- The csmrest daemon must be restarted whenever the associated csmd it is connected to is restarted.
- There are several daemons that all start during system boot when enabled via systemd (xcatd, nvidia-persistenced, dcgm, csmd-master, csmd-aggregator, csmd-utility, csmd-compute, csmrestd, ibmpowerhwmon). Currently the startup of these daemons does not occur in controlled fashion. As a temporary workaround, it may be required to start these services using a script that inserts appropriate wait times between dependent daemons.
- Change of CSM daemon ID of collision
    - There's a 1:3.4*10^36 chance that generated daemonIDs collide. The csm_infrastructure_health_check performs a uniqueness-test to confirm that this didn't happen.

1.3 Reporting issues with CSM

To obtain support and report issues with CSM, please contact IBM Support Service.

1.4 CSM Installation and configuration

When you are ready to install CSM, please see the *CSM Installation and Configuration Guide*, which can be found on Box site.

At this point in time we suggest installing CSM. The following chapters of this user guide provide additional knowledge and reference for sub systems of CSM.

2 CSM Database

2.1 Overview

To create the CSM DB database, you must run the *csm_db_script.sh* script.

The *csm_db_script.sh* creates a log file */var/log/ibm/csm/csm_db_script.log*

Notes:

- CSM DB schema version is pre populated into the csm_db_schema_version table from a csv file
- csm_ras_type table is also pre populated which, contains the description and details for each of the possible RAS event types
- The CSM DB automated script also creates a "csmdb" user with these privileges:
    - SELECT, INSERT, UPDATE, and DELETE on all tables in the schema.

2.1.1 Using csm_db_script.sh

For a quick overview of the script functionality, run ***/opt/ibm/csm/db/csm_db_script.sh –h, --help.*** This help command **(-h, --help)** specifies each of the options available to use.

A new DB set up (default db)	Command	Result
running the script with no options	***./csm_db_script.sh***	This will create a default db with tables and populated data (specified by user or db admin)
running the script with –x, --nodata	***./csm_db_script.sh –x*** ***./csm_db_script.sh --nodata***	This will create a default db with tables and no populated data
A new DB set up (new user db)	**Command**	**Result**
running the script with –n, --newdb	***./csm_db_script.sh -n (my_db_name)*** ***./csm_db_script.sh --newdb (my_db_name)***	This will create a new db with tables and populated data.
running the script with –n, --newdb, -x, --nodata	***./csm_db_script.sh -n (my_db_name) –x*** ***./csm_db_script.sh --newdb (my_db_name) --nodata***	This will create a new db with tables and no populated data.
If a DB already exists	**Command**	**Result**
Drop DB totally	***./csm_db_script.sh -d (my_db_name)*** ***./csm_db_script.sh --delete (my_db_name)***	This will totally remove the DB from the system
Drop only the existing CSM DB tables	***./csm_db_script.sh -e (my_db_name)*** ***./csm_db_script.sh --eliminatetables (my_db_name)***	This will only drop the specified CSM DB tables. (useful if integrated within another DB (e.x. "XCATDB")
Force overwrite of existing DB.	***./csm_db_script.sh -f (my_db_name)*** ***./csm_db_script.sh --force (my_db_name)***	This will totally drop the existing tables in the DB and recreate them with populated table data.
Force overwrite of existing DB.	***./csm_db_script.sh -f (my_db_name) –x*** ***./csm_db_script.sh --force (my_db_name) --nodata***	This will totally drop the existing tables in the DB and recreate them without table data.
Remove just the data from all the tables in the DB	***./csm_db_script.sh -r (my_db_name)*** ***./csm_db_script.sh --removetabledata (my_db_name)***	This will totally remove all data from all the tables within the DB.

Example (Usage)

```
-bash-4.2$ ./csm_db_script.sh -h
=================================================================================================================
[Info ] csm_db_script.sh : CSM database creation script with additional features
[Usage] csm_db_script.sh : [OPTION]... [DBNAME]... [OPTION]
-----------------------------------------------------------------------------------------------------------------
[Options]
------------------------|-----------|-----------------------------------------------------------------------------
  Argument              | DB Name   | Description
------------------------|-----------|-----------------------------------------------------------------------------
  -x, --nodata          | [DEFAULT] | creates database with tables and does not pre populate table data
                        | [db_name] | this can also be used with the -f --force, -n --newdb option when
                        |           | recreating a DB. This should follow the specified DB name
  -d, --delete          | [db_name] | totally removes the database from the system
  -e, --eliminatetables| [db_name] | drops CSM tables from the database
  -f, --force           | [db_name] | drops the existing tables in the DB, recreates and populates with table data
  -n, --newdb           | [db_name] | creates a new database with tables and populated data
  -r, --removetabledata| [db_name] | removes data from all database tables
  -h, --help            |           | help
------------------------|-----------|-----------------------------------------------------------------------------
[Examples]
-----------------------------------------------------------------------------------------------------------------
  [DEFAULT] csm_db_script.sh                          |          |
  [DEFAULT] csm_db_script.sh -x, --nodata             |          |
            csm_db_script.sh -d, --delete             | [DBNAME] |
            csm_db_script.sh -e, --eliminatetables    | [DBNAME] |
            csm_db_script.sh -f, --force              | [DBNAME] |
            csm_db_script.sh -f, --force              | [DBNAME] | -x, --nodata
            csm_db_script.sh -n, --newdb              | [DBNAME] |
            csm_db_script.sh -n, --newdb              | [DBNAME] | -x, --nodata
            csm_db_script.sh -r, --removetabledata    | [DBNAME] |
            csm_db_script.sh -h, --help               |          |
=================================================================================================================
```

Setting up or creating a new DB (manually)

1.) To create your own DB, run **/opt/ibm/csm/db/db_script.sh (–n, --newdb) my_db_name** (where **my_db_name** is the name of your DB). By default if no DB name is specified, then the script will create a DB called **csmdb**.

Example (successful DB creation**):**

```
$ /opt/ibm/csm/db/csm_db_script.sh
-----------------------------------------------------------------------------------------------------------------
[Start   ] Welcome to CSM database automation script.
[Info    ] PostgreSQL is installed
[Info    ] csmdb database user: csmdb already exists
[Complete] csmdb database created.
[Complete] csmdb database tables created.
[Complete] csmdb database functions and triggers created.
[Complete] csmdb table data loaded successfully into csm_db_schema_version
[Complete] csmdb table data loaded successfully into csm_ras_type
[Info    ] csmdb DB schema version (14.24)
-----------------------------------------------------------------------------------------------------------------
```

The script checks to see if the given name exists. If the database does not exist, then it will be created. If the database already exists, then the script prompts an error message indicating a database with this name already exists and exits the program.

Example (DB already exists):

```
$ /opt/ibm/csm/db/csm_db_script.sh
-----------------------------------------------------------------------------------------------------------------
[Info    ] PostgreSQL is installed
[Error   ] Cannot perform action because the csmdb database already exists. Exiting.
-----------------------------------------------------------------------------------------------------------------
```

2.) The script automatically populates data in specified tables using csv files. (For example, ras message type data, into the ras message type table.) If a user does not want to populate these tables, then they should indicate a **(-x, --nodata)** in the command line during the initial setup process.

Example:

```
$ /opt/ibm/csm/db/csm_db_script.sh -x
-------------------------------------------------------------------------------------------------------------
[Info    ] PostgreSQL is installed
[Info    ] csmdb database user: csmdb already exists
[Complete] csmdb database created.
[Complete] csmdb database tables created.
[Complete] csmdb database functions and triggers created.
[Info    ] csmdb skipping data load process.
[Complete] csmdb initialized csm_db_schema_version data
[Info    ] csmdb DB schema version (14.24)
-------------------------------------------------------------------------------------------------------------
```

Existing DB Options

There are some other features in this script that will assist users in a "clean-up" process. If the database already exists, then these actions will work.

1.) Delete the database **(/opt/ibm/csm/db/csm_db_script.sh –d, --delete** followed by the **my_db_name**)

```
$ /opt/ibm/csm/db/csm_db_script.sh -d csmdb
-------------------------------------------------------------------------------------------------------------
[Info    ] PostgreSQL is installed
[Info    ] This will drop csmdb database including all tables and data. Do you want to continue [y/n]?y
[Complete] csmdb database deleted
-------------------------------------------------------------------------------------------------------------
```

2.) Remove just data from all the tables **(/opt/ibm/csm/db/csm_db_script.sh –r**, **--removetabledata** followed by the **my_db_name**)

```
$ /opt/ibm/csm/db/csm_db_script.sh -r csmdb
-------------------------------------------------------------------------------------------------------------
[Info    ] PostgreSQL is installed
[Complete] csmdb database data deleted from all tables excluding csm_schema_version and
           csm_db_schema_version_history tables
-------------------------------------------------------------------------------------------------------------
```

3.) Force a total overwrite of the database (drops tables and recreates them).
(**/opt/ibm/csm/db/./csm_db_script.sh –f, --force** followed by the **my_db_name**)
auto populates table data.

```
$ /opt/ibm/csm/db/csm_db_script.sh -f csmdb
-------------------------------------------------------------------------------------------------------------
[Start   ] Welcome to CSM database automation script.
[Info    ] PostgreSQL is installed
[Info    ] csmdb database user: csmdb already exists
[Complete] csmdb database tables and triggers dropped
[Complete] csmdb database functions dropped
[Complete] csmdb database tables recreated.
[Complete] csmdb database functions and triggers recreated.
[Complete] csmdb table data loaded successfully into csm_db_schema_version
[Complete] csmdb table data loaded successfully into csm_ras_type
[Info    ] csmdb DB schema version (14.24)
-------------------------------------------------------------------------------------------------------------
```

4.) Force a total overwrite of the database (drops tables and recreates them without prepopulated data). (**/opt/ibm/csm/db/csm_db_script.sh –f, --force** followed by the **my_db_name** followed by **–x, --nodata**) does not populate table data.

```
$ /opt/ibm/csm/db/csm_db_script.sh -f csmdb -x
-----------------------------------------------------------------------------------------------------------
[Start   ] Welcome to CSM database automation script.
[Info    ] PostgreSQL is installed
[Info    ] csmdb database user: csmdb already exists
[Complete] csmdb database tables and triggers dropped
[Complete] csmdb database functions dropped
[Complete] csmdb database tables recreated.
[Complete] csmdb database functions and triggers recreated.
[Complete] csmdb skipping data load process.
[Complete] csmdb table data loaded successfully into csm_db_schema_version
[Info    ] csmdb DB schema version (14.24)
-----------------------------------------------------------------------------------------------------------
```

5.) The "csmdb" user will remain in the system unless an admin manually deletes this option. If the user has to be deleted for any reason the Admin can run this command inside the psql postgres DB connection. DROP USER csmdb. If any current database are running with this user, then the user will get a response similar to the example below

```
ERROR:  database "csmdb" is being accessed by other users
DETAIL:  There is 1 other session using the database.
```

It is not recommended to delete the csmdb user. If the process has to be done manually then the admin can run these commands as logged in a postgres super user

Manual process

As root user log into postgres:

```
su - postgres
psql -t -q -U postgres -d postgres -c "DROP USER csmdb;"
psql -t -q -U postgres -d postgres -c "CREATE USER csmdb;"
```

The command below can be executed if specific privileges are needed.

```
psql -t -q -U postgres -d postgres -c "GRANT SELECT, INSERT, UPDATE, DELETE ON ALL TABLES IN
SCHEMA public TO csmdb"
```

If admin wants to change the ownership of the DB to postgres then use the command below.

```
ALTER DATABASE csmdb OWNER TO postgres
ALTER DATABASE csmdb OWNER TO csmdb
```

The automated DB script will check the existence of the DB user. If the user is already created then the process will be skipped. (Please see */var/log/ibm/csm/csm_db_script.log for details)*

2.1.2 **Using csm_db_stats.sh script**

This script will gather statistical information related to the CSM DB which includes, table data activity, index related information, and table lock monitoring.

For a quick overview of the script functionality, run ***/opt/ibm/csm/db/csm_db_stats.sh –h, --help***. This help command (-h, --help) will specify each of the options available to use.

The *csm_db_stats.sh* creates a log file */var/log/ibm/csm/csm_db_stats.log* for each query executed.

Options	Command	Result
Table data activity	***./csm_db_stats.sh –t (my_db_name)*** ***./csm_db_stats.sh --tableinfo (my_db_name)***	see details below
Index related information	***./csm_db_stats.sh –i (my_db_name)*** ***./csm_db_stats.sh --indexinfo (my_db_name)***	see details below
Table Locking Monitoring	***./csm_db_stats.sh –l (my_db_name)*** ***./csm_db_stats.sh --lockinfo (my_db_name)***	see details below
Schema Version Query	***./csm_db_stats.sh –s (my_db_name)*** ***./csm_db_stats.sh --schemaversion (my_db_name)***	see details below
DB connections stats Query	***./csm_db_stats.sh –c (my_db_name)*** ***./csm_db_stats.sh --connectionsdb (my_db_name)***	see details below
DB user stats query	***./csm_db_stats.sh –u (my_db_name)*** ***./csm_db_stats.sh –usernamedb (my_db_name)***	see details below
PostgreSQL Version Installed	***./csm_db_stats.sh -v csmdb*** ***./csm_db_stats.sh --postgresqlversion csmdb***	see details below
--help	***./csm_db_stats.sh (-h, --help)***	see details below

Example (Usage)

```
-bash-4.2$ ./csm_db_stats.sh --help
================================================================================================
[Info ] csm_db_stats.sh : List/Kill database user sessions
[Usage] csm_db_stats.sh : [OPTION]... [DBNAME]
------------------------------------------------------------------------------------------------
  Argument               |  DB Name  | Description
-------------------------|-----------|------------------------------------------------------------
 -t, --tableinfo         | [db_name] | Populates Database Table Stats:
                         |           | Live Row Count, Inserts, Updates, Deletes, and Table Size
 -i, --indexinfo         | [db_name] | Populates Database Index Stats:
                         |           | tablename, indexname, num_rows, tbl_size, ix_size, uk,
                         |           | num_scans, tpls_read, tpls_fetched
 -l, --lockinfo          | [db_name] | Displays any locks that might be happening within the DB
 -s, --schemaversion     | [db_name] | Displays the current CSM DB version
 -c, --connectionsdb     | [db_name] | Displays the current DB connections
 -u, --usernamedb        | [db_name] | Displays the current DB user names and privileges
 -v, --postgresqlversion | [db_name] | Displays the current version of PostgreSQL installed
                         |           | along with environment details
 -h, --help              |           | help
-------------------------|-----------|------------------------------------------------------------
[Examples]
------------------------------------------------------------------------------------------------
   csm_db_stats.sh -t, --tableinfo          [dbname]    | Database table stats
   csm_db_stats.sh -i, --indexinfo          [dbname]    | Database index stats
   csm_db_stats.sh -l, --lockinfo           [dbname]    | Database lock stats
   csm_db_stats.sh -s, --schemaversion      [dbname]    | Database schema version (CSM_DB only)
   csm_db_stats.sh -c, --connectionsdb      [dbname]    | Database connections stats
   csm_db_stats.sh -u, --usernamedb         [dbname]    | Database user stats
   csm_db_stats.sh -v, --postgresqlversion  [dbname]    | Database (PostgreSQL) version
   csm_db_stats.sh -h, --help               [dbname]    | Help menu
================================================================================================
```

1. Table data activity

Run: (***/opt/ibm/csm/db/csm_db_stats.sh –t, --tableinfo [my_db_name]***)

Column_Name	Description
tablename	table name
live_row_count	current row count in the CSM_DB
insert_count	number of rows inserted into each of the tables
update_count	number of rows updated in each of the tables

delete_count	number of rows deleted in each of the tables
table_size	table size

This query will display information related to the CSM DB tables (or other specified DB). The query will display results based on if the insert, update, and delete count is > 0. If there is no data in a particular table it will be omitted from the results.

Example (DB Table info.)

```
-bash-4.2$ ./csm_db_stats.sh -t csmdb
-------------------------------------------------------------------------------------------------------
        relname        | live_row_count | insert_count | update_count | delete_count | table_size
-----------------------+----------------+--------------+--------------+--------------+------------
 csm_db_schema_version |              1 |            1 |            0 |            0 | 8192 bytes
 csm_gpu               |              4 |            4 |            0 |            0 | 8192 bytes
 csm_hca               |              2 |            2 |            0 |            0 | 8192 bytes
 csm_node              |              2 |            2 |            0 |            0 | 8192 bytes
 csm_ras_type          |              4 |            4 |            0 |            0 | 8192 bytes
 csm_ras_type_audit    |              4 |            4 |            0 |            0 | 8192 bytes
(6 rows)
-------------------------------------------------------------------------------------------------------
```

2. Index related information

Run: (***/opt/ibm/csm/db/./csm_db_stats.sh –i, --indexinfo (my_db_name)***)

Column_Name	Description
tablename	table name
indexname	index name
num_rows	number of rows within the table
table_size	table size
index_size	index size
unique	if the index is unique (‘T’ or ‘F’)
number_of_scans	the number returned is the amount of times the index was used
tuples_read	Number of index entries returned by scans on this index
tuples_fetched	Number of live rows fetched by index scans

This query will display information about indexes related to each table in use. Results will be displayed if the num_rows, num_scans, tuples_read, and tuples_fetched count are greater than zero. If there is no data in a particular table it will be omitted from the results.

Example (Indexes)

```
-bash-4.2$ ./csm_db_stats.sh -i csmdb
-------------------------------------------------------------------------------------------------------------------
 tablename |   indexname   | num_rows | table_size | index_size | unique | number_of_scans | tuples_read | tuples_fetched
-----------+---------------+----------+------------+------------+--------+-----------------+-------------+----------------
 csm_node  | ix_csm_node_a |        0 | 8192 bytes | 16 kB      | Y      |              15 |           9 |              9
 csm_gpu   | csm_gpu_pkey  |        0 | 8192 bytes | 16 kB      | Y      |               8 |           4 |              4
 csm_hca   | csm_hca_pkey  |        0 | 8192 bytes | 16 kB      | Y      |               6 |           2 |              2
 csm_ssd   | ix_csm_ssd_a  |        0 | 0 bytes    | 8192 bytes | Y      |               3 |           0 |              0
(4 rows)
-------------------------------------------------------------------------------------------------------------------
```

3. Table Lock Monitoring

Run: (***/opt/ibm/csm/db/./csm_db_stats.sh –l, --lockinfo (my_db_name)***)

Column_Name	Description

blocked_pid	Process ID of the server process holding or awaiting this lock, or null if the lock is held by a prepared transaction
blocked_user	The user that is being blocked
current_or_recent_statement_in_blocking_process	The query statement that is displayed as a result
state_of_blocking_process	Current overall state of this backend.
blocking_duration	Evaluates when the process begin and subtracts from the current time when the query began
blocking_pid	Process ID of this backend
blocking_user	The user that is blocking other transactions
blocked_statement	The query statement that is displayed as a result
blocked_duration	Evaluates when the process begin and subtracts from the current time when the query began

Example (Lock Monitoring)

```
-bash-4.2$ ./csm_db_stats.sh -l csmdb
-[ RECORD 1 ]-----------------------------------+-------------------------------------------------------------
blocked_pid                                     | 38351
blocked_user                                    | postgres
current_or_recent_statement_in_blocking_process | update csm_processor set status='N' where serial_number=3;
state_of+blocking_process                       | active
blocking_duration                               | 01:01:11.653697
blocking_pid                                    | 34389
blocking_user                                   | postgres
blocked_statement                               | update csm_processor set status='N' where serial_number=3;
blocked_duration                                | 00:01:09.048478
-------------------------------------------------------------------------------------------------------------
```

This query displays relevant information related to lock monitoring. It will display the current blocked and blocking rows affected along with each duration. A systems administrator can run the query and evaluate what is causing the results of a "hung" procedure and determine the possible issue.

4. DB schema Version Query

Run: (***/opt/ibm/csm/db/./csm_db_stats.sh –s, --schemaversion (my_db_name)***)

Column_Name	Description
version	This provides the current CSM DB version that is current being used.
create_time	This column indicated when the database was created
comment	This column indicates the "current version" as comment

This query provides the current database version the system is running along with its creation time.

Example (DB Schema Version)

```
-bash-4.2$ ./csm_db_stats.sh -s csmdb
-----------------------------------------------------------------------------------------------------------------------
version |         create_time          |     comment
---------+------------------------------+-----------------
 14.24  | 2018-04-04 09:41:57.784378 | current_version
(1 row)
-----------------------------------------------------------------------------------------------------------------------
```

5. DB Connections with details

Run: (***/opt/ibm/csm/db/./csm_db_stats.sh –c, --connectionsdb (my_db_name)***)

Column_Name	Description
pid	Process ID of this backend
dbname	Name of the database this backend is connected to
usename	Name of the user logged into this backend
backend_start	Time when this process was started, i.e., when the client connected to the server
query_start	Time when the currently active query was started, or if state is not active, when the last query was started
state_change	Time when the state was last changed
wait	True if this backend is currently waiting on a lock
query	Text of this backend's most recent query. If state is active this field shows the currently executing query. In all other states, it shows the last query that was executed.

Example (database connections)

```
-bash-4.2$ ./csm_db_stats.sh -c csmdb
-----------------------------------------------------------------------------------------------------------------------------------------------------------
  pid  | dbname | usename  |         backend_start      |          query_start       |         state_change       | wait |            query
-------+--------+----------+----------------------------+----------------------------+----------------------------+------+-----------------------------
 61427 | xcatdb | xcatadm  | 2017-11-01 13:42:53.931094 | 2017-11-02 10:15:04.617097 | 2017-11-02 10:15:04.617112 | f    | DEALLOCATE
       |        |          |                            |                            |                            |      | dbdpg_p17050_384531
 61428 | xcatdb | xcatadm  | 2017-11-01 13:42:53.932721 | 2017-11-02 10:15:04.616291 | 2017-11-02 10:15:04.616313 | f    | SELECT 'DBD::Pg ping test'
 55753 | csmdb  | postgres | 2017-11-02 10:15:06.619898 | 2017-11-02 10:15:06.620889 | 2017-11-02 10:15:06.620891 | f    |
       |        |          |                            |                            |                            |      | SELECT pid,datname AS dbname,
       |        |          |                            |                            |                            |      | usename,backend_start, q.
       |        |          |                            |                            |                            |      |.uery_start, state_change,
       |        |          |                            |                            |                            |      | waiting AS wait,query FROM pg.
       |        |          |                            |                            |                            |      |._stat_activity;
(3 rows)
-----------------------------------------------------------------------------------------------------------------------------------------------------------
```

This query will display information about the database connections that are in use on the system. The pid (Process ID), database name, user name, backend start time, query start time, state change, waiting status, and query will display statistics about the current database activity.

6. PostgreSQL users with details

Run: (***/opt/ibm/csm/db/./csm_db_stats.sh –u, --usernamedb (my_db_name)***)

Column_Name	Description
rolname	Role name (t/f)
rolsuper	Role has superuser privileges (t/f)
rolinherit	Role automatically inherits privileges of roles it is a member of (t/f)
rolcreaterole	Role can create more roles (t/f)
rolcreatedb	Role can create databases (t/f)
rolcatupdate	Role can update system catalogs directly. (Even a superuser cannot do this unless this column is true) (t/f)

rolcanlogin	Role can log in. That is, this role can be given as the initial session authorization identifier (t/f)
rolreplication	Role is a replication role. That is, this role can initiate streaming replication and set/unset the system backup mode using pg_start_backup and pg_stop_backup (t/f)
rolconnlimit	For roles that can log in, this sets maximum number of concurrent connections this role can make. -1 means no limit.
rolpassword	Not the password (always reads as ********)
rolvaliduntil	Password expiry time (only used for password authentication); null if no expiration
rolconfig	Role-specific defaults for run-time configuration variables
oid	ID of role

Example (DB users with details)

```
-bash-4.2$ ./csm_db_stats.sh -u postgres
-------------------------------------------------------------------------------------------------------------------------------
 rolname  | rolsuper | rolinherit | rolcreaterole | rolcreatedb | rolcatupdate | rolcanlogin | rolreplication | rolconnlimit | rolpassword | rolvaliduntil | rolconfig |  oid
----------+----------+------------+---------------+-------------+--------------+-------------+----------------+--------------+-------------+---------------+-----------+--------
 postgres | t        | t          | t             | t           | t            | t           | t              |           -1 | ********    |               |           |     10
 xcatadm  | f        | t          | f             | f           | f            | t           | f              |           -1 | ********    |               |           |  16385
 root     | f        | t          | f             | f           | f            | t           | f              |           -1 | ********    |               |           |  16386
 csmdb    | f        | t          | f             | f           | f            | t           | f              |           -1 | ********    |               |           | 704142
(4 rows)
-------------------------------------------------------------------------------------------------------------------------------
```

This query will display specific information related to the users that are currently in the postgres database. These fields will appear in the query: rolname, rolsuper, rolinherit, rolcreaterole, rolcreatedb, rolcatupdate, rolcanlogin, rolreplication, rolconnlimit, rolpassword, rolvaliduntil, rolconfig, and oid. See below for details.

7. PostgreSQL Version Installed

Run: (***/opt/ibm/csm/db/./csm_db_stats.sh –v, --postgresqlversion (my_db_name)***)

Column_Name	Description
version	This provides the current PostgreSQL installed on the system along with other environment details.

Example (DB Schema Version)

```
-bash-4.2$ ./csm_db_stats.sh -v csmdb
-------------------------------------------------------------------------------------------------------
                                                version
-------------------------------------------------------------------------------------------------------
 PostgreSQL 9.2.18 on powerpc64le-redhat-linux-gnu, compiled by gcc (GCC) 4.8.5 20150623 (Red Hat 4.8.5-9), 64-bit
(1 row)
-------------------------------------------------------------------------------------------------------
```

This query provides the current version of PostgreSQL installed on the system along with environment details.

2.1.3 **Using csm_db_connections_script.sh**

To list and or kill PostgreSQL database connection(s).

The *csm_db_connections_script.sh* creates a log file
/var/log/ibm/csm/csm_db_connections_script.log
(Each session is logged according to each query executed)

For a quick overview of the script functionality, run ***/opt/ibm/csm/db/./csm_db_connections_script.sh –h, --help.*** This help command **(-h, --help)** will specify each of the options available to use.

A new DB set up (default db)	Command	Result
running the script with no options	**./csm_db_connections_script.sh**	Try 'csm_db_connections_script.sh --help' for more information.
running the script with –l, --list	**./csm_db_connections_script.sh –l, --list**	list database sessions
running the script with -k, --kill	**./csm_db_connections_script.sh –k, --kill**	kill/terminate database sessions
running the script with –f, --force	**./csm_db_connections_script.sh –f, --force**	force kill (do not ask for confirmation, use in conjunction with -k option)
running the script with –u, --user	**./csm_db_connections_script.sh –u, --user**	specify database user name
running the script with –p, --pid	**./csm_db_connections_script.sh –p, --pid**	specify database user process id (pid)

Example (usage)

```
-bash-4.2$ ./csm_db_connections_script.sh --help
[Info ] PostgreSQL is installed
================================================================================================================================
[Info ] csm_db_connections_script.sh : List/Kill database user sessions
[Usage] csm_db_connections_script.sh : [OPTION]... [USER]
--------------------------------------------------------------------------------------------------------------------------------
[Options]
----------------|---------------------------------------------------------------------------------------------------------------
  Argument      | Description
----------------|---------------------------------------------------------------------------------------------------------------
   -l, --list   | list database sessions
   -k, --kill   | kill/terminate database sessions
   -f, --force  | force kill (do not ask for confirmation,
                | use in conjunction with -k option)
   -u, --user   | specify database user name
   -p, --pid    | specify database user process id (pid)
   -h, --help   | help menu
----------------|---------------------------------------------------------------------------------------------------------------
[Examples]
--------------------------------------------------------------------------------------------------------------------------------
   csm_db_connections_script.sh -l, --list                       | list all session(s)
   csm_db_connections_script.sh -l, --list -u, --user [USERNAME] | list user session(s)
   csm_db_connections_script.sh -k, --kill                       | kill all session(s)
   csm_db_connections_script.sh -k, --kill -f, --force           | force kill all session(s)
   csm_db_connections_script.sh -k, --kill -u, --user [USERNAME] | kill user session(s)
   csm_db_connections_script.sh -k, --kill -p, --pid  [PIDNUMBER]| kill user session with a specific pid
================================================================================================================================
```

Listing all DB connections

3.) To display all current DB connections, run **/opt/ibm/csm/db//*./csm_db_connections_script.sh*** **(-l, --list).** The script will display all current sessions open.

Example (-l, --list)

```
-bash-4.2$ ./csm_db_connections_script.sh -l
--------------------------------------------------------------------------------------------------------------------------------
[Start] Welcome to CSM datatbase connections script.
[Info ] PostgreSQL is installed
================================================================================================================================
[Info    ] Database Session     | (all_users):        13
--------------------------------------------------------------------------------------------------------------------------------
  pid  | database |   user    | connection_duration
-------+----------+-----------+---------------------
 61427 | xcatdb   | xcatadm   | 02:07:26.587854
```

```
 61428 | xcatdb   | xcatadm  | 02:07:26.586227
 73977 | postgres | postgres | 00:00:00.000885
 72657 | csmdb    | csmdb    | 00:06:17.650398
 72658 | csmdb    | csmdb    | 00:06:17.649185
 72659 | csmdb    | csmdb    | 00:06:17.648012
 72660 | csmdb    | csmdb    | 00:06:17.646846
 72661 | csmdb    | csmdb    | 00:06:17.645662
 72662 | csmdb    | csmdb    | 00:06:17.644473
 72663 | csmdb    | csmdb    | 00:06:17.643285
 72664 | csmdb    | csmdb    | 00:06:17.642105
 72665 | csmdb    | csmdb    | 00:06:17.640927
 72666 | csmdb    | csmdb    | 00:06:17.639771
(13 rows)
================================================================================================================
```

4.) To display specified user(s) currently connected to the DB, run **/opt/ibm/csm/db//*./csm_db_connections_script.sh* (-l, --list –u, --user <username>).** The script will display the total users connected along with total users.

Example (-l, --list –u, --user)

```
-bash-4.2$ ./csm_db_connections_script.sh -l -u postgres
----------------------------------------------------------------------------------------------------------------
[Start] Welcome to CSM datatbase connections script.
[Info ] DB user: postgres is connected
[Info ] PostgreSQL is installed
================================================================================================================
[Info    ] Database Session     | (all_users):        13
[Info    ] Session List         | (postgres):         1
----------------------------------------------------------------------------------------------------------------
  pid  | database |   user   | connection_duration
-------+----------+----------+---------------------
 74094 | postgres | postgres | 00:00:00.000876
(1 row)
================================================================================================================
```

Example (not specifying user or invalid user in the system)

```
-bash-4.2$ ./csm_db_connections_script.sh -k -u
[Error] Please specify user name
----------------------------------------------------------------------------------------------------------------
-bash-4.2$ ./csm_db_connections_script.sh -k -u csmdbsadsd
[Error] DB user: csmdbsadsd is not connected or is invalid
----------------------------------------------------------------------------------------------------------------
```

Kill all DB connections

The user has the ability to kill all DB connections by using the –k, --kill option. Run **/opt/ibm/csm/db//*./csm_db_connections_script.sh* (-k, --kill).** If this option is chosen by itself, the script will prompt each session with a yes/no request. The user has the ability to manually kill or not kill each session. All responses are logged to the */var/log/ibm/csm/csm_db_connections_script.log*

Example (-k, --kill)

```
-bash-4.2$ ./csm_db_connections_script.sh -k
----------------------------------------------------------------------------------------------------------------
[Start] Welcome to CSM datatbase connections script.
[Info ] PostgreSQL is installed
[Info ] Kill database session (PID:61427) [y/n] ?:
================================================================================================================
```

```
-bash-4.2$ ./csm_db_connections_script.sh -k
----------------------------------------------------------------------------------------------------------------
[Start] Welcome to CSM datatbase connections script.
[Info ] PostgreSQL is installed
[Info ] Kill database session (PID:61427) [y/n] ?:
```

```
[Info ] User response: n
[Info ] Kill database session (PID:61428) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:74295) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:72657) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:72658) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:72659) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:72660) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:72661) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:72662) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:72663) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:72664) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:72665) [y/n] ?:
[Info ] User response: n
[Info ] Kill database session (PID:72666) [y/n] ?:
[Info ] User response: n
================================================================================================================
```

Force kill all DB connections

The user has the ability to force kill all DB connections by using the –k, --kill –f, --force option. Run **/opt/ibm/csm/db//*./csm_db_connections_script.sh* (-k, --kill –f, --force).** If this option is chosen by itself, the script will kill each open session(s). All responses are logged to the */var/log/ibm/csm/csm_db_connections_script.log*

Example (-k, --kill –f, --force)

```
-bash-4.2$ ./csm_db_connections_script.sh -k -f
----------------------------------------------------------------------------------------------------------------
[Start] Welcome to CSM datatbase connections script.
[Info ] PostgreSQL is installed
[Info ] Killing session (PID:61427)
[Info ] Killing session (PID:61428)
[Info ] Killing session (PID:74295)
[Info ] Killing session (PID:72657)
[Info ] Killing session (PID:72658)
[Info ] Killing session (PID:72659)
[Info ] Killing session (PID:72660)
[Info ] Killing session (PID:72661)
[Info ] Killing session (PID:72662)
[Info ] Killing session (PID:72663)
[Info ] Killing session (PID:72664)
[Info ] Killing session (PID:72665)
./csm_db_connections_script.sh: line 360: kill: (72665) - No such process
================================================================================================================
```

Example Log file output:

```
2017-11-01 15:54:27 (postgres) [Start] Welcome to CSM datatbase automation stats script.
2017-11-01 15:54:27 (postgres) [Info ] DB Names:  template1 | template0 | postgres |
2017-11-01 15:54:27 (postgres) [Info ] DB Names:  xcatdb | csmdb
2017-11-01 15:54:27 (postgres) [Info ] PostgreSQL is installed
2017-11-01 15:54:27 (postgres) [Info ] Script execution: csm_db_connections_script.sh -k, --kill
2017-11-01 15:54:29 (postgres) [Info ] Killing user session (PID:61427) kill -TERM 61427
2017-11-01 15:54:29 (postgres) [Info ] Killing user session (PID:61428) kill -TERM 61428
2017-11-01 15:54:29 (postgres) [Info ] Killing user session (PID:74295) kill -TERM 74295
2017-11-01 15:54:29 (postgres) [Info ] Killing user session (PID:72657) kill -TERM 72657
2017-11-01 15:54:29 (postgres) [Info ] Killing user session (PID:72658) kill -TERM 72658
2017-11-01 15:54:30 (postgres) [Info ] Killing user session (PID:72659) kill -TERM 72659
2017-11-01 15:54:30 (postgres) [Info ] Killing user session (PID:72660) kill -TERM 72660
2017-11-01 15:54:30 (postgres) [Info ] Killing user session (PID:72661) kill -TERM 72661
2017-11-01 15:54:30 (postgres) [Info ] Killing user session (PID:72662) kill -TERM 72662
```

```
2017-11-01 15:54:31 (postgres) [Info ] Killing user session (PID:72663) kill -TERM 72663
2017-11-01 15:54:31 (postgres) [Info ] Killing user session (PID:72664) kill -TERM 72664
2017-11-01 15:54:31 (postgres) [Info ] Killing user session (PID:72665) kill -TERM 72665
2017-11-01 15:54:31 (postgres) [Info ] Killing user session (PID:72666) kill -TERM 72666
2017-11-01 15:54:31 (postgres) [End  ] Postgres DB kill query executed
-------------------------------------------------------------------------------------------------------------
```

Kill user connection(s)

The user has the ability to kill specific user DB connections by using the –k, --kill along with –u, --user option. Run **/opt/ibm/csm/db//*./csm_kill_db_connections_test_1.sh* (-k, --kill –u, --user <username>).** If this option is chosen then the script will prompt each session with a yes/no request. The user has the ability to manually kill or not kill each session. All responses are logged to the */var/log/ibm/csm/csm_db_kill_script.log*

Example (-k, --kill –u, --user <username>)

```
-bash-4.2$ ./csm_db_connections_script.sh -k -u csmdb
-------------------------------------------------------------------------------------------------------------
[Start] Welcome to CSM datatbase connections script.
[Info ] DB user: csmdb is connected
[Info ] PostgreSQL is installed
[Info ] Kill database session (PID:61427) [y/n] ?:
-------------------------------------------------------------------------------------------------------------
```

(Single session user kill)

```
-bash-4.2$ ./csm_db_connections_script.sh -k -u csmdb
-------------------------------------------------------------------------------------------------------------
[Start] Welcome to CSM datatbase connections script.
[Info ] DB user: csmdb is connected
[Info ] PostgreSQL is installed
[Info ] Kill database session (PID:61427) [y/n] ?:y
[Info ] Killing session (PID:61427)
-------------------------------------------------------------------------------------------------------------
```

(Multiple session user kill)

```
-bash-4.2$ ./csm_db_connections_script.sh -k -u csmdb
-------------------------------------------------------------------------------------------------------------
[Start] Welcome to CSM datatbase connections script.
[Info ] DB user: csmdb is connected
[Info ] PostgreSQL is installed
[Info ] Kill database session (PID:61427) [y/n] ?:y
[Info ] Killing session (PID:61427)
[Info ] Kill database session (PID: 61428) [y/n] ?:y
[Info ] Killing session (PID:61428)
-------------------------------------------------------------------------------------------------------------
```

Kill PID connection(s)

The user has the ability to kill specific user DB connections by using the –k, --kill along with –p, --pid option. Run **/opt/ibm/csm/db//*./csm_db_connections_script.sh* (-k, --kill –p, --pid <pidnumber>).** If this option is chosen then the script will prompt the session with a yes/no request. The response is logged to the */var/log/ibm/csm/csm_db_connections_script.log*

Example (-k, --kill –u, --pid <pidnumber>)

```
-bash-4.2$ ./csm_db_connections_script.sh -k -p 61427
-------------------------------------------------------------------------------------------------------------
[Start] Welcome to CSM datatbase connections script.
```

```
[Info ] DB PID: 61427 is connected
[Info ] PostgreSQL is installed
[Info ] Kill database session (PID:61427) [y/n] ?:
-------------------------------------------------------------------------------------------------------------
-bash-4.2$ ./csm_db_connections_script.sh -k -p 61427
-------------------------------------------------------------------------------------------------------------
[Start] Welcome to CSM datatbase connections script.
[Info ] DB PID: 61427 is connected
[Info ] PostgreSQL is installed
[Info ] Kill database session (PID:61427) [y/n] ?:y
[Info ] Killing session (PID:61427)
-------------------------------------------------------------------------------------------------------------
```

2.1.4 **Using csm_ras_type_script.sh**

This script is for importing or removing records in the **csm_ras_type** table.

The *csm_db_ras_type_script.sh* creates a log file */var/log/ibm/csm/ csm_db_ras_type_script.log*

Notes:

- csm_ras_type table is pre populated which, contains the description and details for each of the possible RAS event types. This may change over time and new message types can be imported into the table. The script is ran and a temp table is created and appends the csv file data with the current records in the csm_ras_type table. If any duplicate (key) values exist in the process, they will get dismissed and the rest of the records get imported. A total record count is displayed and logged, along with the after live csm_ras_type count and also for the csm_ras_type_audit table.
- A complete cleanse of the csm_ras_type table may also need to take place. If this step is necessary then the auto script can be ran with the –r option. A prompt will display to the admins to ensure this execution is really what they want. If the n option is selected then the process is aborted and results are logged accordingly.

For a quick overview of the script functionality, run ***/opt/ibm/csm/db/ csm_db_ras_type_script.sh –h, --help.*** This help command **(-h, --help)** will specify each of the options available to use.

Example (Usage)

```
-bash-4.2$ ./csm_db_ras_type_script.sh -h
-----------------------------------------------------------------------------------------
[Start   ] Welcome to CSM datatbase ras type automation script.
=========================================================================================
[Info ] csm_db_ras_type_script.sh : Load/Remove data from csm_ras_type table
[Usage] csm_db_ras_type_script.sh : [OPTION]... [DBNAME]... [CSV_FILE]
-----------------------------------------------------------------------------------------
  Argument               |  DB Name  | Description
-------------------------|-----------|-------------------------------------------------------
 -l, --loaddata          | [db_name] | Imports CSV data to csm_ras_type table (appends)
                         |           | Live Row Count, Inserts, Updates, Deletes, and Table Size
 -r, --removedata        | [db_name] | Removes all records from the csm_ras_type table
 -h, --help              |           | help
-------------------------|-----------|-------------------------------------------------------
[Examples]
-----------------------------------------------------------------------------------------
   csm_db_ras_type_script.sh -l, --loaddata          [dbname]    | [csv_file_name]
   csm_db_ras_type_script.sh -r, --removedata        [dbname]    |
   csm_db_ras_type_script.sh -h, --help              [dbname]    |
```

```
================================================================================================
```

Importing records into csm_ras_type table (manually)

5.) To import data to the csm_ras_type table, run **/opt/ibm/csm/db/csm_db_ras_type_script.sh (–l, --loaddata) my_db_name** (where **my_db_name** is the name of your DB) and the **csv_file_name**. The script will check to see if the given name is available and if the database does not exist then it will exit with an error message.

Example (non DB existence**):**

```
-bash-4.2$ ./csm_db_ras_type_script.sh -l csmdb csm_ras_type_data.csv
--------------------------------------------------------------------------------
[Start   ] Welcome to CSM datatbase ras type automation script.
[Info    ] csm_ras_type_data.csv file exists
[Info    ] PostgreSQL is installed
[Error   ] Cannot perform action because the csmdb database does not exist. Exiting.
--------------------------------------------------------------------------------
```

Example (non csv_file_name existence**):**

```
-bash-4.2$ ./csm_db_ras_type_script.sh -l csmdb csm_ras_type_data_file.csv
--------------------------------------------------------------------------------
[Start   ] Welcome to CSM datatbase ras type automation script.
[Error   ] File csm_ras_type_data_file.csv can not be located or doesn't exist
[Info    ] Please choose another file or check path
--------------------------------------------------------------------------------
```

Example (successful execution**):**

```
-bash-4.2$ ./csm_db_ras_type_script.sh -l csmdb csm_ras_type_data_2.csv
--------------------------------------------------------------------------------
[Start   ] Welcome to CSM datatbase ras type automation script.
[Info    ] csm_ras_type_data_2.csv file exists
[Info    ] PostgreSQL is installed
[Info    ] Record import count: 4
[Info    ] csm_ras_type live row count: 4
[Info    ] csm_ras_type_audit live row count: 46
[Info    ] Database csv upload process complete for csm_ras_type table.
--------------------------------------------------------------------------------
```

Removing records from csm_ras_type table (manually)

1. The script will remove records from the csm_ras_type table. The option **(-r, --removedata)** can be executed. A prompt message will appear and the admin has the ability to choose y/n. Each of the logging message will be logged accordingly.

Example (successful execution**):**

```
-bash-4.2$ ./csm_db_ras_type_script.sh -r csmdb
--------------------------------------------------------------------------------
[Start   ] Welcome to CSM datatbase ras type automation script.
```

```
[Info    ] PostgreSQL is installed
[Warning ] This will drop csm_ras_type table data from csmdb database. Do you want to continue [y/n]?
[Info    ] User response: y
[Info    ] Record delete count from the csm_ras_type table: 4
[Info    ] csm_ras_type live row count: 0
[Info    ] csm_ras_type_audit live row count: 50
[Info    ] Data from the csm_ras_type table has been successfully removed
--------------------------------------------------------------------------------
```

Example (unsuccessful execution**):**

```
-bash-4.2$ ./csm_db_ras_type_script.sh -r csmdb
--------------------------------------------------------------------------------
[Start   ] Welcome to CSM datatbase ras type automation script.
[Info    ] PostgreSQL is installed
[Warning ] This will drop csm_ras_type table data from bills_test_db database. Do you want to continue [y/n]?
[Info    ] User response: n
[Info    ] Data removal from the csm_ras_type table has been aborted
--------------------------------------------------------------------------------
```

2.1.5 **Using csm_history_wrapper_archive_script_template.sh**

This section describes the archiving process associated with the CSM DB history tables.

This is a general overview of the CSM DB archive history process demonstrating the csm_node_history table using the *csm_history_wrapper_archive_script_template.sh* script. The script is broken down into three sections which include:

1. Creating a temporary table to archive history data based on a condition.
2. Copies all satisfied history data to a csv file
3. Then updates the archive history timestamp field, which can be later deleted during the purging process)

Section 1:

1. Connects to the DB with the postgres user
2. Drops (if exists) and Creates the temp table used in the archiving process
3. The first query selects all the specific fields in the table
4. The second and third query is a nested query that defines a particular row count that a user can pass in. The data is filter by using the history_time)
5. The where clause defines whether the archive_history_time field is NULL
6. The user will have the option to pass in a row count value (ex. 10,000 records)
7. The data will be ordered by history_time ASC.

Section 2:

1. Copies all the results from the temp table to a csv file

Section 3:

1. Updates the csm_[table_name]_history table

2. Sets the archive_history_time = current timestamp
3. From clause on the temp table
4. WHERE (compares history_time, from history table to temp table) AND history.archive_history_time IS NULL

The script will take in take in these arguments:

1. Database name
2. Archive counter (how many records to be archived)
3. History table name (example: csm_[table_name]_history)
4. Specified directory to be written to

Sample wrapper script:

[./csm_history_wrapper_archive_script_template.sh] [dbname] [archive_counter] [history_table_name] [/data_dir/]

Script out results:

```
-bash-4.2$ ./csm_history_wrapper_archive_script_template.sh csmdb 10000 csm_node_history /tmp/
--------------------------------------------------------------------------------
  Table                         |  Time       |  Archive Count
--------------------------------|-------------|--------------------------------
 csm_node_history               |  0.157      |   10000
--------------------------------------------------------------------------------
 Date/Time:                     |  2018-04-05.09.26.36.411615684
 DB Name:                       |  csmdb
 DB User:                       |  postgres
 archive_counter:               |  10000
 Total time:                    |  0.157
 Average time:                  |  0.157
--------------------------------------------------------------------------------
```

While using the csm_stats_script (in another session) the user can monitor the results ./csm_db_stats.sh –t <db_name>

Directory:
Currently the scripts are setup to archive the results in a specified directory.

Example output file:

The history table data will be archived in a csv file along with the log file: (example below)
csm_db_archive_script.log
csm_node_history.archive.2018-04-05.09.26.36.434758051.csv

2.1.6 **Using csm_history_wrapper_delete_script_template.sh**

This particular script (when called manually) will delete history records which have been archived with a archive_history_timestamp. Records in the history table that do not have an archived_history_timestamp then the records will remain in the system until it has been archived.

The script will take in take in certain flags:

1. Database name
2. Interval time (in minutes)
3. History table name
4. Specified directory to be written to

Sample wrapper script:

[./csm_history_wrapper_delete_script_template.sh] [dbname] [time_mins] [history_table_name] [data_dir]

Script out results:

```
-bash-4.2$ ./csm_history_wrapper_delete_script_template.sh csmdb 1 csm_node_history
/tmp/csm_node_history_delete/
--------------------------------------------------------------------------------
  Table                         | Time        |  Delete Count
--------------------------------|-------------|--------------------------------
 csm_node_history               |  0.005      |   0
--------------------------------------------------------------------------------
 Date/Time:                     |  2018-04-05.09.57.42
 DB Name:                       |  csmdb
 DB User:                       |  postgres
 Interval time:                 |  1 Min(s).
 Total time:                    |  0.005
 Average time:                  |  0.005
```

Directory:
Currently the scripts are setup to archive the results in a specified directory.

Example output file:

The history delete timing results will be logged in a csv file:
csm_db_delete_script.log

2.1.7 **Using csm_db_backup_script_v1.sh**

To (manually) cold backup a CSM database on the system run ***./ csm_db_backup_script_v1.sh***. This script can be run as a root or postgres user. As the root user, log into postgres: su – postgres if desired.

The *csm_db_backup_script_v1.sh* creates a log file

/var/lib/pgsql/backups/csm_db_backup_script.log or specified command line directory

There are a few step that should be taken before backing up a CSM or related DB on the system.

1. Stop all CSM daemons.

2. Run the backup script
    a. Example:(**/opt/ibm/csm/db// *csm_db_backup_script_v1.sh [DBNAME] [/DIR/])***
    b. If a directory is not specified then the log and backup file will be written to the default directory: ***/var/lib/pgsql/backups/***
    c. The script will check the DB connections and if there are no active connections then the backup process will begin.
    d. If there are any active connections to the DB, then an Error message will display the current connections and exit out of the program.
        i. To terminate active connection please use the csm_db_connections_script.sh –h. --help script
3. Once the DB been successfully backed-up then the admin can restart the daemons.

Using csm_db_backup_script_v1.sh

Run ./csm_db_backup_script_v1.sh –h, --help for usage options/overview

```
./csm_db_backup_script_v1.sh -h, --help
================================================================================================================================
[Info ] csm_db_backup_script_v1.sh : csmdb /tmp/csmdb_backup/
[Info ] csm_db_backup_script_v1.sh : csmdb
[Usage] csm_db_backup_script_v1.sh : [OPTION]... [/DIR/]
--------------------------------------------------------------------------------------------------------------------------------
[Options]
----------------|---------------------------------------------------------------------------------------------------------------
  Argument      | Description
----------------|---------------------------------------------------------------------------------------------------------------
   -h, --help   | help menu
----------------|---------------------------------------------------------------------------------------------------------------
[Examples]
--------------------------------------------------------------------------------------------------------------------------------
csm_db_backup_script_v1.sh [DBNAME]                | (default) will backup database to/var/lib/pgpsql/backups/
(directory)
csm_db_backup_script_v1.sh [DBNAME] [/DIRECTORY/   | will backup database to specified directory
                                                   | if the directory doesn't exist then it will be mode and
                                                     written.
================================================================================================================================
```

Common errors

If the user tries to run the script as local user without PostgreSQL installed and does not provide a database name:

1. An info message will prompt ([Info] Database name is required)
2. The usage message will also prompt the usage help menu

```
bash-4.1$ ./csm_db_backup_script_v1.sh
[Info   ] Database name is required
================================================================================================================================
[Info ] csm_db_backup_script_v1.sh : csmdb /tmp/csmdb_backup/
[Info ] csm_db_backup_script_v1.sh : csmdb
[Usage] csm_db_backup_script_v1.sh : [OPTION]... [/DIR/]
--------------------------------------------------------------------------------------------------------------------------------
[Options]
----------------|---------------------------------------------------------------------------------------------------------------
  Argument      | Description
----------------|---------------------------------------------------------------------------------------------------------------
```

```
   -h, --help   | help menu
----------------|-------------------------------------------------------------------------------------------------------
[Examples]
-------------------------------------------------------------------------------------------------------------------
csm_db_backup_script_v1.sh [DBNAME]                | (default) will backup database to/var/lib/pgpsql/backups/
(directory)
csm_db_backup_script_v1.sh [DBNAME] [/DIRECTORY/   | will backup database to specified directory
                                                   | if the directory doesn't exist then it will be mode and
                                                    written.
```

If the user tries to run the script as local user (non-root and postgresql not installed):

```
bash-4.1$ ./csm_db_backup_script_v1.sh csmdb /tmp/
-----------------------------------------------------------------------------------------
[Error ] PostgreSQL may not be installed. Please check configuration settings
-----------------------------------------------------------------------------------------
```

If the user tries to run the script as local user (non-root and postgresql not installed)and doesn't specify a directory (default directory: /var/lib/pgsql/backups

```
bash-4.1$ ./csm_db_backup_script_v1.sh csmdb
-----------------------------------------------------------------------------------------
[Error  ] make directory failed for: /var/lib/pgsql/backups/
[Info   ] User: csmcarl does not have permission to write to this directory
[Info   ] Please specify a valid directory
[Info   ] Or log in as the appropriate user
-----------------------------------------------------------------------------------------
```

2.1.8 Using csm_db_schema_version_upgrade.sh

To migrate the CSM database to the newest schema version run *csm_db_schema_version_upgrade.sh*

The *csm_db_schema_version_upgrade.sh* creates a log file */var/log/ibm/csm/ csm_db_schema_upgrade_script.log*

This script upgrades the CSM (or other specified) DB to the newest schema version. The script has the ability to alter tables, field types, indexes, triggers, functions, and any other relevant DB updates or requests. The script will only modify or add specific fields to the database and never eliminating certain fields.

For a quick overview of the script functionality, run ***/opt/ibm/csm/db/ csm_db_schema_version.sh. –h, --help*** If the script is ran without any options, then the usage function is displayed.

Example (Usage)

```
-bash-4.2$ ./csm_db_schema_version_upgrade.sh -h
-----------------------------------------------------------------------------------------------
[Start   ] Welcome to CSM database schema version upgrade schema script.
[Error   ] Please specify DB name
===============================================================================================
[Info    ] csm_db_schema_version_upgrade.sh : Load CSM DB upgrade schema file
[Usage   ] csm_db_schema_version_upgrade.sh : csm_db_schema_version_upgrade.sh [DBNAME]
-----------------------------------------------------------------------------------------------
  Argument       |  DB Name  | Description
-----------------|-----------|-----------------------------------------------------------------
  script_name    | [db_name] | Imports sql upgrates to csm db table(s) (appends)
```

```
                 |           | fields, indexes, functions, triggers, etc
-----------------|-----------|-------------------------------------------------------------------
===================================================================================================
```

Upgrading CSM DB (manual process)

To upgrade the csm DB, run **/opt/ibm/csm/db/csm_db_schema_version_upgrade.sh my_db_name** (where **my_db_name** is the name of your DB). The script will check to see if the given name exists. If the database name does not exist, then it will exit with an error message.

Example (non DB existence**):**

```
-bash-4.2$ ./csm_db_schema_version_upgrade.sh csmdb
-------------------------------------------------------------------------------
[Start   ] Welcome to CSM database schema version upgrate script.
[Info    ] PostgreSQL is installed
[Error   ] Cannot perform action because the csmdb database does not exist. Exiting.
-------------------------------------------------------------------------------
```

The script will check for the existence of the *csm_db_schema_version_data.csv* file. When an upgrade process happens, the new RPM will consist of a new schema version csv file to be loaded into a (completley new) DB. Once the file is updated then the migration script can be executed. There is a built in check that does a comparison againt the DB schema version and the csv file. (This is just one of the check processes that takes place)

Example (non csv_file_name existence**):**

```
-bash-4.2$ ./csm_db_schema_version_upgrade.sh csmdb
-------------------------------------------------------------------------------
[Start   ] Welcome to CSM database schema version upgrate script.
[Error   ] File csm_db_schema_version_data.csv can not be located or doesn't exist
-------------------------------------------------------------------------------
```

The second check makes sure the file exists and compares the actual SQL upgrade version to the hardcoded version number. If the criteria is met successfully, then the script will proceed. If the process fails, then an error message will prompt.

Example (non compatible migration**):**

```
-bash-4.2$ ./csm_db_schema_version_upgrade.sh csmdb
-------------------------------------------------------------------------------
[Start   ] Welcome to CSM database schema version upgrate script.
[Error   ] Cannot perform action because the $migration_db_version is not compatible.
-------------------------------------------------------------------------------
```

If the user selects the “n/no” option when prompted to migrate to the newest DB schema upgrade, then the program will exit with the message below.

Example (user prompt execution with “n/no” option**):**

```
-bash-4.2$ ./csm_db_schema_version_upgrade.sh csmdb
-------------------------------------------------------------------------------
[Start   ] Welcome to CSM database schema version upgrate script.
[Info    ] PostgreSQL is installed
[Info    ] csmdb current_schema_version 14.23
[Info    ] csmdb schema_version_upgrade: 14.24
[Warning ] This will migrate csmdb database to schema version 14.24. Do you want to continue [y/n]?:
[Info    ] User response: n
[Error   ] Migration session for DB: csmdb User response: ****(NO)****  not updated
-------------------------------------------------------------------------------------------------------------
```

If the user selects the “y/yes” option when prompted to migrate to the newest DB schema upgrade, then the program will begin execution.

Example (user prompt execution with “y/yes” option**):**

```
-bash-4.2$ ./csm_db_schema_version_upgrade.sh csmdb
-------------------------------------------------------------------------------
[Start   ] Welcome to CSM database schema version upgrate script.
[Info    ] PostgreSQL is installed
[Info    ] csmdb current_schema_version 14.23
[Info    ] csmdb schema_version_upgrade: 14.24
[Warning ] This will migrate csmdb database to schema version 14.9. Do you want to continue [y/n]?:
[Info    ] User response: y
[Info    ] csmdb migration process begin.
[Info    ] There are no connections to csmdb
[Complete] csmdb database schema update 14.24.
-------------------------------------------------------------------------------------------------------------
```

If there are existing DB connections, then the migration script will prompt a message and the admin will have to kill connections before proceeding. The *csm_db_connections_script.sh* script can be used with the –l option to quickly list the current connections. (Please see user guide or –h for usage function). This script has the ability to terminate user sessions based on pids, users, or a –f force option will kill all connections if necessary. Once the connections are terminated then the *csm_db_schema_version_upgrade.sh* script can be executed. The log message will display current connection of user, database name, connection count, and duration.

Example (user prompt execution with “y/yes” option and existing DB connection(s)**):**

```
-bash-4.2$ ./csm_db_schema_version_upgrade.sh csmdb
-------------------------------------------------------------------------------
[Start   ] Welcome to CSM database schema version upgrate script.
[Info    ] PostgreSQL is installed
[Info    ] csmdb current_schema_version 14.23
[Info    ] csmdb schema_version_upgrade: 14.24
[Warning ] This will migrate csmdb database to schema version 14.23. Do you want to continue [y/n]?:
[Info    ] User response: y
[Info    ] csmdb migration process begin.
[Error   ] csmdb has existing connection(s) to the database.
```

```
[Error   ] User: csmdb has 1 connection(s)
[Info    ] See log file for connection details
-------------------------------------------------------------------------------------------------------------------
```

3 CSM Infrastructure

3.1 Overview

The CSM infrastructure consists of master, aggregator, utility, and compute daemons.

The CSM master daemon runs on the management node. Aggregators run on the service nodes (optional: run on management node too). The CSM utility daemon runs on the login and launch node. The CSM compute daemon runs on the compute node. This is illustrated below in Figure 1.

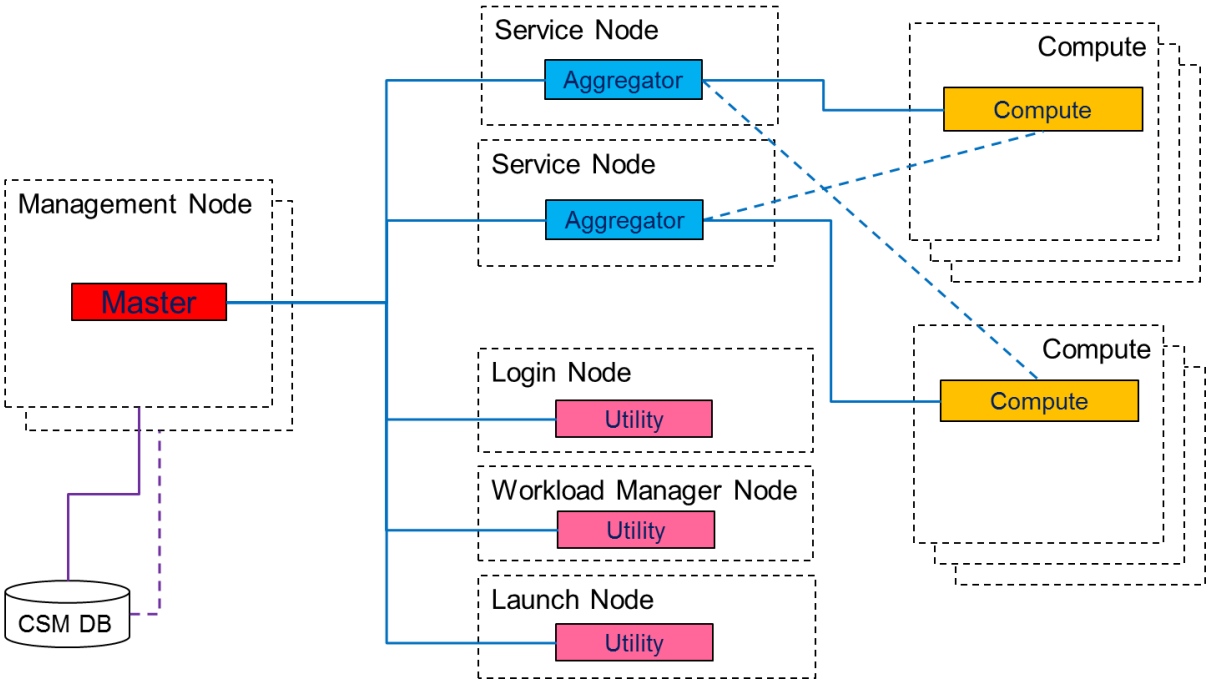


Figure 1: CSM Infrastructure

As shown above, all daemons communicate directly point to point. The compute daemon communicates directly to one aggregator daemon (the primary) and can be configured to connect to a secondary aggregator for fault tolerance however, almost all communication will go through the primary. The aggregator communicates directly to the master daemon. The utility daemon communicates directly to the master daemon. Only the master daemon is allowed to communicate to the CSM database.

3.2 Configuration

Each type of daemon has its own configuration file. Default configuration files can be found here: **"/opt/ibm/csm/share/etc"**

CSM Daemons and Corresponding Configuration Files	
Master Daemon	csm_master.cfg
Aggregator Daemon	csm_aggregator.cfg
Utility Daemon	csm_utility.cfg
Compute Daemon	csm_compute.cfg

If edits are made to a configuration file, then a daemon must be killed and started again.

3.3 Daemon Functionality

Master daemon

CSM master daemon runs on the management node and supports the following activities:

- CSM DB access
- CSM API support
- CSM RAS master functions

Aggregator daemon

The CSM aggregator daemon runs on the management node and facilitates communication between the master daemon and the compute daemons. The aggregator daemon supports the following activities:

- Forwarding all the messages from the compute daemons on compute nodes to the master daemon on the management node without any aggregation functionally.
- Supporting the fan-out of internal CSM multicast messages from the master daemon to a list of compute nodes.
- Keep track of active and inactive compute nodes during its livetime. (This data is not persisted. Therefore, if an aggregator is restarted, this info is only partially recaptured based on the current active set of compute nodes.)
- Allow CSM clients to call APIs similar to the utility daemon functionality.

A new connection from compute nodes is considered a secondary connection until the compute node tells the aggregator otherwise. This assures that any messages along the secondary path between compute nodes and master get filtered.

Utility daemon

The CSM utility daemon runs on the login and launch nodes and supports the following activities:

- Node discovery
- Node inventory
- Steps
- CSM API support
- Environmental data collection

Compute daemon

The CSM compute daemon run on the compute node and support the following activities:

- Node discovery
- Node inventory
- Allocation create and delete
- Environmental data bucket execution
- Preliminary environmental data collection (only GPU data)
- Aggregator failover

The CSM compute, aggregator and utility daemons collect the node inventory and send it to the CSM master daemon where it is stored in the CSM DB. This collection process takes place every time the CSM daemon starts or reconnects after a complete disconnect.

The daemons support authentication and encryption via SSL/TLS. To enable, the administrator has to configure the daemons to use SSL. The CA and certificate files have to be the same for use with point-to-point connections between all daemons.

All daemons are enabled to schedule predefined environmental data collection items. Each of the items can be part of buckets where each bucket can be configured with an execution interval. Currently, CSM only has only one predefined bucket for GPU data collection on the compute and utility daemons.

Compute nodes are able to connect to two aggregators if configured. The compute daemon will try to connect to the first configured aggregator. If this succeeds it will also establish the connection to the secondary aggregator and will then be fully connected. If a secondary aggregator fails, the compute will keep the regular operation with the primary and try to reconnect to the secondary in a preset interval (~10s). If and only if a primary aggregator fails, the compute daemon will perform an automatic failover by telling its current secondary

aggregator to take over the role as the primary for this compute node. If the initial primary aggregator restarts, the compute node will connect to it as a secondary. It will only switch to the other aggregator if the currently active primary connection fails. This behavior has an important system level consequence: once a primary aggregator fails and gets restarted there's currently no other way to restore the configured aggregator-compute relation without restarting the compute daemon.

API Timeout Configuration

Each API comes with a predefined timeout limit of 30 seconds. If this is not sufficient (e.g. because the administrator plans to run prolog scripts that take longer during the creation of an allocation), the limit can be set by adding the API and the new timeout to the csm_api.cfg file.

Note that this setting determines the end-to-end timeout seen at the API-calling client. Any required steps in between will be adjusted by hard-coded ratios of that setting. For example, the timeout for the API handling in the master daemon will be ~90% of the configured timeout.

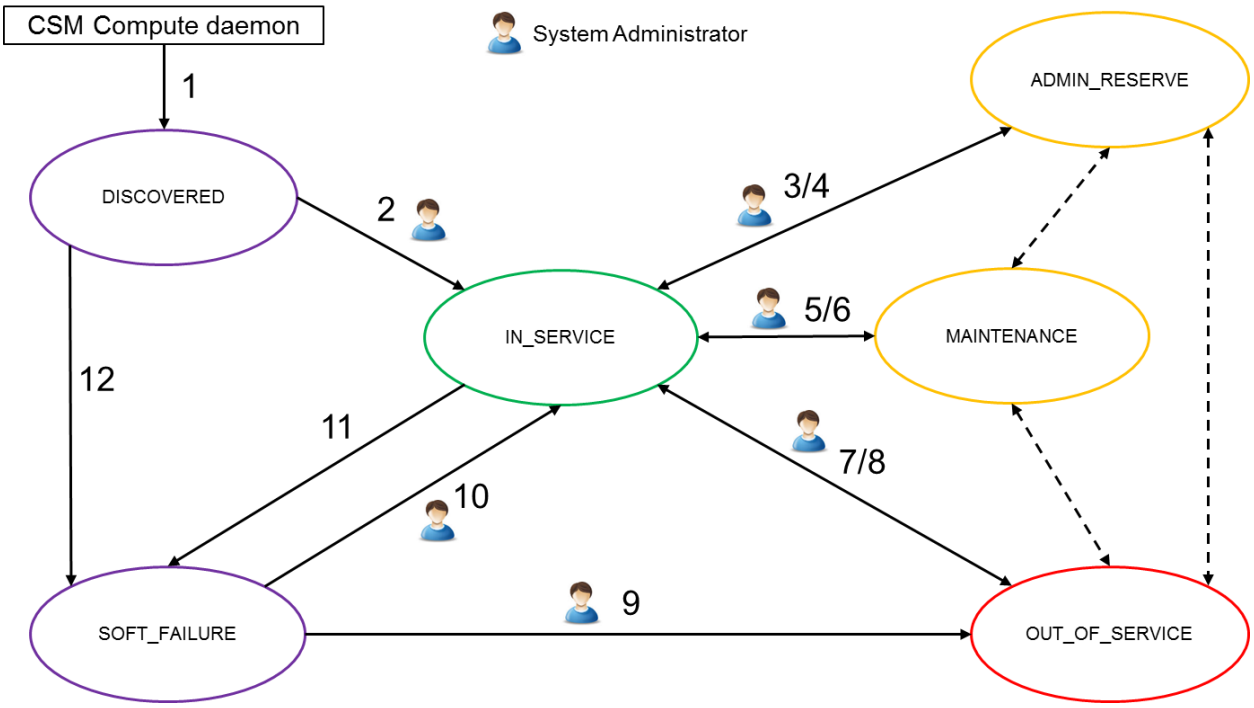
Daemon-to-Daemon Heartbeat

The infrastructure uses a heartbeat mechanism to check if connected daemons can communicate and respond. The configuration file setting csm:net:heartbeat_interval allows to change the default of 15 seconds. Note that about 3 intervals are needed to determine a connection as dead. The heartbeat is implemented in a way that only one direction is checked per heartbeat. This is to improve scalability and reduce the overhead for the network and processing. Whenever a regular CSM message arrives via that connection, the heartbeat timer is reset to avoid unnecessary heartbeats.

Note that the heartbeat is not determining the overall health of a peer daemon. The daemon might be able to respond to heartbeats while still impeded to respond to API calls. A successful exchange of heartbeats tells the daemon that there's a functional network connection and the network mgr thread is able to process inbound and outbound messages. To check if a daemon is able to process API calls, you might use the infrastructure health check tool.

4 Compute node States

State	Ready	Comments
DISCOVERED	No	First time CSM daemon sees the node.
IN_SERVICE	Yes	Node is good for scheduler to use. This is the ONLY state a scheduler should consider the node for scheduling.
ADMIN_RESERVE	No	Reserved for system administrator activities. Still processes RAS events
MAINTENANCE	No	Reserved for system administrator activities. Does NOT process RAS events
SOFT_FAILURE	No	FATAL RAS event Still processes RAS events, but the state will not leave “SOFT_FAILURE”.
OUT_OF_SERVICE	No	Hardware problem Does NOT process RAS events



Number	State A	State B	Comments
1		DISCOVERED	By CSM inventory.
2	DISCOVERED	IN_SERVICE	By system admin action.
3	IN_SERVICE	ADMIN_RESERVE	By system admin action.
4	ADMIN_RESERVE	IN_SERVICE	By system admin action.
5	IN_SERVICE	MAINTENANCE	By system admin action.
6	MAINTENANCE	IN_SERVICE	By system admin action.
7	IN_SERVICE	OUT_OF_SERVICE	By system admin action.
8	OUT_OF_SERVICE	IN_SERVICE	By system admin action.
9	SOFT_FAILUER	OUT_OF_SERVICE	By system admin action.

10	SOFT_FAILURE	IN_SERVICE	By system admin action.
11	IN_SERVICE	SOFT_FAILUER	By CSM RAS subsystem.
12	DISCOVERED	SOFT_FAILURE	By CSM RAS subsystem.

5 Job launch

Any jobs submitted to LSF queues will use the integrated support between LSF and CSM. Depending on the user specifications this integration may also include JSM and BB.

There are few steps required before a job can be launched:

1) The CSM Infrastructure needs to be operational. CSM master, aggregator, utility, and compute daemons need to be up and operational.
2) The CSM compute daemons must collect inventory on the compute nodes and update the CSM DB.
3) System administrator is required to change the “state” field of the compute node to “IN_SERVICE”, using the command line interface of the CSM API *csm_node_attributes_update*.

```
$ /opt/ibm/csm/bin/csm_node_attributes_update –s IN_SERVICE -n c650f06p13
```

6 CSM APIs

6.1 Overview

CSM uses APIs to communicate between its sub systems and to external programs. This section is a general purpose guide for interacting with CSM APIs.

This section is divided into the following subsections:

6.1 Overview – A brief description of the CSM API section.
6.2 Installation – A guide to consult while installing CSM APIs.
6.3 Configuration – A guide to consult while configuring CSM APIs.
6.4 List of CSM APIs – A full list of CSM APIs.
6.5

Implementing new CSM *APIs* – CSM is an open source project that can be contributed to by the community. This section is a guide on how to contribute a new CSM API to this project.

6.2 Installation

The three installation rpms essential to CSM APIs are **ibm-csm-core, ibm-csm-api**, and **ibm-csm-db**. These three rpms must be installed to use CSM APIs.

ibm-csm-core must be installed on the following components:

- management node
- login node
- launch node
- compute node

ibm-csm-api must be installed on the following components:

- management node
- login node
- launch node
- compute node

ibm-csm-db must be installed on the following components:

- management node

6.3 Configuration

CSM APIs have the ability to be configured in various ways. There are default configurations provided for CSM APIs to function, but these settings can be changed and set to a user's preference.

The configurable features of CSM APIs are:

- CSM privileged users
- API security level (privilege, private, and public)
- API log printing
- API timeouts
- allocation/step prolog/epilog user scripts
- The CSM PAM module.

6.3.1 **Configuring user control, security, and access level**

To use CSM APIs at proper security and control levels, an ACL file needs to be configured. Using a combination of a user privilege level and API access level, CSM determines what happens when users call APIs. For example, if the user doesn't have the proper level of privilege on a private API, then information returned will be limited or maybe denied all together.

A default ACL file is included with CSM (*/opt/ibm/csm/share/etc/csm_api.acl*), reproduced below for easy reference.

```
{
  "privileged_user_id": "root",

  "privileged_group_id": "root",

  "private":
  ["csm_allocation_query_details",
   "csm_allocation_step_query_details"],

  "public":
  ["csm_allocation_step_cgroup_create",
   "csm_allocation_step_cgroup_delete",
   "csm_allocation_query",
   "csm_allocation_query_active_all",
   "csm_allocation_step_begin",
   "csm_allocation_step_end",
   "csm_allocation_step_query",
   "csm_allocation_step_query_active_all",
   "csm_diag_run_query",
   "csm_node_attributes_query",
   "csm_node_attributes_query_history",
   "csm_node_resources_query",
   "csm_node_resources_query_all"]
}
```

Source Code 1: csm_api.acl

For standard usage of CSM, this default ACL file may be used. System Administrators may alter this file to fit their environment. The user and user group *root* will always be considered privileged whether listed in this ACL file or not.

Instead of altering the default ACL file, a system administrator may choose to point to a different ACL file for CSM APIs to use. The **api_permission_file** field in the daemon configuration file **csm** section determines the file used by the daemon for ACL.

An excerpt of the *csm_master.cfg* is reproduced below as an example.

```
1 {
2     "csm" :
3     {
4         "role": "Master",
5         "thread_pool_size" : 1,
6         "api_permission_file": "/etc/ibm/csm/csm_api.acl",
7         "api_configuration_file": "/etc/ibm/csm/csm_api.cfg",
```

Source Code 2: api_permission_file excerpt of default master config file

An example of editing this field is shown below.

```
1 {
2     "csm" :
3     {
4         "role": "Master",
5         "thread_pool_size" : 1,
6         "api_permission_file": "/etc/ibm/csm/csm_api_custom_permissions.acl",
7         "api_configuration_file": "/etc/ibm/csm/csm_api.cfg",
```

Source Code 3: edit of api_permission_file field in master config file

If you have trouble finding the config files, then daemon config files are located:

- source repo: “bluecoral/csmconf/”
- ship to: “/opt/ibm/csm/share/”
- run from: “etc/ibm/csm/”

In section 4.2 of the “Installation and Configuration Guide”, a description is provided for adding a new Linux group and making that group privileged in this file.

6.3.2 Configuring a user’s privilege level

A user can be in one of two privileged levels, either *privileged* or *non-privileged*. To be considered a privileged user, you must either be specified as a privileged user in the ACL file or be a member of a user group that is specified in the ACL file. If a user is neither specified nor a part of a user group that was specified, then they are considered a non-privileged user. Below is a capture of the section of the ACL file that deals with setting user privilege.

```
"privileged_user_id": "root",

"privileged_group_id": "root",
```

Source Code 4: privilege level excerpt of default ACL file

Below is an alternate example of the ACL file, if a system administrator was to alter these fields.

```
  "privileged_user_id": "jsmith",

  "privileged_group_id": "csm_admin",
```

Source Code 5: privilege level excerpt of alternate ACL file

The .acl file supports multiple privileged users. An example of an .acl file with multiple privileged users is shown below.

```
"privileged_user_id":
  ["root",
   "jsmith"],

  "privileged_group_id": "sys_admin",
```

Source Code 6: multiple privilege level users

The .acl file also supports multiple privileged user groups. An example of an .acl file with multiple privileged user groups is shown below.

```
"privileged_user_id": "root",

  "privileged_group_id":
   ["sys_admin",
    "csm_admin"],
```

Source Code 7: multiple privilege level user groups

6.3.3 **Configuring API control level**

An API must be set to one of three control levels: *privileged*, *private*, or *public*. If an API is not specified, then it defaults to the privileged level. Below is a capture of the control level section of the default CSM ACL file.

```
  "private":
  ["csm_allocation_query_details",
   "csm_allocation_step_query_details"],

  "public":
  ["csm_allocation_step_cgroup_create",
   "csm_allocation_step_cgroup_delete",
   "csm_allocation_query",
   "csm_allocation_query_active_all",
   "csm_allocation_step_begin",
   "csm_allocation_step_end",
   "csm_allocation_step_query",
   "csm_allocation_step_query_active_all",
   "csm_diag_run_query",
   "csm_node_attributes_query",
   "csm_node_attributes_query_history",
   "csm_node_resources_query",
   "csm_node_resources_query_all"]
```

Source Code 8: control level excerpt of default ACL file

6.3.4 Configuring CSM API logging levels

CSM has lots of things that print out to the logs. Some things are printed at different log levels. You can configure CSM APIs to switch between these log levels. Logging is handled through the CSM infrastructure and divided into two parts, “Front end” and “back end”.

“Front end” is supposed to represent the part of the API a user would interface with and before an API connects and goes into the CSM infrastructure. “Back end” refers to the part of an API that the user would not interact with and after an API connects and goes into the CSM infrastructure.

Front end logging.

Front end logging is done through the csm logging utility. You will need to include the header file to call the function.

```
#include "csmutil/include/csmutil_logging.h"
```

Set your log level with this function:

```
csmutil_logging_level_set(my_level);
```

Where `my_level` is either:

- off
- trace
- debug
- info
- warning
- error
- critical
- always

- disable

After this function is called, the logging level will change. For example, below we set the logging level to “error”. So none of these logging calls will print. When we call the API at the end, then only prints that are at level “error” and above will print.

```
csmutil_logging_level_set(“error”);

//This will print out the contents of the struct that we will pass to the api
    csmutil_logging(debug, "%s-%d:", __FILE__, __LINE__);
    csmutil_logging(debug, "  Preparing to call the CSM API...");
    csmutil_logging(debug, "  value of input:    %p", input);
    csmutil_logging(debug, "  address of input:  %p", &input);
    csmutil_logging(debug, "  input contains the following:");
    csmutil_logging(debug, "    comment:          %s", input->comment);
    csmutil_logging(debug, "    limit:            %i", input->limit);
    csmutil_logging(debug, "    node_names_count: %i", input-
>node_names_count);
    csmutil_logging(debug, "    node_names:       %p", input->node_names);
    for(i = 0; i < input->node_names_count; i++){
        csmutil_logging(debug, "      node_names[%i]: %s", i, input-
>node_names[i]);
    }
    csmutil_logging(debug, "    offset:           %i", input->offset);
    csmutil_logging(debug, "    type:             %s",
csm_get_string_from_enum(csmi_node_type_t, input->type) );

/* Call the C API. */
    return_value = csm_node_attributes_query(&csm_obj, input, &output);
```

If we called the same function, but instead passed in “debug”, then all those logging calls would print, and when we call the API at the end, all prints inside the API that were set to level “debug” and above would print.

CSM API wrappers such as the CMD Line interfaces include access to this function via the –v, --verbose field on the cmd line parameters.

Back end logging.

APIs incorporate the CSM daemon logging system, under the sub channel of “csmapi”. If you want to change the level of default API logging, then you must configure the field in the appropriate csm daemon config file. `"csmapi"` is the field you would need to change. It is found in all the CSM daemon config files, under the “csm” level, then under sub level “log”.

An excerpt of the *csm_master.cfg* is reproduced below as an example.

```
{
    "csm" :
    {
        "log" :
```

```
        {
            "format"                    :   "%TimeStamp% %SubComponent%::%Severity% | %Message%",
            "consoleLog"                :   false,
            "fileLog"                   :   "/var/log/ibm/csm/csm_master.log",
            "__rotationSize_comment_1"  :   "Maximum size (in bytes) of the log file, 10000000000
bytes is ~10GB",
            "rotationSize"              :   10000000000,
            "default_sev"               :   "warning",
            "csmdb"                     :   "info",
            "csmnet"                    :   "info",
            "csmd"                      :   "info",
            "csmras"                    :   "info",
            "csmapi"                    :   "info",
            "csmenv"                    :   "info"
        },
```

Source Code 9: logging excerpt of default master config file

An example of editing this field from "info" to "debug" is shown below.

```
{
    "csm" :
    {
        "log" :
        {
            "format"                    :   "%TimeStamp% %SubComponent%::%Severity% | %Message%",
            "consoleLog"                :   false,
            "fileLog"                   :   "/var/log/ibm/csm/csm_master.log",
            "__rotationSize_comment_1"  :   "Maximum size (in bytes) of the log file, 10000000000
bytes is ~10GB",
            "rotationSize"              :   10000000000,
            "default_sev"               :   "warning",
            "csmdb"                     :   "info",
            "csmnet"                    :   "info",
            "csmd"                      :   "info",
            "csmras"                    :   "info",
            "csmapi"                    :   "debug",
            "csmenv"                    :   "info"
        },
```

Source Code 10: edit of "csmapi" field in master config file

If you have trouble finding the config files, then daemon config files are located:

- source repo: "bluecoral/csmconf/"
- ship to: "/opt/ibm/csm/share/"
- run from: "etc/ibm/csm/"

Note: You may need to restart the daemon for the logging level to change.

If you want to make a run time change to logging, but don't want to change the configuration file. You can use this tool found it here: *opt/ibm/csm/sbin/csm_ctrl_cmd*

You must run this command on the node with the CSM Daemon that you would like to change the logging level of.

6.3.5 **Configuring API timeouts**

CSM APIs return a timeout failure if their execution does not complete in a set amount of time. By default every API times out at 30 seconds. If a system admin wants to extend or shorten this execution window, then they can edit a timeout configuration file.

The packaged configuration can be found here: “etc/ibm/csm/csm_api.cfg”

A default configuration file is included with CSM (*/opt/ibm/csm/share/etc/csm_api.cfg*), reproduced below for easy reference.

```
{
   "csm_allocation_create" : 120,
   "csm_allocation_delete" : 120,
   "csm_allocation_update_state" : 120,
   "csm_allocation_step_end" : 120,
   "csm_allocation_step_begin" : 120,
   "csm_allocation_query" : 120
}
```

Source Code 11: csm_api.cfg

For standard usage of CSM, this default cfg file may be used. System Administrators may alter this file to fit their environment.

I think this file works by specifying an api and a number (in seconds) for the API to time out. By default every API times out at 30 seconds if not over ridden here.

If you wish to change the default API timeout from 30 seconds to something else, then you have to edit the code here: “bluecoral/csmnet/include/csm_timing.h” and tweak the default macros to your new values.

```
// Default timeout.
#define CSM_RECV_TIMEOUT_GRANULARITY ( 5 ) // min and granularity
#define CSM_RECV_TIMEOUT_MIN ( CSM_RECV_TIMEOUT_GRANULARITY ) // shortest
possible timeout
#define CSM_RECV_TIMEOUT_SECONDS ( CSM_RECV_TIMEOUT_GRANULARITY * 6 ) // 30
Secs
#define CSM_RECV_TIMEOUT_MILLISECONDS ( CSM_RECV_TIMEOUT_SECONDS * 1000 )
#define CSM_RECV_TIMEOUT_MILLI_HALF   ( CSM_RECV_TIMEOUT_MILLISECONDS / 2 )
```

We recommend using the configuration file and not touching the source code.

Instead of altering the default configuration file, a system administrator may choose to point to a different file for CSM APIs to use. If this is the case, then the system administrator must update the appropriate daemon configuration files. `"api_configuration_file"` is the field you would need to change. It is found in all the CSM daemon config files, under the “csm” level.

An excerpt of the *csm_master.cfg* is reproduced below as an example.

```
1 {
2     "csm" :
3     {
4         "role": "Master",
5         "thread_pool_size" : 1,
6         "api_permission_file": "/etc/ibm/csm/csm_api.acl",
7         "api_configuration_file": "/etc/ibm/csm/csm_api.cfg",
```

Source Code 12: api_configuration_file excerpt of default master config file

An example of editing this field is shown below.

```
1 {
2     "csm" :
3     {
4         "role": "Master",
5         "thread_pool_size" : 1,
6         "api_permission_file": "/etc/ibm/csm/csm_api.acl",
7         "api_configuration_file": "/etc/ibm/csm/csm_api_custom_timeout.cfg",
```

Source Code 13: edit of api_configuration_file field in master config file

If you have trouble finding the config files, then daemon config files are located:

- source repo: "bluecoral/csmconf/"
- ship to: "/opt/ibm/csm/share/"
- run from: "etc/ibm/csm/"

This feature is primarily used in the context of extending the length of time allocated for Multicast APIs. This is because some of the calls that happen inside of these APIs, such as calling user admin scripts, could vary and take a long time to complete execution.

6.3.6 Configuring allocation prolog and epilog scripts

A *privileged_prolog* and *privileged_epilog* script (with those exact names) must be placed in */opt/ibm/csm/prologs* on a compute node in order to use the *csm_allocation_create*, *csm_allocation_delete* and *csm_allocation_update* APIs. These scripts must be executable and take three command line parameters: *--type*, *--user_flags*, and *--sys_flags*.

To add output from this script to the Big Data Store (BDS) it is recommended that the system administrator producing these scripts make use of their language of choice's logging function.

A sample *privileged_prolog* and *privileged_epilog* written in python is shipped in **ibm-csm-core** at */opt/ibm/csm/share/prologs*. These sample scripts demonstrate the use of the python logging module to produce logs consumable for the BDS.

See Table 1 Mandatory prolog/epilog features, for required implementation features:

Feature	Description
--type	The script must accept a command line parameter *--type* and have support for both allocation and step as a string value.
--sys_flags	The script must have a command line parameter *--sys_flags*. This parameter should take a space delimited list of alphanumeric flags in the form of a string. CSM does not allow special characters, as these represent a potential exposure, allowing unwanted activity to occur.

--user_flags	The script must have a command line parameter *–user_flags*. This parameter should take a space delimited list of alphanumeric flags in the form of a string. CSM does not allow special characters, as these represent a potential exposure, allowing unwanted activity to occur.
Returns 0 on success	Any other error code will be captured by create/delete and the api call will fail.

Table 1 Mandatory prolog/epilog features

See Table 2 Optional prolog/epilog features, for recommend implementation features:

Feature	Description
logging	If the sysadmin wants to track these scripts in BDS, a form of logging must be implemented by the admin writing the script. The sample scripts outline a technique using python and the logging module.

Table 2 Optional prolog/epilog features

See Table 3 Prolog/epilog environment variables, for guaranteed environment variables:

Environment Variable	Description	Allocation Support	Step Support
CSM_ALLOCATION_ID	The Allocation ID of the invoking CSM handler.	Yes	Yes
CSM_PRIMARY_JOB_ID	The Primary Job (Batch) ID of the invoking CSM handler.	Yes	No
CSM_SECONDARY_JOB_ID	The Secondary Job (Batch) ID of the invoking CSM handler.	Yes	No
CSM_USER_NAME	The user associated with the job	Yes	No

Table 3 Prolog/epilog environment variables

6.3.7 **Configuring allocation step prolog and epilog scripts**

A *step prolog* or *step epilog* differs in two ways: the *–type* flag is set to *step* and certain environment variables will not be present. Please refer to Table 1 Mandatory prolog/epilog features and Table 3 Prolog/epilog environment variables for more details.

6.3.8 **Configuring the CSM PAM module**

A CSM PAM module is packaged in **ibm-csm-core** RPM, the following files are installed:

- /etc/pam.d/csm/activelist
- /etc/pam.d/csm/whitelist
- /etc/pam.d/csm/README.md
- /usr/lib64/security/libcsmpam.so

The RPM also adds a disabled line of code to */etc/pam.d/sshd*.

To enable the module for ssh, simply uncomment the added line in */etc/pam.d/sshd* and restart the ssh daemon as outlined in the ***CSM PAM Module*** section in the *CSM Installation and Configuration Guide*. The basic configuration will restrict ssh connections to the root user and any users with an active allocation on the node (who will be placed in the allocation cgroup).

Adding a user to the */etc/pam.d/csm/whitelist* file will authorize them to access the node at any time. This file is new line delimited and will allow any users defined to access through the CSM PAM module. If the system administrator has modified this file, then it will not be removed when **ibm-csm-core** is uninstalled.

The *libcsmpam.so* PAM module is a session module and conforms to the settings described in the pam.conf man page. The only officially supported configuration, however, is bundled in the RPM installation.

For more details about the CSM PAM module, please refer to the bundled */etc/pam.d/csm/README.md* documentation. This document goes into more depth on the behavior of the module, describes the purpose of the active list and provides additional configurations.

6.4 List of CSM APIs

The following is a list of all CSM APIs.

API Name	API Group
csm_allocation_create	Workload Manager
csm_allocation_delete	Workload Manager
csm_allocation_query	Workload Manager
csm_allocation_query_active_all	Workload Manager
csm_allocation_query_details	Workload Manager
csm_allocation_resources_query	Workload Manager
csm_allocation_step_begin	Workload Manager
csm_allocation_step_cgroup_create	Workload Manager
csm_allocation_step_cgroup_delete	Workload Manager
csm_allocation_step_end	Workload Manager
csm_allocation_step_query	Workload Manager
csm_allocation_step_query_active_all	Workload Manager
csm_allocation_step_query_details	Workload Manager
csm_allocation_update_state	Workload Manager
csm_allocation_update_history	Workload Manager
csm_api_object_errcode_get	CSM Session
csm_api_object_errmsg_get	CSM Session
csm_api_object_destroy	CSM Session
csm_bb_cmd	Burst Buffer
csm_bb_lv_create	Burst Buffer
csm_bb_lv_delete	Burst Buffer
csm_bb_lv_update	Burst Buffer
csm_bb_lv_query	Burst Buffer
csm_bb_vg_create	Burst Buffer
csm_bb_vg_delete	Burst Buffer
csm_bb_vg_query	Burst Buffer
csm_diag_result_create	Diagnostics and Health Check
csm_diag_run_begin	Diagnostics and Health Check
csm_diag_run_end	Diagnostics and Health Check
csm_diag_run_query	Diagnostics and Health Check
csm_diag_run_query_details	Diagnostics and Health Check
csm_ib_cable_query	Inventory
csm_ib_cable_query_history	Inventory
csm_ib_cable_update	Inventory
csm_init_lib	CSM Session
csm_node_attributes_query	Inventory
csm_node_attributes_query_details	Inventory

csm_node_attributes_query_history	Inventory
csm_node_attributes_update	Inventory
csm_node_delete	Inventory
csm_node_query_state_history	Inventory
csm_node_resources_query	Workload Manager
csm_node_resources_query_all	Workload Manager
csm_ras_event_create	Reliability Availability Serviceability (RAS)
csm_ras_event_query	Reliability Availability Serviceability (RAS)
csm_ras_event_query_allocation	Reliability Availability Serviceability (RAS)
csm_ras_msg_type_create	Reliability Availability Serviceability (RAS)
csm_ras_msg_type_delete	Reliability Availability Serviceability (RAS)
csm_ras_msg_type_query	Reliability Availability Serviceability (RAS)
csm_ras_msg_type_update	Reliability Availability Serviceability (RAS)
csm_term_lib	CSM Session
csm_switch_attributes_query	Inventory
csm_switch_attributes_query_details	Inventory
csm_switch_attributes_query_history	Inventory
csm_switch_attributes_update	Inventory

Table 4 CSM APIs

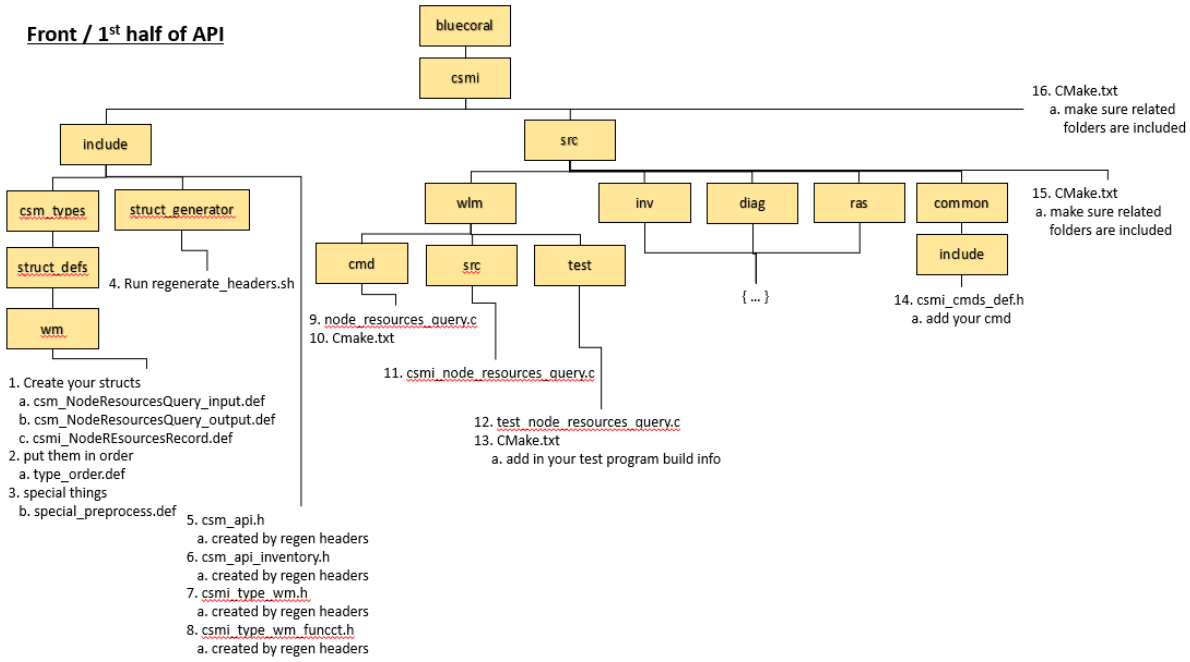
6.5 Implementing new CSM APIs

CSM is an open source project that can be contributed to by the community. This section is a guide on how to contribute a new CSM API to this project.

6.5.1 **Front-end**

“Front end” is supposed to represent the part of the API a user would interface with and before an API connects and goes into the CSM infrastructure.

Follow these steps to create/edit an api. This diagram below shows where to find the appropriate files in the open source repository.



Constructing an API:

The following numbers reference the chart above.

1. When creating an API it should be determined whether it accepts input and produces output. The CSM design follows the pattern of ***<API_Name>_input_t*** for input structs and ***<API_Name>_output_t*** for output structs. These structs should be defined through use of an x-macro in the appropriate folder for the API type under the *csmi/include/csm_types/struct_defs* directory. A README is provided in this directory with an in-depth description of the struct definition process. A sample is reproduced below in Source Code 14 Struct Definition Sample:

```
/*================================================================================

    csmi/include/csm_types/struct_defs/examples/csm_example_api.def

  © Copyright IBM Corporation 2015-2018. All Rights Reserved

    This program is licensed under the terms of the Eclipse Public License
    v1.0 as published by the Eclipse Foundation and available at
    http://www.eclipse.org/legal/epl-v10.html

    U.S. Government Users Restricted Rights:  Use, duplication or disclosure
    restricted by GSA ADP Schedule Contract with IBM Corp.

================================================================================*/

/**
 * CSMI_COMMENT
 * @brief An input wrapper for @ref csm_example_api.
 */

#ifndef CSMI_STRUCT_NAME
    #define CSMI_STRUCT_NAME csm_example_api_input_t

    #undef CSMI_BASIC
    #undef CSMI_STRING
    #undef CSMI_STRING_FIXED
    #undef CSMI_ARRAY
    #undef CSMI_ARRAY_FIXED
```

name of struct type

```
    #undef CSMI_ARRAY_STR
    #undef CSMI_ARRAY_STR_FIXED
    #undef CSMI_STRUCT
    #undef CSMI_ARRAY_STRUCT
    #undef CSMI_ARRAY_STRUCT_FIXED
    #undef CSMI_NONE

    #define CSMI_BASIC              1
    #define CSMI_STRING             1
    #define CSMI_STRING_FIXED       0
    #define CSMI_ARRAY              0
    #define CSMI_ARRAY_FIXED        0
    #define CSMI_ARRAY_STR          1
    #define CSMI_ARRAY_STR_FIXED    0
    #define CSMI_STRUCT             0
    #define CSMI_ARRAY_STRUCT       0
    #define CSMI_ARRAY_STRUCT_FIXED 0
    #define CSMI_NONE               0
#endif

// CSMI_STRUCT_MEMBER(type, name, serial_type, length_member, init_value, extra ) /**< comment */
CSMI_VERSION_START(CSM_VERSION_1_0_0)
CSMI_STRUCT_MEMBER(int32_t , my_first_int         , BASIC    ,                          , -1   , )
/**< Example int32_t value. API will ignore values less than 1.*/
CSMI_STRUCT_MEMBER(uint32_t, my_string_array_count, BASIC    ,                          , 0    , )
/**< Number of elements in the 'my_string_array' array. Must be greater than zero. Size of @ref
my_string_array.*/
CSMI_STRUCT_MEMBER(char**  , my_string_array      , ARRAY_STR, my_string_array_count, NULL, )
/**< comment for my_string_array*/
CSMI_VERSION_END(fc57b7dafbe3060895b8d4b2113cbbf0)

CSMI_VERSION_START(CSM_DEVELOPMENT)
CSMI_STRUCT_MEMBER(int32_t, another_int, BASIC, , -1, ) /**< Another int.*/
CSMI_VERSION_END(0)

#undef CSMI_VERSION_START
#undef CSMI_VERSION_END
#undef CSMI_STRUCT_MEMBER
```

set true or false based off of variables defined below.

Source Code 14 Struct Definition Sample

2. The X-Macro definition files will be collated by their ordering in the local *type_order.def* file. New files added to this ordering should just be the file name. Specific details for this file are in the README.

3. The *special_preprocess.def* file is prepended to the generated header. This file should only be modified if your struct uses a special header or requires some preprocessor directive. Please note that this will apply globally to the generated header file.

4. After defining the X-Macro files the developer should run the *regenerate_headers.sh* script located at *bluecoral/csmi/include/struct_generator/*. This script will prepare the structs and enumerated types for use in the CSM APIs and infrastructure. Serialization functions and python bindings will also be generated. The files modified by this script include:

API Type	*Type Header*	*Function Header*	*Serialization Code*
Common	csmi_type_common.h	csmi_type_common_funct.h	csmi_common_serial.c
Workload Management	csmi_type_wm.h	csmi_type_wm_funct.h	csmi_wm_serialization.c
Inventory	csmi_type_inv.h	csmi_type_inv_funct.h	csmi_inv_serialization.c
Burst Buffer	csmi_type_bb.h	csmi_type_bb_funct.h	csmi_bb_serialization.c

RAS	csmi_type_ras.h	csmi_type_ras_funct.h	csmi_ras_serialization.c
Diagnostic	csmi_type_diag.h	csmi_type_diag_funct.h	csmi_diag_serialization.c
Launch	csmi_type_launch.h	csmi_type_launch_funct.h	csmi_launch_serialization.c

Table 5 regenerate_headers.sh file output

5. Add the API function declaration to the appropriate API file, consult the table below for the correct file to add your API to (in the *bluecoral/csmi/include* directory):

API Type	API File
Common	csm_api_common.h
Workload Management	csm_api_workload_manager.h
Inventory	csm_api_inventory.h
Burst Buffer	csm_api_burst_buffer.h
RAS	csm_api_ras.h
Diagnostic	csm_api_diagnostics.h

Table 6 API Headers

6. Add a command to the *csmi/src/common/include/csmi_cmds_def.h* X-Macro. This will generate an enumerated type in the format of ***CSM_CMD_<csm-contents>*** [cmd(<csm-contents>)] on compilation and used in the front and backend API.

7. The implementation of the C API should be placed in the appropriate *src* directory:

API Type	Source Directory
Common	csmi/src/common/src
Workload Management	csmi/src/wm/src
Inventory	csmi/src/inv/src
Burst Buffer	csmi/src/bb/src
RAS	csmi/src/ras/src
Diagnostic	csmi/src/diag/src

Table 7 API Source Directories

Generally speaking the frontend C API implementation should follow a mostly standard pattern as outlined in Source Code 15 Sample Frontend API Implementation:

```
#include "csmutil/include/csmutil_logging.h"
```

```
#include "csmutil/include/timing.h"
#include "csmi/src/common/include/csmi_api_internal.h"
#include "csmi/src/common/include/csmi_common_utils.h"
#include "csmi/include/“<API_HEADER>

// The expected command, defined in “csmi/src/common/include/csmi_cmds_def.h”
const static csmi_cmd_t expected_cmd = <CSM_CMD>;

// This function must be definedand supplied to the create_csm_api_object
// function if the API specifies an output.
void csmi_<api>_destroy(csm_api_object *handle);

// The actual implementation of the API.
int csm_<api>( csm_api_object **handle, <input_type> *input, <output_type> ** output)
{
        START_TIMING()

        char     *buffer        = NULL; // A buffer to store the serialized input struct.
        uint32_t buffer_length  = 0;    // The length of the buffer.
        char     *return_buffer = NULL; // A return buffer for output from the backend.
        uint32_t return_buffer_len = 0; // The length of the return buffer.
        Int.     error_code  = CSMI_SUCCESS; // The error code, should be of type
                                             // csmi_cmd_err_t.
        // EARLY RETURN
        // Create a csm_api_object and sets its csmi cmd and the destroy function.
        create_csm_api_object(handle, expected_cmd, csmi_<api>_destroy);

        // Test the input to the API, expand this to test input contents.
        if (!input)
        {
                csmutil_logging(error, "The supplied input was null.");

                // The error codes are listed in “csmi/include/csmi_type_common.h”.
                csm_api_object_errcode_set(*handle, CSMERR_INVALID_PARAM);
                csm_api_object_errmsg_set(*handle,
                  strdup(csm_get_string_from_enum(csmi_cmd_err_t, CSMERR_INVALID_PARAM)));
        }

        // EARLY RETURN
         // Serialize the input struct and then test the serialization.
        csm_serialize_struct(<input_type>, input, &buffer, &buffer_length);
        test_serialization(handle, buffer);

        // Execute the send receive command (this is blocking).
        error_code = csmi_sendrecv_cmd(*handle, expected_cmd,
                buffer, buffer_length, &return_buffer, &return_buffer_len);

        // Based on the error code unpack the results or set the error code.
        if ( error_code == CSMI_SUCCESS )
        {
                if ( return_buffer && csm_deserialize_struct(<output_type>, output,
                        (const char *)return_buffer, return_buffer_len) == 0 )
                {
                        // ATTENTION: This is key, the CSM API makes a promise that the
                        // output of the API will be stored in the csm_api_object!
                        csm_api_object_set_retdata(*handle, 1, *output);
                }
                else
                {
                        csmutil_logging(error, "Deserialization failed");
                        csm_api_object_errcode_set(*handle, CSMERR_MSG_UNPACK_ERROR);
                        csm_api_object_errmsg_set(*handle,
                                        strdup(csm_get_string_from_enum(csmi_cmd_err_t,
                                           CSMERR_MSG_UNPACK_ERROR)));
                        error_code = CSMERR_MSG_UNPACK_ERROR;
                }
        }
        else
        {
                csmutil_logging(error, "csmi_sendrecv_cmd failed: %d - %s",
                        error_code, csm_api_object_errmsg_get(*handle));
        }

        // Free the buffers.
        if(return_buffer)free(return_buffer);
        free(buffer);

        END_TIMING( csmapi, trace, csm_api_object_traceid_get(*handle), expected_cmd, api
)
```

```
        return error_code;
}

// This function should destroy any data stored in the csm_api_object by the API call.
void csmi_<api>_destroy(csm_api_object *handle)
{
    csmi_api_internal *csmi_hdl;
    <output_type> *output;

     // free the CSMI dependent data
     csmi_hdl = (csmi_api_internal *) handle->hdl;
     if (csmi_hdl->cmd != expected_cmd)
     {
         csmutil_logging(error, "%s-%d: Unmatched CSMI cmd\n", __FILE__, __LINE__);
         return;
    }

    // free the returned data specific to this csmi cmd
    output = (<output_type> *) csmi_hdl->ret_cdata;
     csm_free_struct_ptr( <output_type>, output);

     csmutil_logging(info, "csmi_<api>_destroy called");
}
```

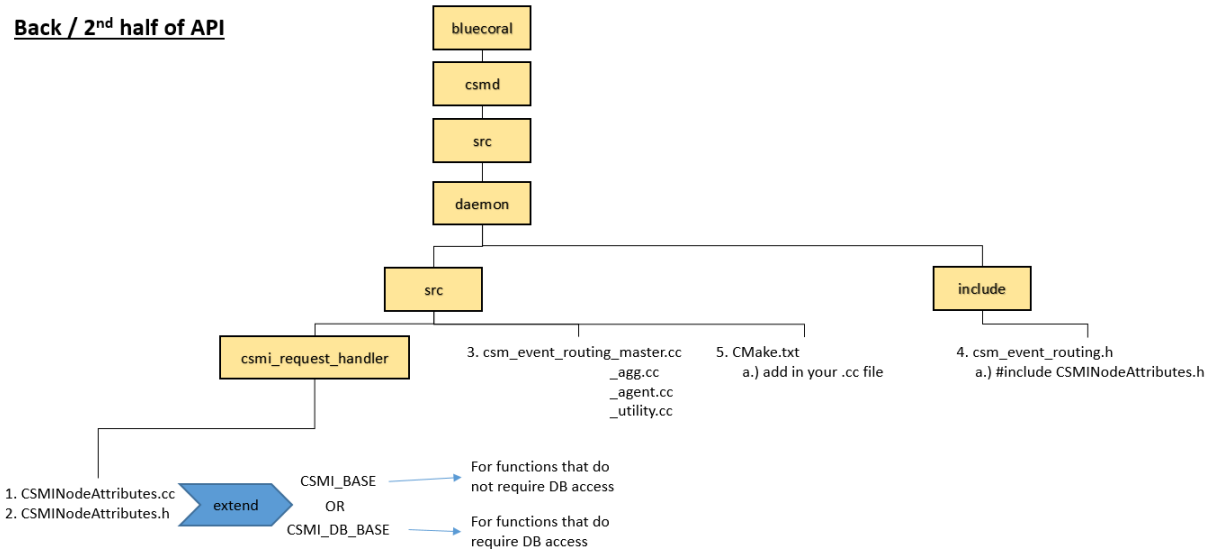
Source Code 15 Sample Frontend API Implementation

8. Optionally, the developer may implement command line interface to the C API. For implementing an API please refer to existing API implementations and section 6.5.4.1.

6.5.2 **Back end**

"Back end" refers to the part of an API that the user would not interact with and after an API connects and goes into the CSM infrastructure.

Follow these steps to create/edit an api. This diagram below shows where to find the appropriate files in the open source repo, along with the numbered order of which ones to work on first.



When implementing a backend API the developer must determine several key details:

- Does the API handler access the database? How many times?
- What daemon will the API handler operate on?
- Does the API need a privilege mode?
- Will the API perform a multicast?

These questions will drive the development process, which in the case of most database APIs is boiler plate as shown in the following sections.

6.5.2.1 Determining the Base Handler Class

In the CSM Infrastructure the back-end API is implemented as an API Handler. This **handler** may be considered a static object which maintains no volatile state. The state of API execution is managed by a **context** object initialized when a request is first received by a back-end handler.

CSM has defined several implementations of handler class to best facilitate the rapid creation of back-end handlers. Unless otherwise specified these handlers are located in *csmd/src/daemon/src/csmi_request_handler* and handler implementations should be placed in the same directory.

6.5.2.1.1 CSMIStatefulDB (csmi_stateful_db.h)

If an API needs to access the database, it is generally recommended to use this handler as a base class. This class provides four virtual functions:

- CreatePayload: Parses the incoming API request, then generates the SQL query.
- CreateByteArray: Parses the response from the database, then generates the serialized response.
- RetrieveDataForPrivateCheck: Generates a query to the database to check the user's privilege level (optional).

- CompareDataForPrivateCheck: Checks the results of the query in *RetrieveDataForPrivateCheck* returning true or false based on the results (optional).

In the simplest Database APIs, the developer needs to only implement two functions: CreatePayload and CreateByteArray. In the case of privileged APIs, the RetrieveDataForPrivateCheck and CompareDataForPrivateCheck must be implemented.

This handler actually represents a state machine consisting of three states which generalize the most commonly used database access path. If your application requires multiple database accesses or multicasts this state machine may be extended by overriding the constructor.

To facilitate multiple database accesses in a single API call CSM has implemented *StatefulDBRecvSend*. *StatefulDBRecvSend* takes a static function as a template parameter which defines the processing logic for the SQL executed by CreatePayload. The constructor for *StatefulDBRecvSend* then takes an assortment of state transitions for the state machine which will depend on the state machine used for the API.

An example of this API implementation style can be found in **CSMIAllocationQuery.cc.** The pertinent section showing expansion of the state machine with the constructor is reproduced and annotated below:

```
#define EXTRA_STATES 1 // There's one additional state being used over the normal StatefulDB.

// Note: CSM_CMD_allocation_query matches the version on the front-end.
CSMIAllocationQuery::CSMIAllocationQuery(csm::daemon::HandlerOptions& options) :
        CSMIStatefulDB(CSM_CMD_allocation_query, options,
                STATEFUL_DB_DONE + EXTRA_STATES) // Send the total number of states to super.
 {
     const uint32_t final_state = STATEFUL_DB_DONE + EXTRA_STATES;
     uint32_t current_state = STATEFUL_DB_RECV_DB;
     uint32_t next_state = current_state + 1;

     SetState( current_state++,
         new StatefulDBRecvSend<CreateResponsePayload>(
             next_state++,    // Successful state.
             final_state,     // Failure state.
              final_state ) ); // Final state.
 }
#undef EXTRA_STATES


bool CSMIAllocationQuery::CreateResponsePayload(
  const std::vector<csm::db::DBTuple *>&tuples,
  csm::db::DBReqContent **dbPayload,
  csm::daemon::EventContextHandlerState_sptr ctx )
{
   // ….
}
```

Source Code 16 Multiple database access

Multicast operations will follow a largely similar behavior, however they exceed the scope of this document, for more details refer to *csmd/src/daemon/src/csmi_request_handler/csmi_mcast*.

6.5.2.1.2 CSMIStateful (csmi_stateful.h)

This handler should be used as a base class in handlers where no database operations are required (see **CSMIAllocationStepCGROUPDelete.h**). Generally, most API implementations will not use this as a base class. If an API is being implemented as *CSMIStateful* it is recommended to

refer the source of **CSMIAllocationStepCGROUPDelete.h** and **CSMIAllocationStepCGROUPCreate.h**.

6.5.2.2 Adding Handler to Compilation

To add the handler to the compilation path for the daemon add it to the *csmd/src/daemon/src/CMakeLists.txt* file's CSM_DAEMON_SRC file GLOB.

6.5.2.3 Registering with a Daemon

After implementing the back-end API the user must then register the API with the daemon routing. Most APIs will only need to be registered on the Master Daemon, however if the API performs multicasts it will need to be registered on the Agent and Aggregator Daemons as well. The routing tables are defined in *csmd/src/daemon/src*:

Daemon	**Routing File**
Agent	csm_event_routing_agent.cc
Aggregator	csm_event_routing_agg.cc
Master	csm_event_routing_master.cc
Utility	csm_event_routing_utility.cc

Table 8 Handler routing

Generally speaking registering a handler to a router is as simple as adding the following line to the RegisterHandlers function: *Register<**Handler_Class**>(**CSM_CMD_<api>**);*

6.5.3 Return Codes

As with all data types that will exist in both the C front-end and C++ back-end return codes are defined with an X-Macro solution. The return code X-Macro file can be located at:

csmi/include/csm_types/enum_defs/common/csmi_errors.def

To protect backwards compatibility this file is guarded by with versioning blocks, for details on how to add error codes please consult the README:

csmi/include/csm_types/enum_defs/README.md

The generated error codes may be included from the *csmi/include/csmi_type_common.h* header. Generally, the **CSMI_SUCCESS** error code should be used in cases of successful execution. Errors should be more granular to make error determination easier for users of the API, consult the list of errors before adding a new one to prevent duplicate error codes.

6.5.4 **CSM API Wrappers**

There exist two documented methodologies for wrapping a CSM API to reduce the barrier of usage for system administrators: python bindings and command line interfaces. Generally speaking python bindings are preferred, as they provide more flexibility to system administrators and end users. Command line interfaces are generally written in C and are used to expose basic functionality to an API.

6.5.4.1 Command Line Interfaces

Command line interfaces in CSM are generally written using native C and expose basic functionality to the API, generally simplifying inputs or control over the output. When properly compiled a native C command line interface will be placed in */csm/bin/* relative to the root of the compiled output. Please consult *csmi/src/wm/cmd/CMakeLists.txt* for examples of compilation settings.

6.5.4.1.1 Naming

The name of the CSM command line interface should be matched one to one to the name of the API, especially in cases where the command line interface simply exposes the function of the API with no special modifications. For example, the *csm_allocation_create* API is literally *csm_allocation_create* on the command line.

6.5.4.1.2 Parameters

CSM command line interfaces must provide long options for all command line parameters. Short options are optional but preferred for more frequently used fields. A sample pairing of short and long options would be in the case of the help flag: *-h, --help*. The *-h, --help* and *-v, --verbose* flag pairings are reserved, always correspond to help and verbose. These flags should be supported in all CSM command line interfaces.

All options should use the *getopts* utility, no options should be position dependent.

> Good:
> *csm_command --node_name node1 --state "some string"*
> *csm_command --state "some string" –node_name node1*
>
> Bad:
> *csm_command **node1** --state "some string"*

6.5.4.1.3 Output

CSM command line requires that the YAML format is a supported output option. This is to facilitate command line parsers. In cases where YAML output is not ideal for command line readability the format may be changed as in the case of **csm_node_query_state_history**:

In the following sample output the output is still considered valid YAML (note the open and close tokens). Data that is not YAML formatted will be commented out with the # character.

```
[root@c650f03p41 bin]# ./csm_node_query_state_history -n c650f03p41
---
node_name: c650f03p41
#         history_time        |      state     |      alteration      | RAS_rec_id, RAS_msg_id
# ----------------------------+----------------+----------------------+------------------------
#  2018-03-26 14:28:25.032879 | DISCOVERED     | CSM INVENTORY        |
#  2018-03-28 19:34:14.037409 | SOFT_FAILURE   | RAS EVENT            | 7, csm.status.down
...
```

Source Code 17 default output

By default, YAML is not presented on the command line. It is supported through a flag.

```
GENERAL OPTIONS:
[-h, --help]                  | Help.
[-v, --verbose verbose_level] | Set verbose level. Valid verbose levels: {off, trace, debug,
info, warning, error, critical, always, disable}
[-Y, --YAML]                  | Set output to YAML. By default for this API, we have a custom
output for ease of reading the long transaction history.
```

Source Code 18 Command line options

By setting the –Y flag, the command line will then display in YAML.

```
[root@c650f03p41 bin]# ./csm_node_query_state_history -n c650f03p41 -Y
---
Total_Records: 2
Record_1:
  history_time: 2018-03-26 14:28:25.032879
  node_name:    c650f03p41
  state:        DISCOVERED
  alteration:   CSM INVENTORY
  RAS_rec_id:
  RAS_msg_id:
Record_2:
  history_time: 2018-03-28 19:34:14.037409
  node_name:    c650f03p41
  state:        SOFT_FAILURE
  alteration:   RAS EVENT
  RAS_rec_id:   7
  RAS_msg_id:   csm.status.down
...
```

Source Code 19 YAML Output

6.5.4.2 Python Interfaces

CSM uses Boost.Python to generate the Python interfaces. Due to warnings in the Boost.Python header used in CSM at this time Python bindings are only supported when building with the Clang compiler. Struct bindings occur automatically when running the *csmi/include/struct_generator/regenerate_headers.sh* script. Each API type has its own file to which the struct bindings will be placed by the automated script and function bindings will be placed by the developer. The following documentation assumes the python bindings are being added to one of the following files:

API Type	Python Binding File	Python Library
Burst Buffer	csmi/src/bb/src/csmi_bb_python.cc	lib_csm_bb_py

Common	csmi/src/common/src/csmi_python.cc	lib_csm_py
Diagnostics	csmi/src/diag/src/csmi_diag_python.cc	lib_csm_diag_py
Inventory	csmi/src/inv/src/csmi_inv_python.cc	lib_csm_inv_py
Launch	csmi/src/launch/src/csmi_launch_python.cc	lib_csm_launch_py
RAS	csmi/src/ras/src/csmi_ras_python.cc	lib_csm_ras_py
Workload Management	csmi/src/wm/src/csmi_wm_python.cc	lib_csm_wm_py

Table 9 Python Binding Files

6.5.4.2.1 Function Binding

Function binding with the Boost.Python library is boilerplate:

```
tuple wrap_<api>(<input-struct> input)
{
    // Always sets the metadata.
    // Ensures that the python binding always matches what it was designed for.
    input._metadata=CSM_VERSION_ID;

    // Output objects.
    csm_api_object * updated_handle;
    <output-struct> * output= nullptr;

    // Run the API
    int return_code = <api>( (csm_api_object**)&updated_handle, &input, &output);

    // A singleton is used to track CSM object handles.
    int64_t oid = CSMIObj::GetInstance().StoreCSMObj(updated_handle);

    // Returned tuples should always follow the pattern:
    // <return code, handler id, output values (optional)>
    return make_tuple(return_code, oid, *output);
}

BOOST_PYTHON_MODULE(lib_csm_<api-type>_py)
{
       def("<api-no-csm>", wrap_<api>, CSM_GEN_DOCSTRING("docstring", ",<output_type>"));
}
```

Source Code 20 Function Binding

6.5.4.2.2 Python Binding Limitations

As CSM was designed predominantly around its use of pointers, and is a C native API, certain operations using the python bindings are not currently Pythonic.

1. The output of the apis must be destroyed using **csm.api_object_destroy(handler_id)**.
2. Array access/creation must be performed through get and set functions. Once an array is set it is currently immutable from python.

These limitations are subject to change.

7 Diagnostic and Health Check

7.1 *Overview*

The Diagnostic application examines the hardware and organizes the results. This is done by running a variety of tests across multiple nodes simultaneously, analyzing resulting errors and data, and generating a simple "PASS" and "FAIL" result.

The diagnostic framework runs as a privileged user. Some of the tests need to run as non-root user, please refer to 7.5 *Security* to create a user to run diagnostics.

Diagnostic PRPQ version supports Firestone (8335-GTA), Garrison (8335-GTB), Witherspoon (8335-GTC, 8335-GTW) and Boston (9006-22C, 5104-22C) machines.

Tests integrated into the diagnostic framework support POWER9 based machines.

The Diagnostic rpm file, **ibm-csm-hcdiag** contains the framework and most of the tests interface. It needs to be installed on all nodes that Diagnostic tests will run.
It will install the following:

- */opt/ibm/csm/hcdiag/bin*, main default root directory for Diagnostic framework
- */opt/ibm/csm/hcdiag/etc,* default location of the configuration files
- */opt/ibm/csm/hcdiag/tests,* default location for the Diagnostic test files
- */opt/ibm/csm/hcdiag/samples,* default location for the source code of tests that is pre-requisite dependent and need to be compiled at the customer environment.

7.2 Framework

The framework manages the scheduling of tests within a run, handles abnormal termination, log directory structures, file locations, etc. It is responsible for parsing all command line parameters and configuration files, organizes the test output file, and it also produces its own log file.

The framework is driven by a configuration file, which is easily customized and extended.

The master configuration file, the tests configuration files, and information retrieved from the CSM database, such as: hardware components and their status, are all used as input to the framework.

The Framework generates a unique identifier of a Diagnostic run, called a "run_id". The run_id is embedded in the directories and log files it generates to clarify which run the output came from. The syntax of the run_id is **"ddddddttttttttttt**", where d is the date in compact form, t is the timestamp up to microseconds (6 digits).

The Diagnostic framework can run in either “Management” or “Node” mode:

- ***Management mode:*** In this mode, Diagnostic runs on the Management node and uses the xCAT environment. It addresses multiple nodes, runs tests, and consolidates the output.

  In this mode, the Diagnostic application can be invoked by the RAS handler.

- ***Node mode***: Run diagnostics in this mode for a non-management node with no xCAT environment. All the tests are executed on the node. In this mode, the Diagnostic application can be invoked by LSF, and from a job prolog/epilog.

The diagnostic system can run with or without CSM. But when run with CSM, more features are enabled.

- Integrated with CSM: records allocations, diagnostics run and results into the CSM database; communicates with RAS subsystem;
- With no CSM: diagnostic results are summarized and displayed in the console. The summary is also saved in the log directory.

The framework is developed in Python 2.7.5, the modules that interfaces with CSM APIs are developed in C and the tests modules are either bash or perl.

7.3 Configuration files

There are three configuration files located at */opt/ibm/csm/hcdiag/etc*, that need to be configured to match the customer environment:

7.3.1 **test.properties**

This file describes the tests supported by the diagnostics. This file also defines diagnostic buckets and collection of tests, to make it easier to run an set of tests.

There are two sections in this file:

- A test section, enclosed in square brackets, and starting with the word “tests”.
- A bucket section, also enclosed in square brackets, and starting with the word “bucket”.

Attributes of a test section:

Name	Description
name	The name of the test
description	Short description of the test
group	Group that the test belongs to

timeout	Time in seconds xCAT waits for output from a program being executed in the remote nodes.
targetType	Type of the node the test applies to. Valid values are: Compute, Management
executable	Full path of the executable.
args	Arguments that will be passed to the executable.
clusterTest	Tells the framework if this is a single node test or a cluster test. Valid values are: yes, no
xcat_cmd	Tells the framework that the test uses xcat command, and t can run if the machine is not up.

Table 10 test.propeties file, tests section

Attributes of a bucket section:

Name	Description
name	The name of the bucket.
description	Short description of the bucket.
tests	List of all the tests, comma separated.

Table 11 test.propeties file, bucket section

7.3.2 **hcdiag.properties**

This file is the diagnostic framework configuration file. It defines the behavior of the application, log directory, verbose mode, etc.

Name	Description
common_fs	Tells the framework if the "logdir" is located in a common filesystem or not. If set to "yes", the output is stored locally on the remote machines, otherwise data will be sent to the management node via pipe.
console_verbose	Sets the console verbosity. Valid values are: debug, info, warn, error, critical
csm	Specify if CSM environment will be used or not.
csmi_bindir	Specify the location of the csm api binaries.
installdir	Specify the Diagnostic install root directory
bds	Specify if Diagnostic log files will be integrated into BDS via syslog. If set to yes, diagnostic will write into the syslog.
logdir	Specify the directory to store the framework log and the tests output
log_verbose	Sets the log verbosity for the hcdiag log file and the syslog (if bds is set to yes). Valid values are: debug, info, warn, error, critical

stoponerror	Set the framework behavior when a diagnostic test fails. • no: the remaining tests will run on all nodes • node: the remaining tests will run on all nodes but the ones that failed • system: Diagnostic application will stop
tempdir	Temporary directory used by the framework to copy the executables on all remote nodes.
testproperties	Specify the full path of the tests properties file.
timeout	xCAT timeout in second. Timeout value for the remote command execution. If any remote target does not provide output to either stdout or stderr within the timeout value, xCAT command will display an error message and exists.
tty_progress_interval	Time in seconds used by the framework to show progress, displaying '…' on the console.
watch_output_progress	Time in seconds used by the framework to monitor the output received from the remote nodes. If set to 0, the framework will not monitor the output and wait for the remote test execution to complete or to timeout.
xcat_bindir	Location of the xcat binaries: xdsh, xdcp, nodestat. Usually /opt/xcat/bin.
xcat_fanout	Minimum number of concurrent remote shell command processes.

Table 12 hcdiag.properties file

7.3.3 **clustconf.yaml**

The clustconf file is a YAML format configuration. It defines the cluster/set of nodes grouped by its attributes. We have the following sections in the file:

Command within square brackets are used by Diagnostic to validate the values.

node_info: each case statement defines a cluster/set of nodes that has common attributes.

case: list of nodes

rvitals: points to one of the rvitals definition in the rvitals section, Table 15 clustconf.yaml: rvitals

ncpus: number of cpus .

[lscpu |grep ^CPU\(s\)]

memory: amount of memory

clock:

max: maximum cpu clock frequency.

[ppc64_cpu –frequency]

min: minimum cpu clock frequency.

[ppc64_cpu --frequency]

firmware: P9 firmware levels.

[rinv <node> firm]

gpu:

pciids: list of all nvidia pci devices IDs.
[nvidia-smi --query-gpu=gpu_bus_id --format=csv]
device: the official product name of the gpu.
[nvidia-smi --query-gpu=gpu_name --format=csv]
vbios: the bios of the GPU board.
[nvidia-smi --query-gpu=vbios_version --format=csv]
clocks_applications_gr: frequency of graphics clock.
[nvidia-smi –query-gpu=clocks.applications.gr –format=csv]
clocks_applications_mem: Frequency of memory clock.
[nvidia-smi –query-gpu=clocks.application.mem –format=csv]
persistence_mode: flag that indicates whether persistence mode is enabled or not.
[nvidia-smi –query-gpu=persistence_mode –format=csv]

ib:
slot_rx: list of rx adapters
pciids: list of all Mellanox pci devices IDs
[lspci | grep Mellanox| awk '{print $1}']
board_id: adapter card's PSID (Parameter-Set Identification)
[ibv_devinfo | grep board_id]
firmware: adapter firmware version
[ibv_devinfo | grep fw_ver]

os:
name: operating system name
pretty_name: formatted operating system name

kernel:
release: kernel version

ufm:
ip_address: IP address of UFM
user: username for authentication
pw: password for authentication

nvme:
vendor: NVMe device vendor name
firmware_rev: NVMe device firmware level

temp:
celsius_high: high threshold for sensor temperature testing, in Celsius
celsius_low: low threshold for sensor temperature testing, in Celsius

```
node_info:
  - case: c650f99p(04|06|08|10|12)
    # water cooled dd2.01 cluster with 4 GPUS
    # ======================================
    rvitals: wspoon_dd2
```

```
    ncpus: 176
    memory: 512
    clock:
      max: 3.8
      min: 2.0
    firmware:
      name: 1742Ex
      versions:
        - 'HOST Firmware Product: IBM-witherspoon-ibm-OP9_v1.19_1.111 (Active)*'
        - 'HOST Firmware Product: -- additional info: buildroot-2017.8-8-g5e23247'
        - 'HOST Firmware Product: -- additional info: capp-ucode-p9-dd2-v2'
        - 'HOST Firmware Product: -- additional info: hostboot-7050d0a'
        - 'HOST Firmware Product: -- additional info: hostboot-binaries-159f2e8'
        - 'HOST Firmware Product: -- additional info: linux-4.13.16-openpower1-
p98dd271'
        - 'HOST Firmware Product: -- additional info: machine-xml-592a6dd'
        - 'HOST Firmware Product: -- additional info: occ-39d5490'
        - 'HOST Firmware Product: -- additional info: op-build-v1.19-320-gf959966-
dirty'
        - 'HOST Firmware Product: -- additional info: petitboot-v1.6.3-p9911320'
        - 'HOST Firmware Product: -- additional info: sbe-b6b7bb7'
        - 'HOST Firmware Product: -- additional info: skiboot-v5.9.8'
        - 'BMC Firmware Product: ibm-v2.0-0-r26-0-g7285b52 (Active)*'
    gpu:
      pciids:
        - '0004:04:00.0'
        - '0004:05:00.0'
        - '0035:03:00.0'
        - '0035:04:00.0'
      device: "Tesla V100-SXM2-16GB"
      vbios: "88.00.13.00.02"
      clocks_applications_gr: 1312
      clocks_applications_mem: 877
      persistence_mode: Enabled
    ib:
      slot_rx: "0003:01:00.[01]|0033:01:00.[01]"
      pciids:
        - '0003:01:00.0'
        - '0003:01:00.1'
        - '0033:01:00.0'
        - '0033:01:00.1'
      board_id: "IBM0000000002"
      firmware: "16.21.0106"
    os:
      name: "Red Hat Enterprise Linux Server"
      pretty_name: "Red Hat Enterprise Linux Server 7.5 Beta (Maipo)"
    kernel:
      release: "4.14.0-38.el7a.ppc64le"
    ufm:
      ip_address: "10.7.0.41"
      user: "admin"
      pw: "123456"
    nvme:
      vendor: "Samsung"
      firmware_rev: "MN12MN12"
    temp:
      celsius_high: "35.0"
      celsius_low: "14.0"
```

Table 13 clustconf.yaml: node_info

- *gpfs_mounts*: defines gpfs mounts expected on allnodes

```
gpfs_mounts:
  - {mount: '/gpfs/r92gpfs01',  match: 'r92gpfs01' }
  - {mount: '/gpfs/r92gpfs02',  match: 'r92gpfs02' }
```

Table 14 clustconf.yaml: gpfs_mounts

- *rvitals*: defines the rvitals parameter, such as temperature, voltage, current, etc
  [rvitals <noderange>]

```
wspoon_dd2:
    - {id: 'Ambient', regex: (\S+), range: [10,40] }
    - {id: 'Fan1 \d',  regex: (\S+), range: [0,24000] }
    - {id: 'Fan[0-3] \d',  regex: (\S+), range: [2500,14000] }
    - {id: 'P\d Vcs Temp',  regex: (\S+), range: [15,80]}
    - {id: 'P\d Vdd Temp',  regex: (\S+), range: [15,80]}
    - {id: 'P\d Vddr Temp',  regex: (\S+), range: [15,80]}
    - {id: 'P\d Vdn Temp',  regex: (\S+), range: [15,80]}
    - {id: 'Ambient', regex: (\S+), range: [10,40] }
    - {id: 'Dimm\d+ Temp', regex: (\S+), range: [15,75,N/A]}
    - {id: 'P\d Core6 Temp',    regex: (\S+), range: [0,90,N/A]}
    - {id: 'P\d Core7 Temp',    regex: (\S+), range: [0,90,N/A]}
    - {id: 'P\d Core\d Temp',    regex: (\S+), range: [10,90,N/A]}
    - {id: 'P\d GPU Power',  regex: (\S+), range: [10,1800,N/A] }
    - {id: 'P\d Io Power',  regex: (\S+), range: [10,500,N/A] }
    - {id: 'P\d Mem Power',  regex: (\S+), range: [10,500,N/A] }
    - {id: 'P\d Power',  regex: (\S+), range: [1,1800,N/A] }
    - {id: 'Ps\d Input Power',  regex: (\S+), range: [10,1800,N/A] }
    - {id: 'Ps\d Input Voltage',  regex: (\S+), range: [200,285,N/A] }
    - {id: 'Ps1 Output Current',  regex: (\S+), range: [0,100,N/A] }
    - {id: 'Ps\d Output Voltage',  regex: (\S+), range: [10,400,0] }
    - {id: 'Ps\d Output Current',  regex: (\S+), range: [10,100,N/A] }
    - {id: 'Ps\d Output Voltage',  regex: (\S+), range: [300,400,N/A] }
    - {id: 'Storage A Power',  regex: (\S+), range: [10,500,N/A] }
    - {id: 'Storage B Power',  regex: (\S+), range: [10,500,N/A] }
    - {id: 'Total Power',  regex: (\S+), range: [10,2000,N/A] }
```

Table 15 clustconf.yaml: rvitals

7.4 Tests

Tests are, by default, installed under: */opt/ibm/csm/hcdiag/tests/<testname>*. Each test should have its own subdirectory.

For example, for test *test_simple:*

/opt/ibm/csm/hcdiag/tests/test_simple/test_simple.sh is where the test will be installed and tests.properties file must have the minimal entry:

```
[tests.test_simple]

description = This is a simple hello test

executable  = /opt/ibm/csm/hcdiag/tests/test_simple/test_simple.sh
```

In Management mode, if the test is visible to the target, the framework just executes it. If the test is not visible to the target, the framework will copy it to the target prior to the execution.

The tests themselves might need to be customized. We assume:

- Default directories for binaries and scripts installed by external RPMs, this might not match the customer's installation.
- Default behaviour for certain hardware component: for example GPU persistent mode.

Few definitions prior to describe the tests:

Test: A single executable running on a single or multiple nodes that exercises one portion of the system.

Test Group: An abstract collection of test, based on specific hardware coverage, for example memory, processor, GPU, and network.

***Test Bucket*:** A collection of test cases configured per some principle. For example, a bucket might be based on node health. Alternatively, it might be based on the thoroughness of the health check or on the network stress it provides. A test bucket will typically contain multiple test groups.

Test type:

- Non-intrusive tests are status check programs that ensure that all the hardware is configured as expected and is not missing any critical components. These nonintrusive tests should be run at system install, upgrade and periodically to ensure that there are no unexpected system changes. These tests can run in parallel with user jobs. Health check tests are non-intrusive.
- Intrusive tests need to run on a dedicated system. They cannot run in parallel with user jobs and should been properly marked and not used for scheduled jobs. It is strongly recommended that these tests run at initial system installation and whenever there is a

system change, such as a software or hardware upgrade. Tests that affect performance here are considered “intrusive”. Tests that might affect a user job’s performance on the nodes are also considered intrusive.

Test category:

- 1st pass: Used to find suspicious hardware or configuration. Provides a wide coverage to detect hardware degradation. In general, the first-pass tests have short durations, simple configuration and light resource usage.
- 2nd pass: Used for in-depth analysis. The second pass tests are not intended to be used on a regular basis because they may take longer than the first-pass tests with more complex configuration.

CSM Diagnostic framework integrates the following tests:

NOTES:

- The tests scripts might need to be customized. We assume default directories for tests that invoke scripts/binaries installed by third party rpms, that might not match the user installation.
- Some configuration variables need to be modified according to the customer requirements.
- For Spectrum MPI tests: daxpy, dgemm, dgemm_gpu and jlink, please make the appropriate changes as instructed by the README files in /opt/ibm/spectrum_mpi/healthcheck/<test>/README.<test>.
- Currently running Diagnostic’s dgemm, dgemm-gpu and jlink tests via LSF are not supported.

Test name	**Coverage**	**Comments**
chk-cpu	CPU	Displays cpu information: smt, scaling governow, cores, numa. Also checks the number of cpus, cores online, clock frequency.
chk-cpu-count	CPU	Check the node’s number of

		cpus, using lscpu command.
chk-csm-health	System	Provides a report of the status of the mater, aggregator(s), compute node(s) daemons, database connection, network connection.
chk-gpfs-mount	Storage	Check gpfs filesystem/mount point.
chk-hca-attributes	IB	Shows a quick overview about the existing host channel adapter (HCA) in the cluster: vendor, model, firmware version, port information and connection speed.
chk-ib-pcispeed	IB	Checks the IB adapters speed/width.
chk-led	Node	Identifies the fault LED on the node.
chk-mlnx-pci	IB	Checks Mellanox PCI configuration.
chk-nvidia-clocks	GPU	Checks nvidia clocks.
chk-nvidia-smi	GPU	Issues nvidia-smi command to check if there is a possible stuck nvidia-smi process.
chk-nvidia-vbios	GPU	Checks if the cpus have the expected type, bios version and bus id.
chk-nvme-mount	Storage	Checks if nvme device is not mounted or not found.
chk-nvme	Storage	Checks nvme device vendor and firmware level.
chk-os	Node	Checks operating system and kernel version on a node.
chk-sys-firmware	Node	Checks firmware levels.
chk-zombies	system	Checks possible stuck zombie tasks.
daxpy	Memory	Test and measures aggregate memory bandwidth. (should run as non-root)
dcgm-diag	GPU	Data Center GPU management. Extensive check to find potential power and/or performance

		problems. It supports the 3 levels, passed as args in test.properties: 1 - Quick System Validation (~ seconds) 2 - Extended System Validation (~ 2 minutes) 3 - System HW Diagnostics (~ 15 minutes) * starts dcgm daemon,if not running.
dcgm-health	GPU	Simple test to check if GPUs are healthy or not. * starts dcgm daemon, if not running.
dgemm	CPU	Test and measure CPU performance. (should run as non-root)
dgemm-gpu	GPU	Measures the GPU performance. (should run as non-root)
fieldiag	GPU	Nvidia fieldiag hw diagnostic.
gpu-health	GPU	Check GPU memory bandwidth, gpu dgemm flops and nvlink transfer speeds. NOTE: The source code for this test is shipped in hcidag/samples and should be compiled by the customer.
hxecache	L1/L2/L3 cache	L2/L2/L3 cache stress tests.
hxecpu	Generic processor core features	Processor core test.
hxecpu_pass2	Generic processor core features	Processor core test pass2.
hxediag_eth	Network	Exerciser for Ethernet adapters.

hxediag_ib	Network	Exerciser for IB adapters.
hxeewm_pass2	Processor	Thermal stress test.
hxefabricbus_pass2	Processor	Processor fabric bus test: inter node and intra node fabric busses.
hxefpu64	floating point	VSU (Vector Scalar Unit) test.
hxefpu64_pass2	floating point	VSU (Vector Scalar Unit) test pass2.
hxemem64	Memory subsystem	Memory subsystem stress test.
hxemem64_pass2	Memory subsystem	Memory subsystem stress test pass2.
hxenvidia	GPU	Exerciser for Nvidia GPUs in IBM Power System.
hxenvidia_pass2	GPU	Exerciser for Nvidia GPUs in IBM Power System pass2.
hxerng_pass2	Random number generator Engine	Stress random number generator engine.
hxesctu_pass2	Cache	Test processor cache coherency.
hxestorage_sd	Storage	HTX disk test.
hxestorage_sd_pass2	Storage	HTX pass2 disk test.
hxestorage_nvme	Storage	HTX nvme test.
hxestorage_nvme_pass2	Storage	HTX pass2 nvme test.
ipoib	IB	Shows the state, maximum transmission unit (MTU), and mode of the configured IP over Infiniband (IPoIB) devices on the node.
jlink	IB	Tests bandwidth for all possible node pairings and report bad or low performing links. (should run as non-root)
nvvs	GPU	GPU diagnostics, will be

		replaced by dcgm-diag. Similar to dcgm-diag, it supports the 3 levels, passed as args in test.properties: 1 - Quick System Validation (~ seconds) 2 - Extended System Validation (~ 2 minutes) 3 - System HW Diagnostics (~ 15 minutes) *nvvs will be discontinued. It is in this release as a support tool
p2pBandwidthLatencyTest	GPU	Checks the CUDA Peer-To-Peer (P2P) data transfers between pairs of GPUs and computes latency and bandwidth. NOTE: The source code for this test is shipped as part of the CUDA samples and should be compiled by the customer.
ppping	IPoIB	Check if IPoIB devices are configured correctly and reply to local ping requests.
rpower	Node	Check node's current power state/status.
rvitals	Hardware	Checks the hardware vital information against the information in clustconf.yaml. Vitals information are: temperature, voltage, wattage, fans speed, power and altitude
switch-inv	IB	Shows list of installed modules the switches, together with their part and serial number.
switch-module	IB	Shows list of different modules that are installed in the switches.

temp	Node	Checks all temperature sensors on a node. It uses the lm-sensors package.
test_memsize	Memory	Tests if memory size matches the size passed as argument.
test_simple	Diagnostic infrastructure	A very simple hello test.

7.4.1 **IBM Spectrum MPI Healthcheck**

The ibm-smpi package installs dgemm, dgemm_gpu, daxpy and jlink tests.
The executables are located under /opt/ibm/spectrum_mpi/healthcheck/<test>.
For more details of each test, please refer to the readme files under /opt/ibm/spectrum_mpi/healthcheck/<test>/README.<test>.

dgemm test requires IBM ESSL (IBM Engineering and Scientific Subroutine Library) and IBM XL Fortran installed in addition to spectrum_mpi.

7.4.2 **HTX tests**

HTX (Hardware Test Executive) is a system level test suite for validating pServer hardware.
HTX is now open and source is available at https://github.com/open-power/HTX
Most of the HTX tests starts with the prefix "hxe".
HTX package requires net-tools (ifconfig command), mesa-libGLU-devel and mesa-libGLU packages.

Supported HTX: version/build 475.
Obtain the rpm file from IBM and install on all nodes.
HTX installation creates an entry in the /etc/init.d directory that should be removed, we don't want the htx daemon to start on the node, it will be started on demand.
Also, the installation process starts the htxd daemon automatically, it should be stopped.

The commands bellow assume that you are installing from the management node using xcat utilities.

```
$ xdsh <nodes> rpm –ivh htxrhel72le-475-LE.ppc64le.rpm
$ xdsh <nodes> rm /etc/init.d/htx.d
$ xdsh <nodes> /usr/lpp/htx/etc/scripts/htxd_shutdown
```

7.4.3 **GPU**

The diagnostic system will support GPU diagnostic via DCGM (NVIDIA Data Center GPU Manager). Obtain the rpm from NVIDIA and install datacenter on all nodes. The commands bellow assume that you are installing from the management node using xcat utilities:

```
$ xdsh <nodes> rpm -ivh datacenter-gpu-manager-1.3.3-1.ppc64le.rpm
```

For more details on the installation and pre-requisites refer to the CSM Installation and Configuration document.

7.5 Security

xCAT framework supports SSL to secure the communication messages among Login node, Management Node and Service Node, it also supports to use the SSH as the secure remote shell to execute command on remote nodes.

By default, only user "root" on the management node has xCAT privileges and it can run Diagnostic. Because some tests require to be run as non-root user, and for security reasons, we strongly recommend to create a user for Diagnostic. In this document, this user it will be referenced as *diagadmin*.

A non-root user can be configured to run Diagnostics, by following "Granting users xCAT privileges" page: https://sourceforge.net/p/xCAT/wiki/Granting_Users_xCAT_privileges/ -setup-sudo-for-xCAT-user

These are the main steps to setup a new user to run Diagnostic:

- Creates a Linux user, *diagadmin,* in all nodes of the system.
- *diagadmin* user has to be in the *csm_api.acl* file (it is located in the **"/etc/ibm/csm/csm_api.acl"**), either in the "*privileged_user_id*" list or in a group that is listed in "*privileged_group_id*".
  In this document, we assume that the *diagadmin* is part of a group listed in the acl file as "*privileged_group_id*", and it will be referenced as *csmadmin*.
- Include *diagadmin* to the xCAT policy table, to allow *user* to run xCAT commands (see the "Granting users xCAT privileges" document.
  The user has to be able to issue the following commands: xdcp, nodestat, xdsh. This is specified in the police table. An example of a user that can issue all xcat commands in the police table is:

```
"7.0","diagadmin",,,,,,"allow",,
```

- Create SSL certificates for user *diagadmin.*
- Setup sudo privileges for user *diagadmin*, by creating the file diagadmin in /etc/sudoers.d directory with the following contents:

```
diagadmin        ALL=(ALL)       NOPASSWD: ALL
```

  Or adding the following line in /etc/sudoers file:

```
%diagadmin        ALL=(ALL)       NOPASSWD: ALL
```

  Make sure sudo package is installed on all nodes.
- Disable the requiretty when issuing sudo. Comment the line "Defaults requiretty" in the sudoers file if you have it. Default is do not requiretty

- The management node must be defined in the xCAT database in order to run tests on the management node, tests with "*targetType = Management*" in the test.properties file (chk-sys-firmware, ppping, chk-csm-health. This can be done with the command: *xcatconfig –m*

7.6 User Interface

- *hcdiag_run.py* : command line to run Diagnostic tests
  For details:
  hcdiag_run.py –help

- *hcdiag_query.py* : command line to query Diagnostic run and results
  For details:
  hcdiag_quey.py --help

7.7 Validation

To validate the basic diagnostic installation, run test_simple test:

/opt/ibm/csm/bin/hcdiag_run.py –test test_simple –target <noderange>

8 CSM REST Daemon

8.1 Overview

The CSM REST daemon (csmrestd) is optional and is not required for normal cluster operation. Csmrestd is used to enable RAS events to be created from servers that do not run CSM infrastructure daemons. The CSM REST daemon can be installed on the service nodes to allow BMC RAS events to be reported by the IBM POWER LC Cluster RAS Service (ibm-crassd).

8.2 Packaging and Installation

All required binaries, configuration files, and example scripts are packaged in **ibm-csm-restd-1.0.0-*.ppc64le.rpm**.

To install and configure csmrestd, please refer to the *CSM Installation and Configuration Guide*.

8.3 Creating a CSM RAS event via the REST API

Once csmrestd is installed and configured on the management node, an example RAS event can be created from any server that can reach the csmrestd server ip address. This test can be run locally on the management node or from any other node.

By default the CSM RAS msg type settings for the event created by the create_node_leave_event.sh example script will not impact the cluster. However, this event may be fatal in future releases. In future releases, this test may cause a node to be removed from the list of nodes ready to run jobs.

On a service node:

Copy the sample script from */opt/ibm/csm/share/rest_scripts/spectrum_scale* to some other location for editing.

```
$ cp /opt/ibm/csm/share/rest_scripts/spectrum_scale/create_node_leave_event.sh ~/
```

Edit the copy of create_node_leave_event.sh and replace "__CSMRESTD_IP__" with the IP address that was configured for csmrestd to listen on in */etc/ibm/csm/csmrestd.cfg*. Optionally, the LOCATION_NAME can also be modified to refer to a real node_name in the csm_node table.

Start the local CSM daemon, then csmrestd if either of them are not currently running.

```
$ systemctl start csmd-aggregator
$ systemctl start csmrestd
```

Run the example script and observe a new event get created in */var/log/ibm/csm/csm_ras_events.log*:

```
$ ~/create_node_leave_event.sh
$ cat /var/log/ibm/csm/csm_ras_events.log | grep spectrumscale.node.nodeLeave

Example output:
{"time_stamp":"2017-04-25 09:48:37.829407","msg_id":"spectrumscale.node.nodeLeave",
"location_name":"c931f04p08vm03","raw_data":"","ctxid":"9","min_time_in_pool":"1",
"suppress_ids":"","severity":"WARNING","message":"c931f04p08-vm03 has left the
cluster.","decoder":"none","control_action":"NONE","description":"The specified node has left
the cluster.","relevant_diags":"NONE","threshold_count":"1","threshold_period":"0"}
```

Stop csmrestd:

```
$ systemctl stop csmrestd
```