

SIEMENS EDA

# Calibre® Cluster Manager (CalCM) User's Manual

Software Version 2021.2

SIEMENS

Unpublished work. © 2021 Siemens

This material contains trade secrets or otherwise confidential information owned by Siemens Industry Software, Inc., its subsidiaries or its affiliates (collectively, "Siemens"), or its licensors. Access to and use of this information is strictly limited as set forth in Customer's applicable agreement with Siemens. This material may not be copied, distributed, or otherwise disclosed outside of Customer's facilities without the express written permission of Siemens, and may not be used in any way not expressly authorized by Siemens.

This document is for information and instruction purposes. Siemens reserves the right to make changes in specifications and other information contained in this publication without prior notice, and the reader should, in all cases, consult Siemens to determine whether any changes have been made. Siemens disclaims all warranties with respect to this document including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement of intellectual property.

The terms and conditions governing the sale and licensing of Siemens products are set forth in written agreements between Siemens and its customers. Siemens' **End User License Agreement** may be viewed at: [www.plm.automation.siemens.com/global/en/legal/online-terms/index.html](http://www.plm.automation.siemens.com/global/en/legal/online-terms/index.html).

No representation or other affirmation of fact contained in this publication shall be deemed to be a warranty or give rise to any liability of Siemens whatsoever.

**TRADEMARKS:** The trademarks, logos, and service marks ("Marks") used herein are the property of Siemens or other parties. No one is permitted to use these Marks without the prior written consent of Siemens or the owner of the Marks, as applicable. The use herein of third party Marks is not an attempt to indicate Siemens as a source of a product, but is intended to indicate a product from, or associated with, a particular third party. A list of Siemens' trademarks may be viewed at: [www.plm.automation.siemens.com/global/en/legal/trademarks.html](http://www.plm.automation.siemens.com/global/en/legal/trademarks.html). The registered trademark Linux® is used pursuant to a sublicense from LMI, the exclusive licensee of Linus Torvalds, owner of the mark on a world-wide basis.

Support Center: [support.sw.siemens.com](http://support.sw.siemens.com)

Send Feedback on Documentation: [support.sw.siemens.com/doc\\_feedback\\_form](http://support.sw.siemens.com/doc_feedback_form)

# Table of Contents

---

<b>Chapter 1</b>	
<b>Introduction to Calibre Cluster Manager (CalCM) .....</b>	<b>15</b>
Product Overview .....	16
Benefits of CalCM .....	16
Audience .....	16
CalCM Flow .....	17
CalCM Terminology .....	18
Product Requirements .....	20
Hardware and Networking .....	20
CalCM Daemon .....	20
CalCM Licenses .....	20
Calibre Product Licenses .....	20
CalCM Configuration Files .....	21
Environment Variables .....	21
Modes of Operation .....	22
Syntax Conventions .....	22
<b>Chapter 2</b>	
<b>CalCM Concepts.....</b>	<b>25</b>
CalCM Daemon .....	25
CalCM Configuration Files .....	26
CalCM Applications .....	30
CalCM and Licensing .....	31
CalCM Network Monitor (Nemo) .....	34
CalCM Resource Management .....	35
CalCM Resource Monitoring .....	37
CalCM TAT .....	37
<b>Chapter 3</b>	
<b>Deploying and Running CalCM .....</b>	<b>45</b>
Configuring the CalCM Environment .....	45
Starting the CalCM Daemon .....	49
Configuring Calibre Jobs to Run Under CalCM .....	51
Using the CalCM Dashboard Web Application .....	54
CalCM Dashboard Requirements .....	54
Executing Calibre Jobs to Run Under CalCM with the Dashboard .....	55
Monitoring CalCM Notifications with the Dashboard .....	61
Monitoring CalCM Resources with the Dashboard .....	64
Monitoring Jobs Running Under CalCM with the Dashboard .....	66
Monitoring Resource Usage and Allocation with the Dashboard .....	71
Displaying System Information in the CalCM Dashboard .....	75
Viewing Chart Information in the CalCM Dashboard .....	78

Comparing Job Information in the CalCM Dashboard . . . . .	80
Refreshing Chart Data in the CalCM Dashboard . . . . .	85
Accessing and Updating Settings in the CalCM Dashboard . . . . .	87
CalCM+ Advanced Features for Data Analysis in the CalCM Dashboard . . . . .	92
Using Job Statistics for Data Analysis . . . . .	92
Using Cluster Utilization Information for Data Analysis. . . . .	98
CalCM+ Advanced Features for Accessing CalScope in the CalCM Dashboard . . . . .	103
Running CalCM with a Backup Daemon . . . . .	106

**Chapter 4****CalCM Command Line and Tcl Application Reference Dictionary . . . . .** **109**

CalCM Environment Variables . . . . .	110
CALCMD_CONF_PATH . . . . .	111
CALCMD_LOGLEVEL . . . . .	112
CALCMD_LOG_TIMESTAMPS . . . . .	113
CALCMD_REMOTE_COMMAND_DYNAMIC_LOG_LEVEL . . . . .	114
CALIBRE_ENABLE_SETLAYER_ECP . . . . .	119
CALIBRE_ENABLE_SETLAYER_ERT . . . . .	120
CALIBRE_LBD_LOG_TIMESTAMPS . . . . .	121
CalCM Command Line Reference . . . . .	122
calcmd . . . . .	123
calcml_kill_job . . . . .	126
calcml_send_message . . . . .	127
calcml_submit_job . . . . .	129
Messaging Commands . . . . .	130
adjust_cluster . . . . .	131
adjust_maxchange . . . . .	133
adjust_minmax . . . . .	134
adjust_quota . . . . .	135
adjust_rampupdown . . . . .	136
build_jobanalysis . . . . .	137
check_cluster_limit . . . . .	138
close_node . . . . .	139
log_level . . . . .	140
log_timestamp . . . . .	141
open_node . . . . .	142
prioritize_job . . . . .	143
queue_status . . . . .	144
read_conf . . . . .	146
register_node . . . . .	147
renew_license . . . . .	148
reset_jobanalysis . . . . .	150
reset_license . . . . .	151
reset_notification . . . . .	153
reset_quota . . . . .	154
resume_job . . . . .	155
set_budget . . . . .	156
set_priority_bump . . . . .	157

## Table of Contents

---

span_log .....	158
suspend_job .....	159
unregister_node .....	160
CalCM Tcl Application Reference .....	161
calcml_hmonitor_app.tcl .....	162
calcml_http_server_app.tcl .....	166
calcml_jobanalysis_app.tcl .....	169
calcml_jobqueue_app.tcl .....	174
calcml_notification_app.tcl .....	182
calcml_rmanager_app.tcl .....	195
calcml_rmonitor_app.tcl .....	200
calcml_systemanalysis_app.tcl .....	203
calcml_tat_app.tcl .....	205
calcml_tcl_app.tcl .....	210
 <b>Chapter 5</b>	
<b>CalCM Configuration File Reference Dictionary .....</b>	<b>211</b>
CalCM Configuration File Reference .....	212
APPLICATION .....	213
DATABASE FLAG .....	214
LICENSE FEATURES .....	215
LICENSE PORT .....	217
LICENSE SERVER .....	218
SERVER INTERVAL .....	219
SERVER PORT .....	220
SERVER TIMEOUT .....	221
VARIABLE .....	222
CalCM Cluster Configuration File Reference .....	223
CLUSTER NAME .....	224
MASTER HOST .....	225
RDS HOST .....	226
REMOTE HOST .....	227
CalCM Job Configuration File Reference .....	228
JOB BUDGET .....	231
JOB DIRECTORY .....	234
JOB ENV .....	235
JOB INFO .....	236
JOB KEY .....	239
JOB LOG .....	240
JOB MODE .....	241
JOB MTFLEX .....	242
JOB MTFLEX_OVERRIDE .....	243
JOB NOTIFICATION .....	244
JOB NOTIFICATION_FILTERS .....	245
JOB NOTIFICATION_TRACKERS .....	246
JOB POST_EXEC/JOB POST_EXEC_CHECK .....	247
JOB PRE_EXEC/JOB PRE_EXEC_CHECK .....	248
JOB PRIORITY .....	249

---

JOB QUOTA .....	250
JOB RULE .....	251
JOB SLOT .....	252
JOB TAT_PRIORITY_BUMP.....	253
JOB TOTAL .....	254
JOB USER .....	255
LAUNCH DELAY.....	256
LAUNCH MASTER_NAME.....	257
LAUNCH MAXCOUNT .....	258
LAUNCH SMTFACTOR.....	259
LAUNCH WAIT .....	260
MASTER PREFERRED.....	261
MASTER RESOURCE .....	262
REMOTE COUNT .....	263
REMOTE COUNT_MAX .....	264
REMOTE COUNT_MIN .....	265
REMOTE DATA .....	266
REMOTE DATA_MIN .....	267
REMOTE DATA_RECOVEROFF .....	268
REMOTE DATA_RESOURCE .....	269
REMOTE ENV .....	270
REMOTE INIT_RESOURCE .....	271
REMOTE MAX .....	272
REMOTE MAXCHANGE.....	273
REMOTE MIN .....	275
REMOTE MINMAX .....	276
REMOTE PREFERRED.....	279
REMOTE RAMPUPDOWN .....	280
REMOTE RESOURCE .....	282
<b>Appendix A</b>	
<b>CalCM Patch Requirements .....</b>	<b>285</b>
Calibre Release and Patch Requirements for CalCM .....	285
<b>Appendix B</b>	
<b>Example Configuration Files and Transcript .....</b>	<b>289</b>
CalCM Configuration File Example .....	289
Job Configuration File Example .....	291
CalCM Daemon Transcript Example .....	291
<b>Appendix C</b>	
<b>CalCM Migration Best Practices.....</b>	<b>297</b>
Introduction .....	297
CalCM Users .....	297
Overview of Migration Best Practices.....	298
Migration Tasks .....	300
Preparing for CalCM Daemon Migration.....	300
Starting Up and Qualifying the New Version of CalCM .....	302

## Table of Contents

---

Migrating CalCM .....	304
<b>Appendix D</b>	
<b>Troubleshooting CalCM. ....</b>	<b>307</b>
CalCM Test Environment Set Up .....	307
<b>Appendix E</b>	
<b>Developing CalCM Applications.....</b>	<b>309</b>
Tcl Basics.....	309
CalCM Application Development Overview .....	310
Creating an Activity Log Application .....	318
Setting Up a Notification Application .....	322
<b>Appendix F</b>	
<b>CalCM Nemo API Reference Dictionary .....</b>	<b>325</b>
Cluster Snapshot .....	325
Cluster Snapshot Object Types .....	326
CalCM Nemo API Reference .....	328
nemo::alter .....	329
nemo::application .....	330
nemo::calibre .....	332
nemo::control .....	333
nemo::filesystem.....	336
nemo::host.....	338
nemo::interface.....	341
nemo::job .....	342
nemo::master.....	345
nemo::message .....	348
nemo::node .....	350
nemo::object .....	354
nemo::process .....	355
nemo::remote .....	357
nemo::remotedata .....	359
nemo::singleton .....	361
nemo::snapshot .....	362
<b>Appendix G</b>	
<b>Configuring and Using the CalCM TAT Application.....</b>	<b>363</b>
CalCM TAT Workflow .....	363
Configuring the TAT Application to Run with CalCM.....	364
Starting the CalCM Daemon .....	366
Specifying a Budget for Calibre Job Operations .....	367
Executing Calibre Jobs with the TAT Application .....	369
Dynamically Adjusting the Budget and Priority .....	372
Using Notifications with the TAT Application .....	373

---

<b>Appendix H</b>	
<b>CalCM Supported Platforms.....</b>	<b>377</b>
Customizing CalCM for Remote Shell .....	378
Remote Shell Considerations in CalCM.....	378
Specifying the RSH Remote Shell in CalCM.....	378
Setting Up SSH Connectivity Without Password Authentication .....	379
Platform Tool Job and Remotes Control .....	382
Using LSF to Control Jobs Only.....	382
Using LSF to Control Jobs and Remotes .....	383
Using Grid to Control Jobs Only .....	384
Using Grid to Control Jobs and Remotes .....	385
Using MySQL in CalCM.....	386
Upgrading the CalCM Database .....	387
Implementing convert_database .....	387
convert_database.....	388
Schema File Example .....	390
<b>Appendix I</b>	
<b>CalScope Usage and Reference .....</b>	<b>391</b>
Introduction to Calibre System Monitoring Solution (CalScope).....	392
CalScope Overview .....	392
CalScope Flow .....	392
CalScope Key Concepts .....	393
CalScope Requirements .....	394
CalScope Modes of Operation .....	395
Running CalScope .....	396
Setting Up Database Access .....	396
CalScope Environment .....	400
Configuring the CalScope Environment .....	400
CalScope Example Configuration File .....	402
Invoking CalScope .....	404
Interacting With CalScope Monitored Hosts .....	404
Importing Job Information With CalScope .....	406
Viewing Job Information in the CalScope Dashboard .....	407
Comparing Job Information in the CalScope Dashboard.....	413
CalScope Command Reference .....	420
calscope.....	421

## Index

## Third-Party Information

# List of Figures

---

Figure 1-1. Interaction of CalCM with Calibre Jobs .....	18
Figure 2-1. CalCM Configuration Files .....	26
Figure 2-2. Licensing Flow for Calibre Jobs Managed By CalCM .....	33
Figure 2-3. CalCM Process Flow .....	34
Figure 2-4. Comparison of Runtime Modeling Job With and Without TAT .....	37
Figure 2-5. CalCM Dynamic Resource Allocation and TAT Control .....	39
Figure 2-6. Dynamic Budget Adjustment .....	40
Figure 2-7. Allocating Resource with CalCM TAT .....	41
Figure 3-1. Recommended CalCM Directory Structure .....	46
Figure 3-2. CalCM Directory Structure .....	51
Figure 3-3. Example CalCM Job Configuration File .....	52
Figure 3-4. CalCM Directory Structure .....	53
Figure 3-5. Active Jobs .....	56
Figure 3-6. Job Detail Plot .....	57
Figure 3-7. Job Action Filter .....	58
Figure 3-8. Search Window .....	58
Figure 3-9. Reorder or Hide Columns Window .....	59
Figure 3-10. Pending Jobs .....	60
Figure 3-11. Finished Jobs .....	60
Figure 3-12. Finished Job Information .....	60
Figure 3-13. Notifications — Job ID Filtered Messages .....	62
Figure 3-14. Notifications — Job ID Resource Plot and Information .....	63
Figure 3-15. Master (Primary) Nodes Registered Filter .....	64
Figure 3-16. Remote Nodes Flag Column .....	65
Figure 3-17. Master (Primary) Nodes Host Column .....	65
Figure 3-18. Host Information .....	66
Figure 3-19. Resource Utilization by Department .....	67
Figure 3-20. Resource Utilization by Cost Center .....	68
Figure 3-21. License Utilization by Department .....	69
Figure 3-22. Job Duration .....	70
Figure 3-23. CPU Allocation by Job .....	72
Figure 3-24. Job Detail Information .....	73
Figure 3-25. CPU Allocation by Quota .....	74
Figure 3-26. License Usage .....	74
Figure 3-27. System Utilization Information .....	76
Figure 3-28. Report Filters .....	76
Figure 3-29. CalCM Applications Information .....	77
Figure 3-30. CalCM Dashboard Version Information .....	77
Figure 3-31. Job Detail Page and Chart Menu .....	79
Figure 3-32. Active Jobs Record Selection and Compare .....	81

Figure 3-33. Compare Jobs Page Functions .....	82
Figure 3-34. Select Baseline Job for Comparison .....	83
Figure 3-35. Selected Job Records for Comparison .....	85
Figure 3-36. Compared Job Plots .....	86
Figure 3-37. Refresh Chart Data Functions .....	87
Figure 3-38. Settings Profile Timezone .....	88
Figure 3-39. Settings Add New User .....	89
Figure 3-40. Settings Groups .....	89
Figure 3-41. Settings Add New Group .....	90
Figure 3-42. Settings User Activity .....	91
Figure 3-43. Job Statistics Page .....	94
Figure 3-44. Job Statistics Period .....	94
Figure 3-45. Job Statistics Filters .....	95
Figure 3-46. Job Statistics Plot .....	95
Figure 3-47. Job Statistics Data Point Value .....	96
Figure 3-48. Job Statistics Data Point Plot .....	96
Figure 3-49. Job Statistics Filtered Jobs Table .....	96
Figure 3-50. Job Statistics Cluster Centers .....	97
Figure 3-51. Job Statistics Filter by Keyword Version (1) .....	97
Figure 3-52. Job Statistics Filter By Keyword Version (2) .....	98
Figure 3-53. Cluster Utilization Plot .....	100
Figure 3-54. Cluster Utilization Plot Detail .....	100
Figure 3-55. Cluster Utilization Top Ten Remote Usage .....	101
Figure 3-56. Cluster Utilization Top Ten Remote Usage Detail .....	101
Figure 3-57. Viewing Second Criteria Distribution With Two Aggregates .....	102
Figure 3-58. Logview/Monitor Host Job List .....	104
Figure 3-59. Accessing CalScope From the Nodes Page .....	105
Figure 3-60. Accessing CalScope From the Node Details Page .....	105
Figure 3-61. Accessing CalScope From the Job Details Notifications Table .....	106
Figure C-1. Recommended Directory Structure for CalCM .....	298
Figure C-2. CalCM Files Needed for the Migration .....	301
Figure E-1. Defining Event Subcommands in the nemo::application Statement .....	312
Figure E-2. Application Initialization .....	313
Figure E-3. Cluster State .....	314
Figure E-4. Message Processing .....	315
Figure E-5. Notification in CalCM Dashboard Job Detail Plot .....	324
Figure E-6. Notifications Table in CalCM Dashboard .....	324
Figure F-1. Cluster Snapshot .....	326
Figure F-2. Hierarchy of Object Types in Cluster Snapshot .....	327
Figure G-1. Workflow for Running TAT with CalCM .....	364
Figure G-2. CalCM Dashboard Web Application .....	367
Figure G-3. CalCM Job Configuration File .....	368
Figure G-4. Active Jobs View .....	371
Figure G-5. Job Detail View .....	371
Figure I-1. CalScope Flow .....	393

## List of Figures

---

Figure I-2. CalScope LogView/Monitor Host Job List . . . . .	407
Figure I-3. Job Operations Table . . . . .	408
Figure I-4. Job Events Chart . . . . .	408
Figure I-5. Job Plot Metrics . . . . .	409
Figure I-6. Job Events Detail . . . . .	409
Figure I-7. Host Alert Information . . . . .	410
Figure I-8. Monitor Host Plots for Event . . . . .	410
Figure I-9. Job Plot Metric Detail . . . . .	411
Figure I-10. Job Add Plots Window . . . . .	411
Figure I-11. Job Detail Global Options . . . . .	412
Figure I-12. Job Detail Local Options . . . . .	412
Figure I-13. CalScope Job Selection and Compare . . . . .	414
Figure I-14. CalScope Compare Jobs Page . . . . .	415
Figure I-15. Job Comparisons . . . . .	416
Figure I-16. Controlling Which Hosts to Display in Plots . . . . .	416
Figure I-17. Highlight an Operation . . . . .	418
Figure I-18. Mouseover the Chart . . . . .	419



# List of Tables

---

Table 1-1. Syntax Conventions .....	22
Table 2-1. Recognized Variables .....	29
Table 4-1. Description of Verbose Output from CALCMD_REMOTE_COMMAND_DYNAMIC_LOG_LEVEL .....	114
Table 4-2. Transcript Messages for calcm_kill_job .....	126
Table 4-3. Queue Status Bit-Fields .....	144
Table 5-1. Configuration File Commands .....	212
Table 5-2. Cluster Configuration Commands .....	223
Table 5-3. CalCM Job Configuration Commands .....	228
Table 5-4. Job Information Keywords .....	236
Table A-1. Patch Requirements for CalCM .....	285
Table E-1. Tcl Special Characters .....	309
Table F-1. Tcl Nemo API Commands .....	328
Table I-1. Global and Local Icon Descriptions .....	412



# Chapter 1

## Introduction to Calibre Cluster Manager (CalCM)

---

This section introduces Calibre® Cluster Manager (CalCM) and provides information on common CalCM terminology as well as product requirements.

<b>Product Overview .....</b>	<b>16</b>
<b>CalCM Flow .....</b>	<b>17</b>
<b>CalCM Terminology .....</b>	<b>18</b>
<b>Product Requirements.....</b>	<b>20</b>
<b>Modes of Operation .....</b>	<b>22</b>
<b>Syntax Conventions .....</b>	<b>22</b>

## Product Overview

---

Calibre Cluster Manager (CalCM) is a centralized service that collects pertinent information about currently running Calibre jobs, as well as the cluster and network on which these jobs are run.

<b>Benefits of CalCM .....</b>	<b>16</b>
<b>Audience .....</b>	<b>16</b>

## Benefits of CalCM

The benefits of CalCM include a series of applications that use the collected data to perform resource management and monitoring of Calibre jobs.

- Dynamic resource management, by allocating cluster nodes and licenses according to the CPU demand of running jobs, their priorities, and available hardware resources.
- Activity logging, by examining important job, hardware, and network states and saving this information in a file for subsequent analysis.
- Visual monitoring, by providing an interface for viewing and analyzing the state of cluster resources and running jobs.
- Job queueing, by providing a single point for submitting jobs for execution on the cluster without having to manually configure Calibre MTflex runs.
- Targeted job completion, using the CalCM TAT (Turn Around Time) application to specify a targeted completion time for a Calibre job to run through to completion.

## Audience

This section identifies the different roles and responsibilities associated with administering and using CalCM.

CalCM roles and responsibilities include:

- **CalCM Administrator** — Responsibilities include one or more of the following: installing the Calibre software, setting up network access, or configuring tool access. The CalCM administrator is typically responsible for configuring, launching, supporting, and monitoring the CalCM daemon.

This role assumes a basic understanding of Tcl and SQLite<sup>®</sup><sup>1</sup> for Tcl programming skills, in addition to networking concepts and knowledge of the existing network configuration.

---

1. SQLite<sup>®</sup> is a registered trademark of Hipp, Wyrick & Company, Inc.

- Calibre End Users — Responsibilities include configuring and executing Calibre jobs. This role assumes an understanding of the different modes for submitting batch Calibre jobs and their requirements, and also includes an understanding of the CalCM Administrator's process for launching Calibre jobs.

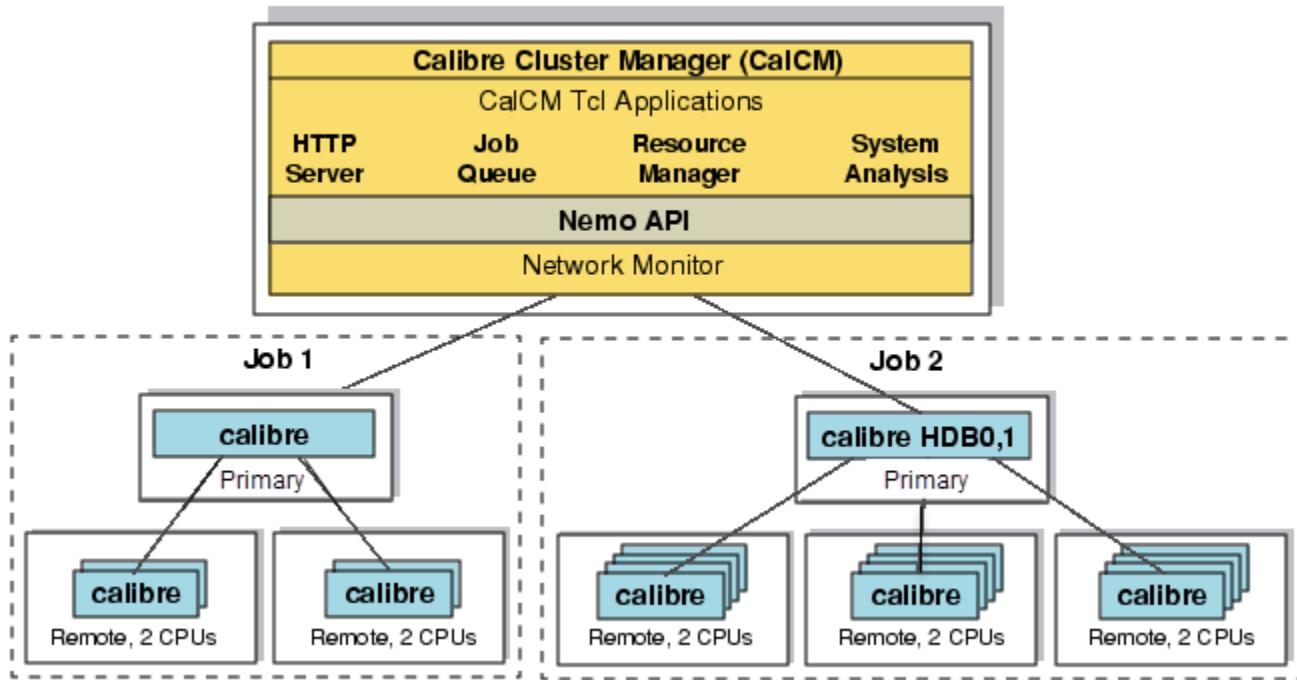
## CalCM Flow

The CalCM flow includes several tasks associated with each job, such as monitoring, analyzing, communicating, updating, and dispatching. The network monitoring functionality in CalCM communicates with Calibre jobs running on the cluster and updates the status of the Calibre jobs at regular intervals.

[Figure 1-1](#) illustrates how CalCM is used to dispatch jobs (Job 1 and Job 2 in this example) to different primary hosts that are networked to different remote hosts. CalCM includes a set of Tcl applications that work with the network monitoring functionality to provide the following capabilities:

- Provides the ability to query hardware resources, run Calibre jobs, track associated nodes, servers, running threads, and job progress.
- Allows you to monitor the job queue and automatically dispatch jobs based on submission priority.
- Automatically adds or removes CPUs based on job demand and performs contention resolution.
- Writes top-level cluster utilization statistics into a log file.
- Calculates the actual quota of resources (CPU).

**Figure 1-1. Interaction of CalCM with Calibre Jobs**



## Related Topics

[CalCM Applications](#)

[CalCM Tcl Application Reference](#)

## CalCM Terminology

This section includes definitions for some key terms that are used when discussing CalCM.

- **Cluster** — A group of linked computers connected to each other through a local area network.
- **ECP** — Estimated completion percentage.
- **ERT** — Estimated remaining time.
- **HDB** — Pseudo Hierarchical Databases (HDBs) are created when you run a Calibre job in hyperscaling (-hyper) or in hyper remote (-hyper remote) mode. Hyper remote mode reduces the primary host memory requirements and improves Calibre MTflex performance.
- **LBD** — The Calibre License Broker Daemon (LBD) manages licenses during a Calibre run. Refer to “[CalCM and Licensing](#)” for more information.
- **Primary** — The primary is the host machine from which you invoke a Calibre job. The primary host launches and connects to remote hosts and handles tasks not assigned to the

remote hosts. The primary host is sometimes referred to in commands, environment variables, scripts, and transcripts as a “local” or “master” host.

- **Remote** — The remote hosts run a Calibre task that is assigned by the Calibre job running on the primary host. The remote hosts must be able to access the CALIBRE\_HOME tree for the respective platform. For environments using multiple platforms, the remote hosts may need to access different CALIBRE\_HOME trees for their respective platforms.

See “[Glossary](#)” in the *Calibre Administrator’s Guide* for complete descriptions of hardware and distributed processing terms.

# Product Requirements

This section provides information on the requirements for running CalCM.

<b>Hardware and Networking</b> .....	<b>20</b>
<b>CalCM Daemon</b> .....	<b>20</b>
<b>CalCM Licenses</b> .....	<b>20</b>
<b>Calibre Product Licenses</b> .....	<b>20</b>
<b>CalCM Configuration Files</b> .....	<b>21</b>
<b>Environment Variables</b> .....	<b>21</b>

## Hardware and Networking

All remote hosts must be able to communicate with all primary hosts and the machine hosting the CalCM daemon must be able to communicate with all primary and remote hosts.

## CalCM Daemon

You must run the 2011.4 or newer version of the CalCM daemon to access the TAT application related features.

## CalCM Licenses

The Calibre Cluster Manager license is required to run CalCM and the Calibre Cluster Manager Plus licenses are required to use the advanced features of CalCM+.

Refer to “[Calibre Cluster Manager \(CalCM\)](#)” and “[Calibre Cluster Manager Plus \(CalCM+\)](#)” in the *Calibre Administrator’s Guide* for licensing information for CalCM and CalCM+.

## Calibre Product Licenses

CalCM works in conjunction with the Calibre License Broker Daemon (LBD) to handle the licensing of Calibre jobs that run under CalCM.

As part of the configuration of the CalCM cluster, you must dedicate a set of Calibre product licenses to be used for jobs running under CalCM. Calibre jobs running under CalCM can only utilize licenses that have been allocated in this manner.

Refer to the *Calibre Administrator’s Guide* for licensing information for Calibre products.

## CalCM Configuration Files

A CalCM configuration file and cluster configuration file are required in order to run the CalCM daemon.

## Environment Variables

This section describes the environment variables that you use to configure the CalCM environment.

- **CALIBRE\_HOME**

You must define the CALIBRE\_HOME variable to the location of the Calibre software tree. Refer to the *Calibre Administrator's Guide* for information on using this variable.

- **CALCMD\_CONF\_PATH**

This is an optional variable that defines the location of the CalCM configuration file. You must specify this variable if this is how you want the daemon to locate the file.

- **CALIBRE\_ENABLE\_SETLAYER\_ECP**

This is a required variable if you are using TAT application related features. This variable forces controllable SVRF operations to provide TAT with the information necessary to make resource allocation decisions accordingly. You must set this variable to “2” to enable the Setlayer ERT feature.

- **CALIBRE\_ENABLE\_SETLAYER\_ERT**

This is a required variable if you are using TAT application related features. This variable forces controllable SVRF operations to provide TAT with the information necessary to make resource allocation decisions accordingly. You must set this variable to “2” to enable the Setlayer ERT feature.

- **CALCMD\_LOGLEVEL**

This is an optional variable used to control the level of information that is output to the transcript.

- **CALCMD\_LOG\_TIMESTAMPS**

This is an optional variable used to print timestamps for the CalCM daemon and the LBD in the transcript. You must set this variable to “1” to enable the timestamps printing feature.

- **CALIBRE\_LBD\_LOG\_TIMESTAMPS**

This is an optional variable used to control the timestamps printing feature for the LBD only. If CALCMD\_LOG\_TIMESTAMPS is enabled, you must set this variable to “0” to override the timestamps printing for the LBD.

## Related Topics

[Calibre Administrator's Guide \[Calibre Administrator's Guide\]](#)  
[CalCM and Licensing](#)  
[CalCM Configuration Files](#)  
[CALCMD\\_CONF\\_PATH](#)  
[CALIBRE\\_ENABLE\\_SETLAYER\\_ECP](#)  
[CALIBRE\\_ENABLE\\_SETLAYER\\_ERT](#)  
[CALCMD\\_LOGLEVEL](#)  
[CALCMD\\_LOG\\_TIMESTAMPS](#)  
[CALIBRE\\_LBD\\_LOG\\_TIMESTAMPS](#)

## Modes of Operation

You execute the CalCM daemon from the command line.

It is recommended that the daemon run as a background process. Your interactions with CalCM are through command line scripts you use to submit jobs and communicate with the jobs. CalCM includes a web application that you can use to monitor and interact with jobs.

## Related Topics

[CalCM Concepts](#)  
[Deploying and Running CalCM](#)

## Syntax Conventions

The command descriptions use font properties and several metacharacters to document the command syntax.

**Table 1-1. Syntax Conventions**

Convention	Description
<b>Bold</b>	Bold fonts indicate a required item.
<i>Italic</i>	Italic fonts indicate a user-supplied argument.
Monospace	Monospace fonts indicate a shell command, line of code, or URL. A bold monospace font identifies text you enter.
<u>Underline</u>	Underlining indicates either the default argument or the default value of an argument.

**Table 1-1. Syntax Conventions (cont.)**

Convention	Description
UPPercase	For certain case-insensitive commands, uppercase indicates the minimum keyword characters. In most cases, you may omit the lowercase letters and abbreviate the keyword.
[ ]	Brackets enclose optional arguments. Do not include the brackets when entering the command unless they are quoted.
{ }	Braces enclose arguments to show grouping. Do not include the braces when entering the command unless they are quoted.
“ ”	Quotes enclose metacharacters that are to be entered literally. Do not include single quotes when entering braces or brackets in a command.
or	Vertical bars indicate a choice between items. Do not include the bars when entering the command.
...	Three dots (an ellipsis) follows an argument or group of arguments that may appear more than once. Do not include the ellipsis when entering the command.
<b>Example:</b>	
<pre><b>DE</b>vice {<i>element_name</i> [‘(‘<i>model_name</i>‘)’]}     <i>device_layer</i> {<i>pin_layer</i> [‘(‘<i>pin_name</i>‘)’] ...}         [‘&lt;’<i>auxiliary_layer</i>‘&gt; ...]         [‘(‘<i>swap_list</i>‘)’ ...]     [<b>BY NET</b>   <b>BY SHAPE</b>]</pre>	



# Chapter 2

## CalCM Concepts

---

This section discusses several core concepts related to running CalCM.

<b>CalCM Daemon</b> .....	<b>25</b>
<b>CalCM Configuration Files</b> .....	<b>26</b>
<b>CalCM Applications</b> .....	<b>30</b>
<b>CalCM and Licensing</b> .....	<b>31</b>
<b>CalCM Network Monitor (Nemo)</b> .....	<b>34</b>
<b>CalCM Resource Management</b> .....	<b>35</b>
<b>CalCM Resource Monitoring</b> .....	<b>37</b>
<b>CalCM TAT</b> .....	<b>37</b>

## CalCM Daemon

The CalCM daemon runs as a background process collecting information about the jobs submitted under CalCM, as well as the cluster and network.

The following rules apply to running and shutting down the CalCM daemon:

- The CalCM configuration file (`calcmd.conf`) and cluster configuration file (`cluster.conf`) must exist in order to run CalCM.
- Multiple instances of the CalCM daemon can be started using one configuration file, but only the instance that successfully puts a file lock on the configuration file becomes the active daemon. The other daemons remain inactive and serve as backup in case the active daemon goes down.
- If the configuration file is a soft link, then the lock is put on the source file.
- The CalCM daemon should be shutdown using the “`kill -HUP calcmd_process_id`” command. This command kills both the CalCM daemon process and its associated LBD process. Refer to “[CalCM and Licensing](#)” for more information on the LBD process.

---

### Note

 When the primary CalCM daemon is killed, any Calibre jobs that are running will continue to run without interruption. If you implement a backup daemon, as described in “[Running CalCM with a Backup Daemon](#),” it can automatically start a new daemon to serve as the primary daemon and take over the running of the jobs.

---

When you start the daemon, it automatically creates the calcm\_send\_message and calcm\_submit\_job scripts. These scripts are used to submit and communicate with jobs running under CalCM.

## Related Topics

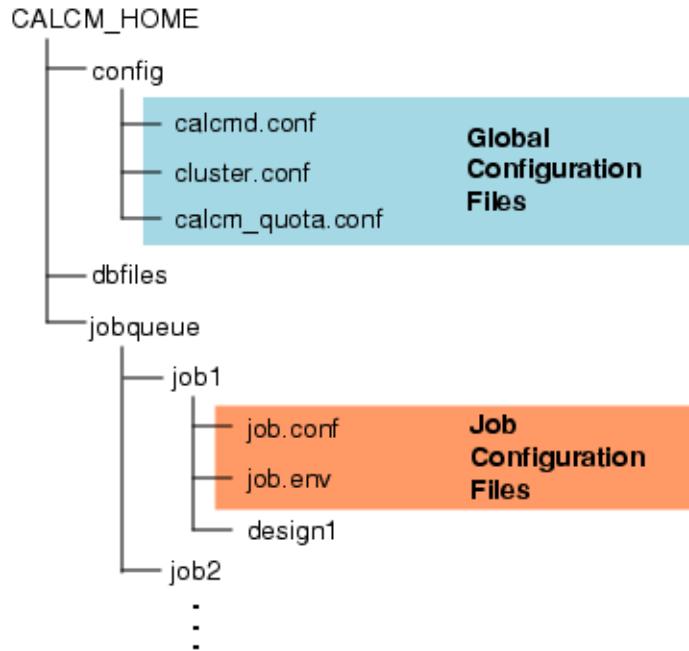
- [CalCM Configuration Files](#)
- [CalCM and Licensing](#)
- [calcm\\_send\\_message](#)
- [calcm\\_submit\\_job](#)

# CalCM Configuration Files

You use configuration files to configure CalCM for your environment. Some configuration files are used to define aspects of CalCM at a global level, while other configuration files are used to define aspects at a job (or local) level.

A recommended directory structure for these configuration files is shown in [Figure 2-1](#).

**Figure 2-1. CalCM Configuration Files**



## CalCM Configuration File (calcmd.conf)

The CalCM daemon requires a configuration file that defines the cluster manager options. The CalCM configuration file is an ASCII file consisting of a series of statements that define some aspects related to running the CalCM daemon.

Use the CalCM configuration file to define the following aspects associated with running the CalCM daemon:

- The applications that should run under the CalCM daemon.
- The connection between the CalCM daemon and License Broker daemon, as well as the licenses to allocate for CalCM jobs.
- The configuration of the server that is hosting the CalCM daemon.

When you invoke CalCM, the daemon searches for a CalCM configuration file in the following locations and order listed below:

1. A file defined by the -config command line option.
2. A file defined by the CALCMD\_CONF\_PATH environment variable.
3. The *calcmd.conf* file in the current working directory.
4. The *.calcmd.conf* file (note the period) in your home directory.
5. The *calcmd.conf* file in the system */etc* directory.

Upon locating the configuration file, the daemon locks the file and then loads the Tcl applications that are defined in the configuration file.

## Cluster Configuration File (*cluster.conf*)

A cluster configuration file is an ASCII file you use to define the Primary (Master), RDS, or Remote hosts to allocate for the Calibre jobs that run under CalCM.

The *cluster.conf* file typically resides in the *config* directory along with the *calcmd.conf* file (as shown in [Figure 2-1](#)). The format of the cluster configuration file is:

```
-----  
// MASTER CONFIGURATION  
-----  
MASTER HOST <host_name> SLOT <#_slots>  
  
-----  
// REMOTE CONFIGURATION  
-----  
REMOTE HOST <host_name> SLOT <#_slots>  
  
-----  
// RDS CONFIGURATION  
-----  
RDS HOST <host_name> SLOT <#_slots>
```

You can specify more than one Primary (Master), RDS, or Remote host statement in the cluster configuration file.

## Job Configuration File (job.conf)

You use the job configuration file to define job-specific attributes, such as the priority to assign the job, the rule file to use for the job, and the budget to assign job operations. The job configuration file is also used to define job account information.

In CalCM, there are two account types:

- Department — The first value specified in the JOB QUOTA statement defines the department associated with the job.
- Cost center — The first value specified in the JOB USER statement defines the cost center associated with the job.

The job configuration file should reside in the same directory as the job data, as this is where the calcm\_submit\_job command looks for this file. When you use the calcm\_submit\_job command to submit a Calibre job to run under CalCM, the command reads the job configuration file and passes the attributes to the job.

The format of the job configuration file is:

```
JOB MODE MTFLEX
JOB PRIORITY 0
JOB RULE drc.rules
JOB ENV job.env
```

## Job Environment File (job.env)

You can use the job environment file to define job-specific environment variables, such as the MGC\_HOME environment variable, error output, design database file format (OASIS®<sup>1</sup> or GDSII). CalCM reads the information in this file and sets the values for the run environment prior to executing Calibre.

An example of a job environment file is:

```
setenv MGC_HOME path_to_MGC_HOME
setenv OUT_ERROR out.err
setenv OUT_OASIS out.oas
```

An environment variable within the value is allowed.

[Table 2-1](#) lists special variables that are recognized and parsed accordingly. Other application-specific environment variables may also be specified.

---

1. OASIS® is a registered trademark of Thomas Grebinski and licensed for use to SEMI®, San Jose. SEMI® is a registered trademark of Semiconductor Equipment and Materials International.

**Table 2-1. Recognized Variables**

<b>Variable</b>	<b>Description</b>
MGC_HOME	Specifies the path to the MGC home directory. The parameter “%VCO” in the value is mapped to “\${VCO}”.
MGC_DIR	Specifies a directory for MGC_MASTER_DIR and MGC_REMOTE_DIR.
MGC_MASTER_DIR	Specifies a directory used in calcm_mtflex.conf as “LOCAL HOST DIR \$MGC_DIR”.
MGC_REMOTE_DIR	Specifies a directory to be passed as a first argument when invoking rcalibre.

The location of the job environment file is defined by the JOB ENV statement.

## Quota Configuration File (calcめ\_quota.conf)

The quota configuration file is used to specify hierarchical quota distributions which are used by the calcめ\_rmanager\_app.tcl application to distribute resources.

You can create a quota configuration file (calcめ\_quota.conf) using the following syntax:

```
[GROUP]
SUB_GROUP    CPU_QUOTA
```

For example:

```
[TOP]
DEFAULT 1
D1      4
D2      2

[DEFAULT]
DEFAULT 1

[D1]
DEFAULT 1
P1      4
P2      2

[D2]
DEFAULT 1
P2      4
P3      4
```

You can then define the calcm\_rmanager\_app.tcl QUOTADB argument in the CalCM configuration file (*calcmd.conf*) to point to the quota configuration file. For example:

```
APPLICATION TCL calcm_rmanager_app.tcl FILE [
    QUOTADB = $CALCM_HOME/conf/calc quota.conf
]
```

You can also specify a hierarchical quota distribution for a job by defining the JOB QUOTA statement in the job configuration file.

## Related Topics

[JOB ENV](#)

[JOB QUOTA](#)

[JOB USER](#)

[CalCM Resource Management](#)

[Configuring the CalCM Environment](#)

[CalCM Configuration File Reference Dictionary](#)

# CalCM Applications

This section describes the key Tcl applications that control different aspects of CalCM. These applications are encrypted in the Siemens EDA software tree, but several code excerpts are included as examples in this manual.

---

### Note

 Some of the Tcl applications include their own library for application-specific use, as well as relying on a common library. For the Tcl applications to work correctly, these libraries must reside in the same location as the application.

---

## HTTP Server

The HTTP Server application (calcm\_http\_server.app.tcl) provides the ability to monitor hardware resources and to track associated nodes, servers, running threads, and job progress. CalCM provides access to a web application so that you can view and analyze cluster and job utilization.

## Job Queue

The Job Queue application (calcm\_jobqueue\_app.tcl) allows you to monitor the job queue and automatically dispatch jobs based on submission priority. This application provides a single point for you to submit jobs for execution on the cluster without having to manually configure Calibre MTflex runs.

## Resource Manager

The Resource Manager application (calcm\_rmanager\_app.tcl) automatically adds or removes CPUs based on job demand and performs contention resolution. This application allocates cluster nodes based on the CPU demand of running a job, the job priorities, and the available hardware resources.

## Resource Monitor

The Resource Monitor application (calcm\_rmonitor\_app.tcl) collects the resource (CPU) and license utilization for each job and generates a distribution of the finished job duration.

## System Analysis

The System Analysis application (calcm\_systemanalysis\_app.tcl) collects statistical and graphical analysis information that can then be viewed using the calcm\_http\_server\_app.tcl application.

## Turn Around Time (TAT)

The TAT application (calcm\_tat\_app.tcl) provides the ability to configure the budget and priority for a Calibre job in order to control the turn around time of the job.

## Related Topics

[CalCM Tcl Application Reference](#)

[calcm\\_http\\_server\\_app.tcl](#)

[calcm\\_jobqueue\\_app.tcl](#)

[calcm\\_notification\\_app.tcl](#)

[calcm\\_rmanager\\_app.tcl](#)

[calcm\\_rmonitor\\_app.tcl](#)

[calcm\\_systemanalysis\\_app.tcl](#)

[calcm\\_tat\\_app.tcl](#)

[calcm\\_tcl\\_app.tcl](#)

# CalCM and Licensing

CalCM utilizes the Calibre License Broker Daemon (LBD) to manage licenses during a Calibre run. The CalCM daemon startup process automatically starts the License Broker Daemon.

As part of the configuration of the CalCM cluster, you must dedicate a set of Calibre product licenses to be used for jobs running under CalCM. The set of licenses acquired in this manner is referred to as the cluster's license pool. Calibre jobs running under CalCM can only utilize

licenses from the license pool. You can adjust and renew the licenses assigned to the license pool using the [reset\\_license](#) and [renew\\_license](#) messaging commands.

Every licensed Calibre product falls into a specific product group. Some examples of product groups are “base” which includes DRC and DFM operations and “opcpro” which includes OPC operations. During a Calibre run, Calibre checks out the licenses in a product group until the operations in that group are complete. Calibre then releases those licenses and acquires the licenses for another product group that are required for performing those operations.

CalCM does support license substitutions. Refer to the [Calibre Administrator’s Guide](#) for information on license substitutions.

CalCM does not support the use of the -wait and -nowait command line arguments, used to control license queueing. The -turbo\_all command line argument is also not supported.

## CalCM and License Configuration

This section describes the statements you use in your CalCM configuration file to configure licensing for use with CalCM.

The licensing statements include:

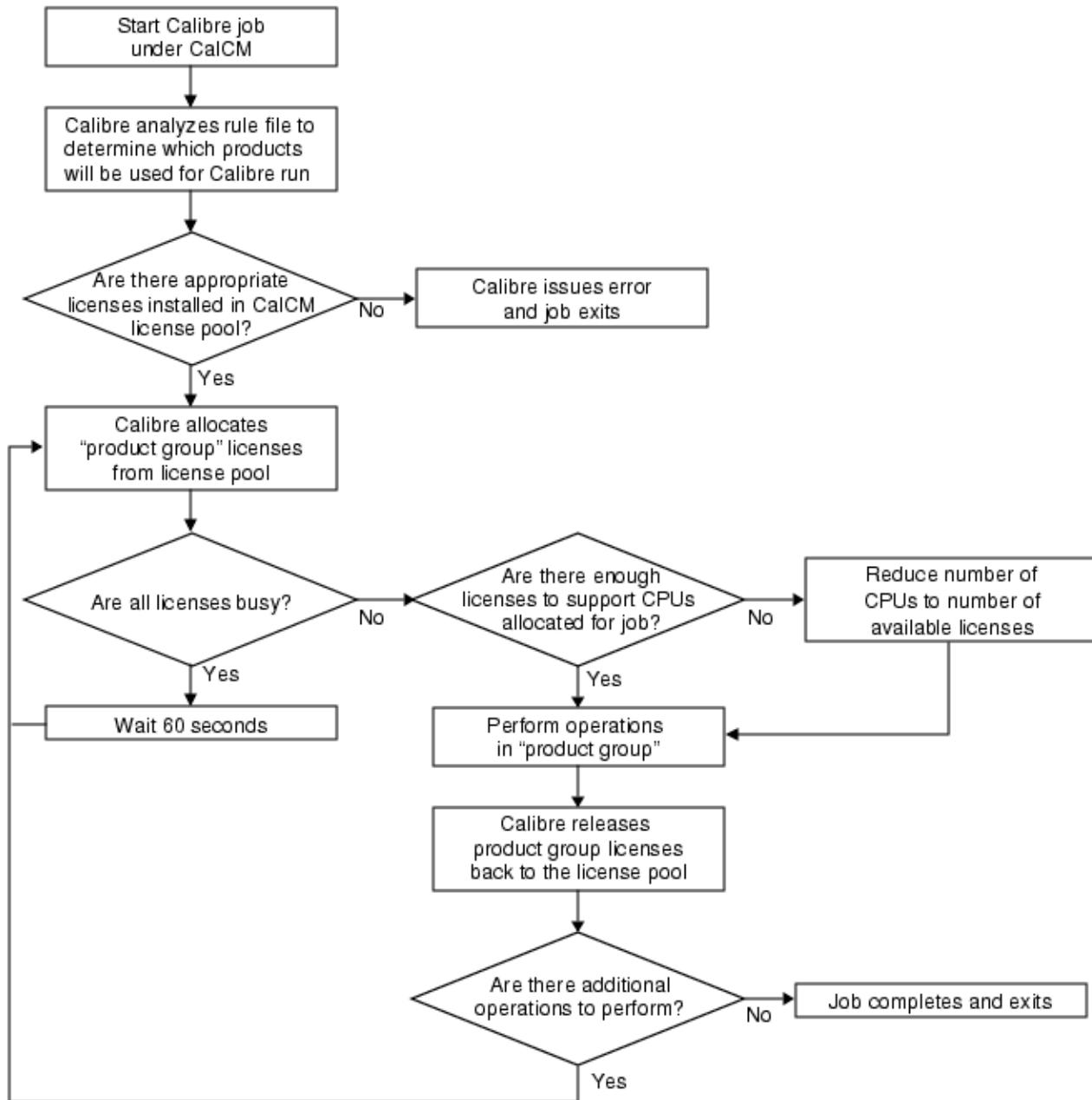
- LICENSE FEATURES — A required statement that defines the Calibre product license names and number of licenses to be acquired by CalCM for the license pool.
- LICENSE PORT — Specifies the TCP port used by the CalCM daemon to provide licenses for Calibre jobs. You only need to define this statement if you want to use a port other than the default port of 9901.
- LICENSE SERVER — Specifies the CalCM daemons that provide licenses from the LBD for Calibre jobs. This statement is required if you are running any backup CalCM daemons.

## CalCM and the LBD Process

This section describes the licensing flow for Calibre jobs that are managed by CalCM. When you start the CalCM daemon, the LBD also starts and establishes a connection with the CalCM daemon. The LBD then acquires the Calibre product licenses as defined by the LICENSE FEATURES statement and creates a license pool for CalCM. If the LBD is unable to immediately acquire all licenses specified for the pool, it continues to retry until all specified licenses are acquired.

[Figure 2-2](#) illustrates the licensing flow for Calibre jobs that are managed by CalCM.

**Figure 2-2. Licensing Flow for Calibre Jobs Managed By CalCM**



## Related Topics

[reset\\_license](#)

[LICENSE FEATURES](#)

[LICENSE PORT](#)

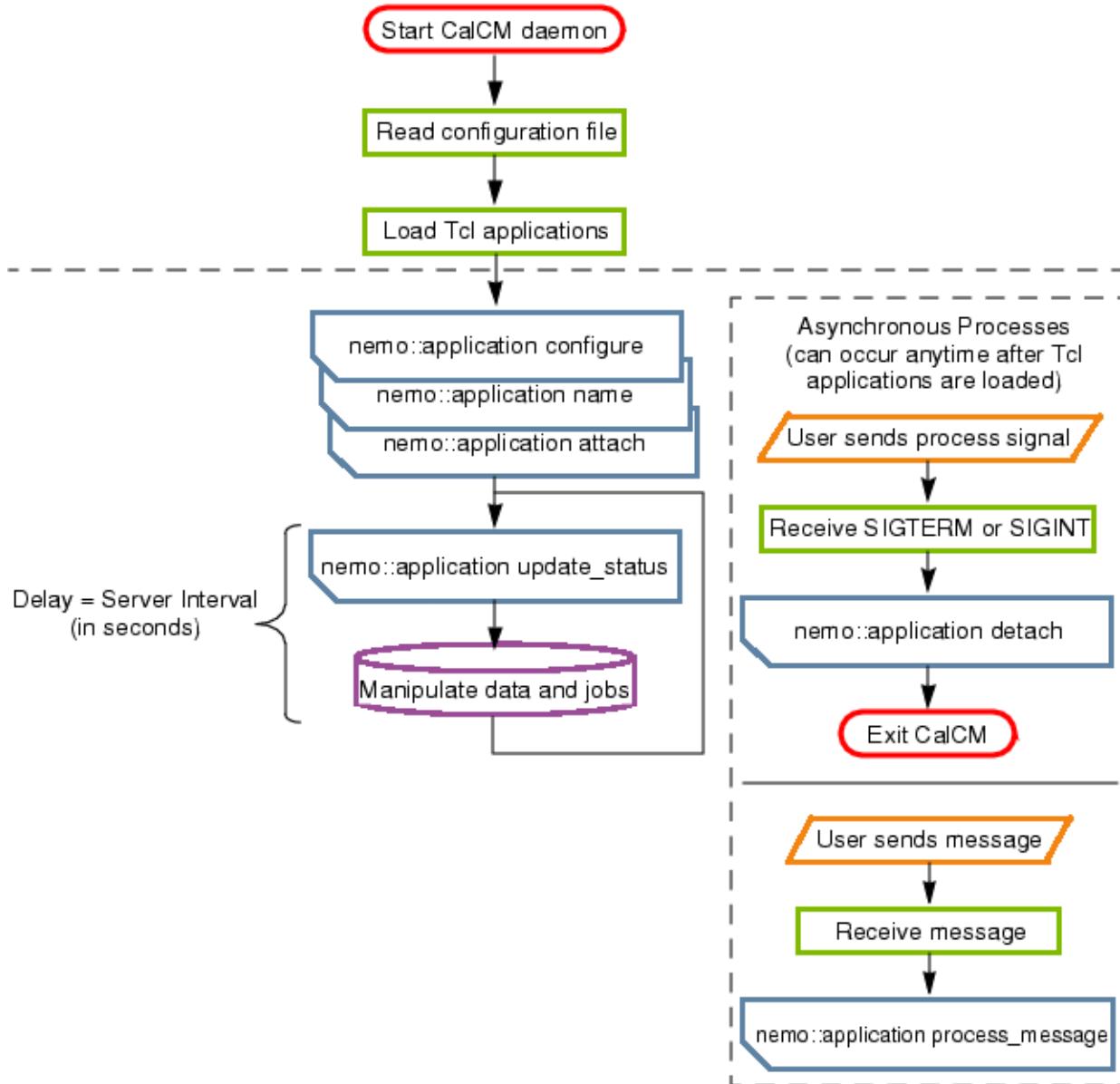
[LICENSE SERVER](#)

## CalCM Network Monitor (Nemo)

The core of CalCM is the Network Monitor functionality, also referred to as Nemo, whose primary role is to collect information about Calibre jobs. All interactions with the OS and hardware for the purpose of data collection and system monitoring are actually performed by the Calibre primary and remote processes. The information about the cluster state is made available to Tcl applications hosted via the Network Monitor API (Nemo API).

Figure 2-3 illustrates how Nemo communicates with Calibre jobs running on the cluster, transmits control commands, and periodically collects relevant job and system parameters.

**Figure 2-3. CalCM Process Flow**



**Note**

 Because, by its nature, Calibre Cluster Manager can host external applications, which can open and create files and network connections, running it as “root” can be a serious security threat. An under-privileged user account, such as “nobody”, should be used instead.

---

# CalCM Resource Management

This section describes some of the basic CalCM resource management tasks.

CalCM performs the following resource management tasks:

- Allocates resources when there is a demand in a job and revokes the resources when there is no longer a demand.
- Manages the resources based on information specified in the job and cluster configuration files.
- Continually monitors the resources to ensure they are appropriately balanced for each of the jobs running under CalCM.

## Resource Allocation

This section describes the job priority and job quota mechanisms in CalCM that you use to override the resource allocation.

### Job Priority

The priority can be passed to a job by specifying a JOB PRIORITY value in the job configuration file. The default minimum and maximum values are 0 and 16382, respectively. A job with a greater maximum value is given higher priority over jobs with a lower maximum value.

When the resource manager sees a job with a higher priority over other jobs, it does the following:

- Dispatches the highest priority job first from the job queue.
- Allocates resources for the prioritized job. If necessary, remotes are revoked from the lowest priority job in order to accommodate a higher priority job. The lowest priority jobs can continue to run with a minimum number of remotes.

Using a job priority for all jobs can cause an inefficient utilization of the remote cluster. To avoid this situation, you can set a quota for each job as described in the next section.

### Job Quota

The quota configuration file is used to specify hierarchical quota distributions, which are used by the resource manager (calcm\_rmanager\_app.tcl) to distribute resources among groups based on the CPU demand.

In addition to the quota configuration file, you can also specify a hierarchical quota for a job using the JOB QUOTA statement in the job configuration file. For example:

#### **JOB QUOTA D2/P3**

This information is passed to the resource manager which calculates the actual quota (or proportional share) of resources for a job and uses this information to allocate or revoke resources for the job.

### **Resource Allocation Process and Recommendations**

This section provides information on how CalCM allocates resources for jobs that are submitted without a priority or with the same priority as other jobs.

A CalCM job that is submitted without a priority or with the same priority as other jobs is managed as follows:

- The resource manager attempts to allocate resources based on the job quota and the targeted maximum.
- If all the jobs have a demand on resources, the resource manager matches the number of remotes to the targeted maximum.
- Due to lower scalability, the resource requirements for some jobs may not grow to the targeted maximum. In this case, the additional resources are distributed to other jobs with resource demands.

In the following example, the resource manager displays a log that shows the current number of remotes with demands. In this case, all jobs have 10 remotes and a total of 80 demands (job 1 + job 2). There is no priority assigned to these jobs.

```
Job "1" (0) : 10+40
Job "2" (0) : 10+40
Job "3" (0) : 10+0
```

If there are 55 free remotes that can be allocated and the targeted maximum resource allocation for job 1 and job 2 is 100 and 50, respectively, then job 1 will acquire 40 remotes and job 2 will acquire the remaining 15 remotes. This resource allocation maintains the 2:1 ratio between job 1 and job 2.

```
Job "1" (0) : 50+0
Job "2" (0) : 25+0
Job "3" (0) : 10+0
```

This mechanism for managing the targeted maximum resource allocation is an alternative method for balancing the jobs with no priority along with jobs that have a priority. In other words, you should only assign a priority to urgent jobs and use the targeted maximum mechanism to handle normal priority jobs.

## Related Topics

[JOB PRIORITY](#)

[JOB QUOTA](#)

[CalCM Job Configuration File Reference](#)

[CalCM Cluster Configuration File Reference](#)

[CalCM Configuration Files](#)

[calcm\\_rmanager\\_app.tcl](#)

# CalCM Resource Monitoring

The Resource Monitoring application (`calcm_rmonitor_app.tcl`) collects resource, license, and job information. Using a CalCM web application, you can view this collected information in the form of custom usage plots and data tables.

## Related Topics

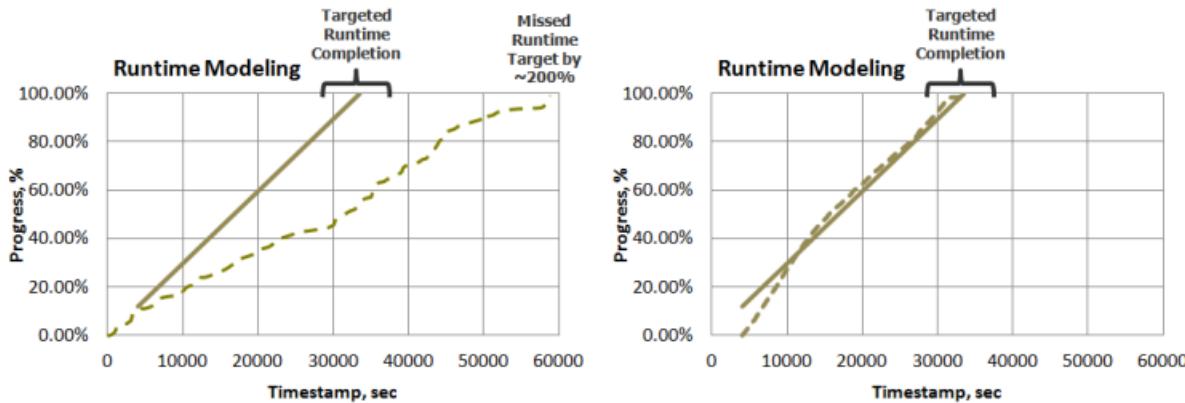
[calcm\\_rmonitor\\_app.tcl](#)

# CalCM TAT

The CalCM TAT application extends the CalCM capabilities by providing improved predictability for runtime modeling jobs.

[Figure 2-4](#) compares the runtime of a Calibre job run with and without the CalCM TAT application.

**Figure 2-4. Comparison of Runtime Modeling Job With and Without TAT**



To achieve a targeted runtime completion, the TAT application handles the following aspects of each Calibre job that runs under CalCM TAT:

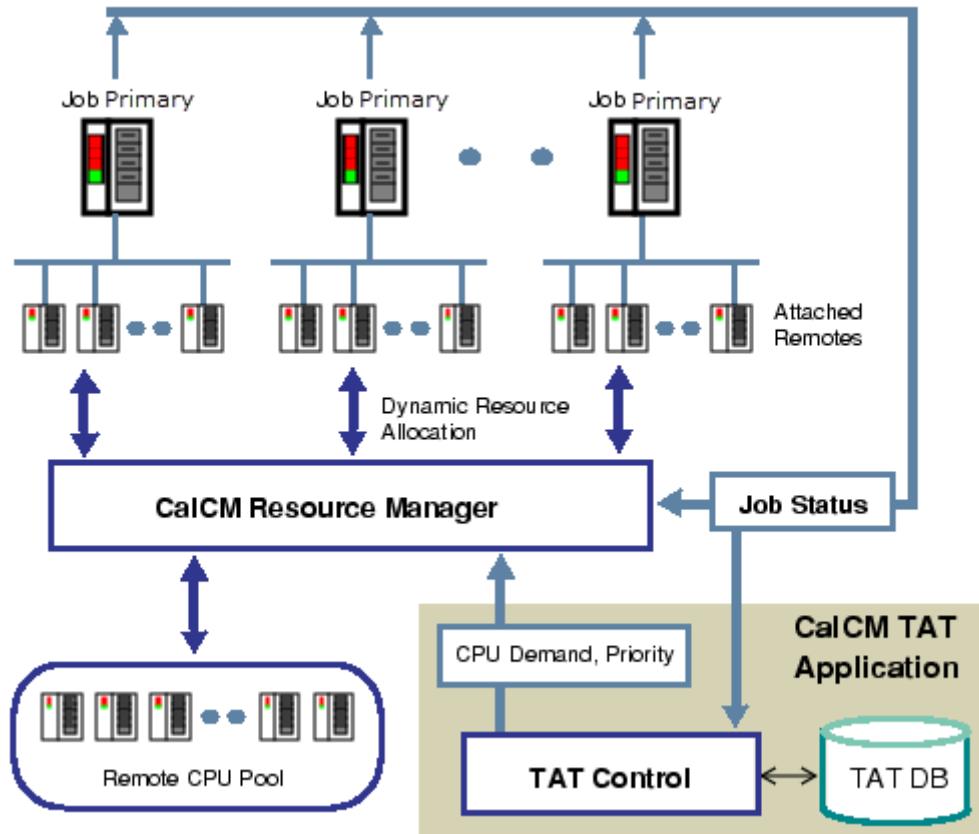
- Budget Management
- Resource Allocation
- Execution Order
- Priority Control

For an overview of the configuration and workflow for setting up and running the TAT application with CalCM, refer to “[Configuring and Using the CalCM TAT Application](#)” on page 363.

The CalCM TAT application is a Tcl application named *calcm\_tat\_app.tcl*. This application, which resides and works with the existing CalCM applications in the MGC\_HOME tree, is automatically run when you start the CalCM daemon. Refer to the [calcm\\_tat\\_app.tcl](#) reference page for more information on this application.

During the CalCM update cycle, the TAT application collects data about the jobs that are running at that time and stores the information in the TAT database. This data is used to perform calculations for the current and future jobs that run under the TAT application.

[Figure 2-5](#) illustrates how the CalCM TAT application and database work within the existing CalCM framework.

**Figure 2-5. CalCM Dynamic Resource Allocation and TAT Control**

## Budget Management

A single Calibre job typically executes multiple operations, where each operation has unique CPU and scalability requirements. For each operation that supports the progress meter feature<sup>2</sup>, a time budget can be applied to LITHO operations that have a runtime impact.

When running a Calibre job, the CalCM TAT application collects data about the job and stores this information in the TAT database. The CalCM TAT application then uses this data to calculate the budget for new jobs and operations. There are two budgeting modes:

- Manual budgeting — In manual mode, you define the budget for the job operations. Using the JOB BUDGET statement in the job configuration file (*job.conf*), you define the budget for specific job operations. For example:

```
JOB BUDGET nmOPC1=120;verify=300
```

---

2. The progress meter is used to monitor the progress of the LITHO operations in a run. This feature provides metrics for analysis and projection of runtime. There is a small (1%) performance penalty when using this feature.

- Automatic budgeting — In automatic mode, the TAT application defines the budget for job operations using data collected from previous runs. To enable this mode, you must use the JOB BUDGET statement to specify automatic mode and the total budget for the job. For example:

```
JOB BUDGET _AUTO_; TOT:400
```

**Note**

 The automatic mode is recommended, as it is easier to use.

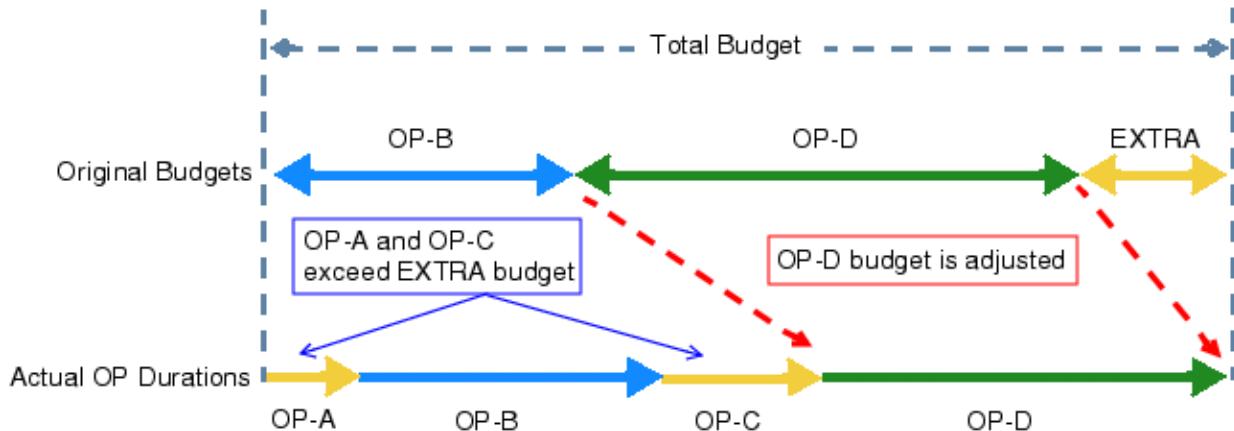
---

During the Calibre run, the budget specified in the job configuration file is dynamically adjusted when both conditions are met:

- ADJUST\_BUDGET in the CalCM configuration file is enabled (this is the default).
- The remaining job duration exceeds the accumulated budget.

The TAT application dynamically adjusts the budget for an operation to ensure the job meets the specified job budget (see [Figure 2-6](#)). This is achieved through database queries that continuously update the job budget based on the status of the current job and any previously-collected job data.

**Figure 2-6. Dynamic Budget Adjustment**



## Resource Allocation

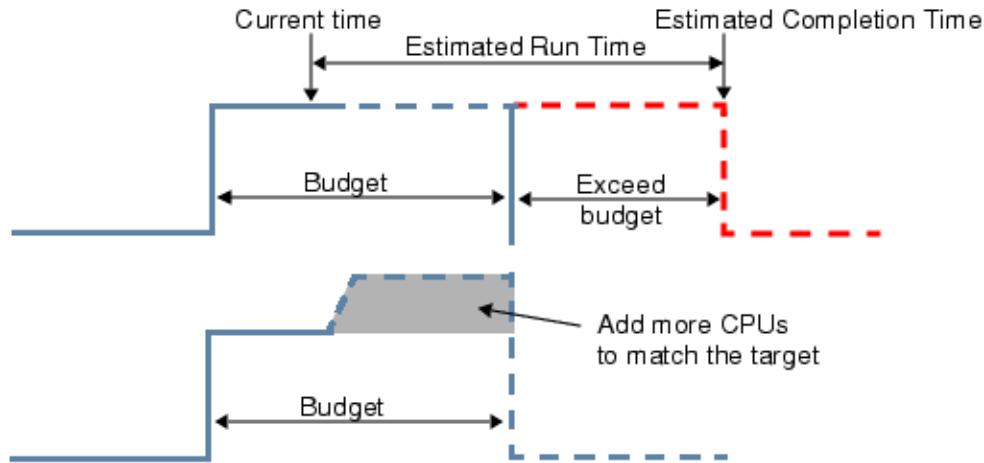
This section discusses how the CalCM Resource Manager application dynamically allocates resources to Calibre jobs running under CalCM.

The TAT application provides resource demand information to the CalCM Resource Manager which uses this information to allocate resources to jobs (as shown in [Figure 2-7](#)) based on the following rules:

- Resources are allocated to jobs by CPU demand and priority.

- The CPU demand is computed based on the job scalability, which is computed by monitoring resource usage and processing the data queue length.

**Figure 2-7. Allocating Resource with CalCM TAT**



## Execution Order

During a Calibre run, frequent changes to the allocation of resources can cause runtime penalties due to the overhead associated with managing the resources. The TAT application attempts to minimize any resource demand conflicts by scheduling the execution order of jobs. If a job includes low priority operations that can wait for the completion of a job with higher priority operations, then the TAT application assigns a lower priority to the job containing the low priority operations.

## Priority Control

CalCM provides the ability to set the priority for a job by defining the **JOB PRIORITY** statement in the job configuration file. With the TAT application, you can also assign a “priority bump” value which specifies the maximum value that the TAT application can increase the priority for a job. A default global priority bump value is defined in the `calcm_tat_app.tcl` application.

You can override this global value by defining a new priority bump value in the CalCM configuration file (`calcmd.conf`). The global value can be overridden for a job by specifying the **JOB\_TAT\_PRIORITY\_BUMP** in the job configuration file (`job.conf`) or by using the `set_priority_bump` messaging command.

You can use a CalCM web application in a browser to view the priority bump for a job. Refer to “[Using the CalCM Dashboard Web Application](#)” on page 54 for information on using this application.

## Notification Events

This section discusses the different runtime (or notification) events supported by the TAT application through the execution of a user-specified script.

The supported runtime events include:

- Check Point
- Estimation Exceed
- Runtime Exceed
- Exit Status

You use the NOTIFICATION\_FILE\_PATH argument in calcm\_tat\_app.tcl to specify the path to the notification script. The notification feature is disabled if a notification script is not specified.

### Check Point

A check point event occurs when a specified operation is not finished within a specified time. To enable this event, you must specify the CHECK argument for the JOB BUDGET statement in the job configuration file.

The format for this statement and argument is:

JOB BUDGET CHECK:*operation\_name:check\_time*

### Estimation Exceed

An estimation exceed event occurs when the estimated job duration exceeds the budget by more than the specified exceed ratio. The exceed ratio is specified as a percentage by defining the NOTIFICATION\_EXCEED\_RATIO argument for the calcm\_tat\_app.tcl application. The default value for this argument is 0 which disables the estimation exceed event.

The argument NOTIFICATION\_INTERVAL specifies the minimum time interval, in minutes, that the notification event occurs. The default value for this argument is 0 which causes a notification event to be issued only once.

### Runtime Exceed

A runtime exceed event occurs when the actual job duration exceeds the total budget by more than the specified exceed ratio. The exceed ratio is specified as a percentage by defining the NOTIFICATION\_EXCEED\_RATIO argument for the calcm\_tat\_app.tcl application. The default value is 0 which disables the runtime exceed event.

### Exit Status

A job exit status is sent to CalCM when a job finishes. An exit status of “0” indicates the job finished without any errors. The exit status can be controlled by defining the

NOTIFICATION\_ESTATUS argument for the calcm\_tat\_app.tcl application. The default value is 0 which enables an “exit status” event when the exit status is not “0.”

## Related Topics

[CalCM TAT Workflow](#)

[JOB BUDGET](#)

[JOB PRIORITY](#)

[JOB TAT\\_PRIORITY\\_BUMP](#)

[set\\_priority\\_bump](#)

[calcm\\_tat\\_app.tcl](#)

[Using Notifications with the TAT Application](#)



# Chapter 3

## Deploying and Running CalCM

---

This section includes several procedures that explain how to deploy and run CalCM and monitor resources and job information using the CalCM dashboard web application.

<b>Configuring the CalCM Environment . . . . .</b>	<b>45</b>
<b>Starting the CalCM Daemon . . . . .</b>	<b>49</b>
<b>Configuring Calibre Jobs to Run Under CalCM . . . . .</b>	<b>51</b>
<b>Using the CalCM Dashboard Web Application . . . . .</b>	<b>54</b>
CalCM Dashboard Requirements . . . . .	54
Executing Calibre Jobs to Run Under CalCM with the Dashboard . . . . .	55
Monitoring CalCM Notifications with the Dashboard . . . . .	61
Monitoring CalCM Resources with the Dashboard . . . . .	64
Monitoring Jobs Running Under CalCM with the Dashboard . . . . .	66
Monitoring Resource Usage and Allocation with the Dashboard . . . . .	71
Displaying System Information in the CalCM Dashboard . . . . .	75
Viewing Chart Information in the CalCM Dashboard . . . . .	78
Comparing Job Information in the CalCM Dashboard . . . . .	80
Refreshing Chart Data in the CalCM Dashboard . . . . .	85
Accessing and Updating Settings in the CalCM Dashboard . . . . .	87
CalCM+ Advanced Features for Data Analysis in the CalCM Dashboard . . . . .	92
CalCM+ Advanced Features for Accessing CalScope in the CalCM Dashboard . . . . .	103
<b>Running CalCM with a Backup Daemon . . . . .</b>	<b>106</b>

## Configuring the CalCM Environment

The following procedure provides the setting for deploying CalCM.

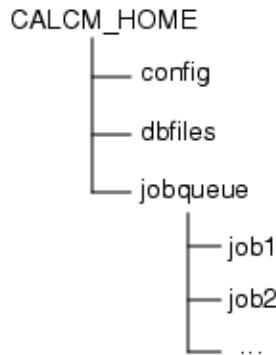
### Prerequisites

- A central location for storing CalCM configuration files, database files, and job data.
- Knowledge of hardware configuration, such as which server the CalCM daemon will run on.
- Knowledge of installed Calibre product licenses.
- Primary and remote hosts for running Calibre jobs.
- Web server for monitoring the CalCM run.

## Procedure

1. Set your working directory to the desired location in which you will create the recommended directories and files required to run CalCM.
2. Create the CalCM directory structure as shown in [Figure 3-1](#).

**Figure 3-1. Recommended CalCM Directory Structure**



3. Create a CalCM configuration file (*calcmd.conf*) in the *config* directory.
  - a. Use the **VARIABLE** statement to define any environment variables, such as the location of your CALCM\_HOME directory or CalCM database files, for your environment. The CALCM\_APP definition points to the source CalCM Tcl applications in the MGC\_HOME tree. For example:

```
// Define path to calcmd files
VARIABLE CALCM_HOME <installation_path>/calcm
VARIABLE CALCM_APP $MGC_HOME/pkgs/icv_calcm/tcl

// Define database paths
VARIABLE JOB_DB $CALCM_HOME/dbfiles/calcm_job.db
VARIABLE MONITOR_DB $CALCM_HOME/dbfiles/calcm_rmonitor.db
```

- b. Define the CalCM server and license statements for your environment. For the **SERVER PORT** statement, you must replace *port\_number* with the number of the port you want the CalCM daemon to use. For the **LICENSE SERVER** statement, replace *server\_name* with the name of your license server. Add the **LICENSE FEATURES** statement and specify the feature name(s) and number of licenses for your site. For example:

```
SERVER PORT port_number      //Port number used by CalCM daemon

LICENSE PORT 9901
LICENSE FEATURES FILE [
    feature_name = number
    // example: calnmopc = 300
    ...
]
```

- c. Use the **APPLICATION** statement to define the Tcl applications that are loaded when you start the CalCM daemon. For performance reasons, it is recommended that you specify the applications in the following order:

```
// "Job Queue" Tcl application.  
APPLICATION TCL $CALCM_APP/calcm_jobqueue_app.tcl FILE [  
]  
  
// "Resource Manager" Tcl application.  
APPLICATION TCL $CALCM_APP/calcm_rmanager_app.tcl FILE [  
]  
  
// "System Analysis" Tcl application.  
APPLICATION TCL $CALCM_APP/calcm_systemanalysis_app.tcl FILE [  
]  
  
// "Remote Monitor" Tcl application.  
APPLICATION TCL $CALCM_APP/calcm_rmonitor_app.tcl FILE [  
]  
  
// "Http Server" Tcl application.  
APPLICATION TCL $CALCM_APP/calcm_http_server_app.tcl FILE [  
]  
  
// "Snapshot Database" Tcl application.  
APPLICATION TCL $CALCM_APP/calcm_tcl_app.tcl FILE [  
    SNAPSHOTDIR = $CALCM_HTML, 777  
]
```

- d. Customize the Job Queue, Remote Monitor, and HTTP Server, and Snapshot Database Tcl applications.

Add the following lines to customize the Tcl applications:

```
// "Job Queue" Tcl application.  
APPLICATION TCL $CALCM_APP/calcm_jobqueue_app.tcl FILE [  
    //Path for job queue directory  
    QUEUEDIR = $CALCM_HOME/jobqueue  
    //Path to cluster configuration file  
    CLUSTERCONF = $CALCM_HOME/config/cluster.conf  
    //Path for job database  
    JOBDB = $JOB_DB  
]  
  
APPLICATION TCL $CALCM_APP/calcm_rmonitor_app.tcl FILE [  
    DBPATH = $RMONITOR_DB  
]  
  
// "Http Server" Tcl application.  
APPLICATION TCL $CALCM_APP/calcm_http_server_app.tcl FILE [  
    //Path for job database  
    JOBDB = $JOB_DB  
    ENABLE_DASHBOARD = 1  
    DASHBOARDPORT = 9902  
]  
  
// "Snapshot Database" Tcl application.  
APPLICATION TCL $CALCM_APP/calcm_tcl_app.tcl FILE [  
    SNAPSHOTDIR = $CALCM_HTML,777  
]
```

Notice that you defined the JOBDB argument for both the Job Queue and HTTP Server applications. This is necessary as the job database is used by both applications. The cluster Snapshot Database is used by the CalCM web application.

- e. Save and close the file.
4. Create a CalCM cluster configuration file. A cluster configuration file is an ASCII file that defines the different types of hosts (Primary (Master), RDS, Remote) to be managed in CalCM and allocated for the Calibre job.
  - a. Use the **MASTER HOST**, **RDS HOST**, and **REMOTE HOST** statements to define the hosts in your environment. Each statement can be specified multiple times.

```
//-----  
// MASTER CONFIGURATION  
//-----  
MASTER HOST <host_name> SLOT <#_slots>  
  
//-----  
// REMOTE CONFIGURATION  
//-----  
REMOTE HOST <host_name> SLOT <#_slots>  
  
//-----  
// RDS CONFIGURATION  
//-----  
RDS HOST <host_name> SLOT <#_slots>
```

Refer to the “[CalCM Cluster Configuration File Reference](#)” for information on the statements used in the job configuration file.

- b. Save and close the file.

## Starting the CalCM Daemon

This procedure describes how to start the CalCM daemon and view job information in a browser.

### Prerequisites

- CalCM configuration file (*calcmd.conf*).
- Cluster configuration file (*cluster.conf*).
- You must know the server name on which you will run the CalCM daemon and the server port (defined in previous procedure) used to communicate with the daemon. You must also know the host names for the backup CalCM daemon system and backup license server.
- Browser access.

### Procedure

1. Prior to starting the CalCM daemon, you can optionally set the **CALCMD\_LOGLEVEL** environment variable to a value other than the default of “MESSAGE”.
2. Set your working directory to **CALCM\_HOME** (refer to “[Configuring the CalCM Environment](#)” on page 45), and start the CalCM daemon.

```
$MGC_HOME/bin/calcmd -config config/calcmd.conf
```

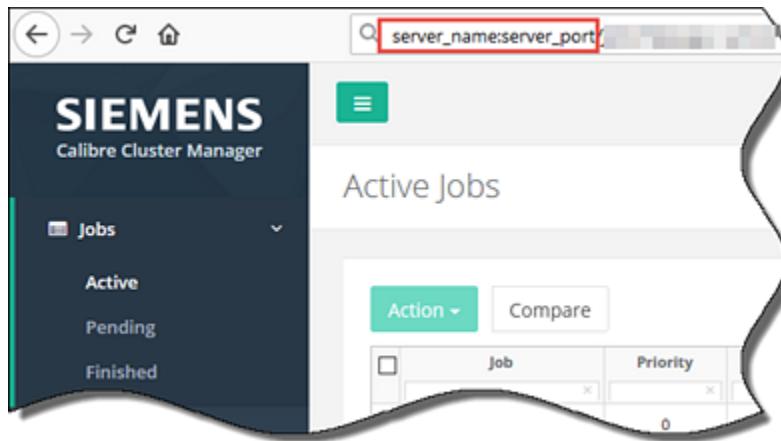
The CalCM daemon startup process creates the `calcml_submit_job` and `calcml_send_message` scripts. These scripts are created in the directory defined by the `QUEUEDIR` argument in the `calcml_jobqueue_app.tcl` application. You use these scripts to launch Calibre jobs under CalCM and to communicate with the CalCM applications.

Upon starting, the CalCM daemon generates a transcript which is captured in `$CALCM_HOME/calcmd.log`.

3. Open a browser and, using the following format, enter the server name on which the CalCM daemon is running and the server port specified in the CalCM configuration file.

`http://server_name:server_port`

- In the CalCM dashboard web application, the following web page displays:



---

#### Note

 The CalCM dashboard displays a web login page requesting authentication information (user name and password). Some menus and actionable items are only accessible at certain user levels. For information on user authentication in the CalCM dashboard, see “[Accessing and Updating Settings in the CalCM Dashboard](#)” on page 87 and LDAP-related configuration keywords in the `calcm_http_server_app.tcl` application.

---

The CalCM dashboard web application displays status and resource information. When Calibre jobs are launched, the browser updates to display the active and pending jobs.

## Results

The CalCM daemon is now running. You can use the SIGINT (interrupt process) or SIGTERM (terminate process) signal to stop the CalCM daemon process.

When you start the daemon, it automatically creates the `calcm_send_message` and `calcm_submit_job` scripts in the `CALCM_HOME/jobqueue` directory. These scripts are used to submit and communicate with jobs running under CalCM.

It is recommended that you start a backup CalCM daemon as described in “[Running CalCM with a Backup Daemon](#).”

# Configuring Calibre Jobs to Run Under CalCM

The following procedure describes how to create job configuration files for Calibre jobs you want to run under CalCM.

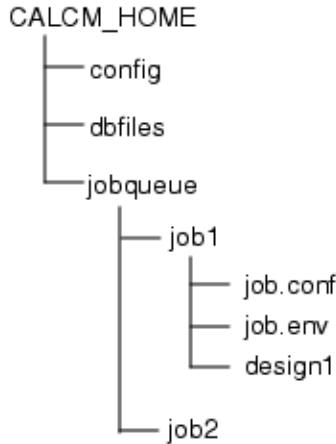
## Prerequisites

- You will need to know information for configuring the job, such as the mode (MT, Calibre MTflex), rule file, and number of remotes.

## Procedure

1. Place a design in the *job1* directory as shown in [Figure 3-2](#).

**Figure 3-2. CalCM Directory Structure**



2. Create a job configuration file (*job.conf*) in the *job1* directory.

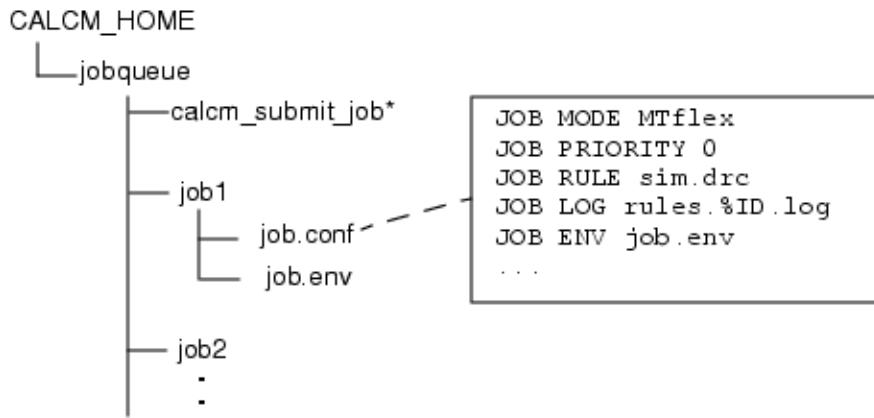
- a. Define the CalCM job statements as needed for your environment, being sure to replace the strings shown in italics with your specific job and environment information.

```
JOB MODE mode           //The mode for running the SVRF rule file
JOB PRIORITY number    //The priority for this job
JOB RULE rule_file      //Rule file used for this job
JOB LOG log_file        //Log file captures results of job
operations
JOB ENV env_file        //Environment file for this job

REMOTE MAX number      //Maximum of remotes for this job

JOB USER username       //User running this job
```

Refer to the “[CalCM Job Configuration File Reference](#)” for information on the statements used in the job configuration file.

**Figure 3-3. Example CalCM Job Configuration File**

\* The `calcm_submit_job` script is automatically created by the CalCM job queue application.

- b. Save and close the file.
3. Optional: You can create a job environment file in the `job1` directory and define OPC-related variables that are specific to `job1`. For example, for an OPC job you can define OPC-related variables in this file.

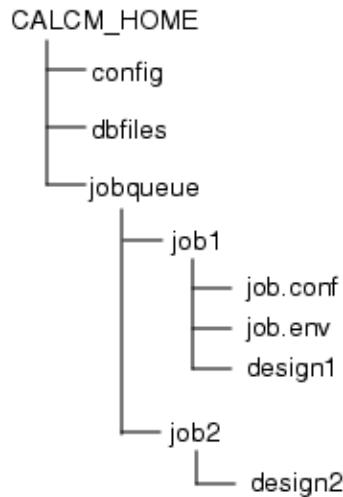
```

setenv MGC_HOME <path_to_MGC_HOME>
setenv OUT_ERROR:out.err
setenv OUT_OASIS:out.oas
  
```

If you create this file, you should define the `JOB ENV` statement in step 2 to point to this file.

4. Place a design in the `job2` directory as shown in [Figure 3-4](#).

**Figure 3-4. CalCM Directory Structure**



5. Create a job configuration file (*job.conf*) in the *job2* directory.
  - a. Define the CalCM job statements as needed for your environment, being sure to replace the strings shown in italics with the correct information.

```
JOB MODE mode
JOB PRIORITY number
JOB RULE rule_file
JOB LOG log_file
JOB ENV env_file

REMOTE MAX number      //Maximum of remotes for this job
JOB USER username      //User running this job
```

- b. Save and close the file.
6. Optional: You can create a job environment file in the *job2* directory and define OPC-related variables that are specific to job2. If you create this file, you should define the JOB ENV statement in step 5 to point to this file.

## Results

You now have two jobs, job1 and job2, that are configured to run under CalCM.

# Using the CalCM Dashboard Web Application

---

The CalCM dashboard web application is used for monitoring a running CalCM process, visualizing graphical data, and providing user management and job control.

<b>CalCM Dashboard Requirements .....</b>	<b>54</b>
<b>Executing Calibre Jobs to Run Under CalCM with the Dashboard.....</b>	<b>55</b>
<b>Monitoring CalCM Notifications with the Dashboard.....</b>	<b>61</b>
<b>Monitoring CalCM Resources with the Dashboard.....</b>	<b>64</b>
<b>Monitoring Jobs Running Under CalCM with the Dashboard.....</b>	<b>66</b>
<b>Monitoring Resource Usage and Allocation with the Dashboard.....</b>	<b>71</b>
<b>Displaying System Information in the CalCM Dashboard .....</b>	<b>75</b>
<b>Viewing Chart Information in the CalCM Dashboard .....</b>	<b>78</b>
<b>Comparing Job Information in the CalCM Dashboard.....</b>	<b>80</b>
<b>Refreshing Chart Data in the CalCM Dashboard .....</b>	<b>85</b>
<b>Accessing and Updating Settings in the CalCM Dashboard .....</b>	<b>87</b>
<b>CalCM+ Advanced Features for Data Analysis in the CalCM Dashboard .....</b>	<b>92</b>
<b>CalCM+ Advanced Features for Accessing CalScope in the CalCM Dashboard .....</b>	<b>103</b>

## CalCM Dashboard Requirements

The system requirements and prerequisites for running the CalCM dashboard web application are provided.

### System Requirements

- Client Requirements

These are the recommend requirements for client PCs to display the dashboard:

- Firefox 77 or later web browser for the dashboard web application (minimum version is Firefox 45 for expected functionality).
- Monitor with 1600x900 (16:9) or greater screen resolution

- Server requirements

These are the mandatory requirements for running the CalCM dashboard on the server:

- Mandatory Linux system packages
  - openssl, openssl-devel
- Optional Linux system packages

- For recommended optional Linux system packages, contact your system administrator or your Siemens representative.

## Prerequisites

These are the prerequisites for running the CalCM dashboard in your environment:

- CalCM 2018.1 P6 (2018.1\_37.28) or later version on your server.
- Recommended directory structure as described in “[Configuring the CalCM Environment](#)” on page 45.
- Customized HTTP server Tcl application in the CalCM configuration file per “CalCM Dashboard Web Application” as described in “[Configuring the CalCM Environment](#)” on page 45.

## Best Practices

It is recommended that you clear your browser cache after a CalCM version migration or after a patch is applied. If a page does not display properly, clear your browser cache and retry the page in the CalCM dashboard.

# Executing Calibre Jobs to Run Under CalCM with the Dashboard

This procedure describes the various job web pages and shows you how to submit Calibre jobs to run under CalCM with the CalCM dashboard web application.

## Prerequisites

- The CalCM daemon (calcmd) and CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Browser access to the CalCM dashboard web application.

---

### Note

 The CalCM dashboard displays a web login page requesting authentication information (user name and password). Some menus and actionable items are only accessible at certain user levels. For information on user authentication in the CalCM dashboard, see “[Accessing and Updating Settings in the CalCM Dashboard](#)” on page 87 and LDAP-related configuration keywords in the `calcm_http_server_app.tcl` application.

---

## Procedure

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM daemon configuration file (*calcmd.conf*).

`http://server_name:server_port`

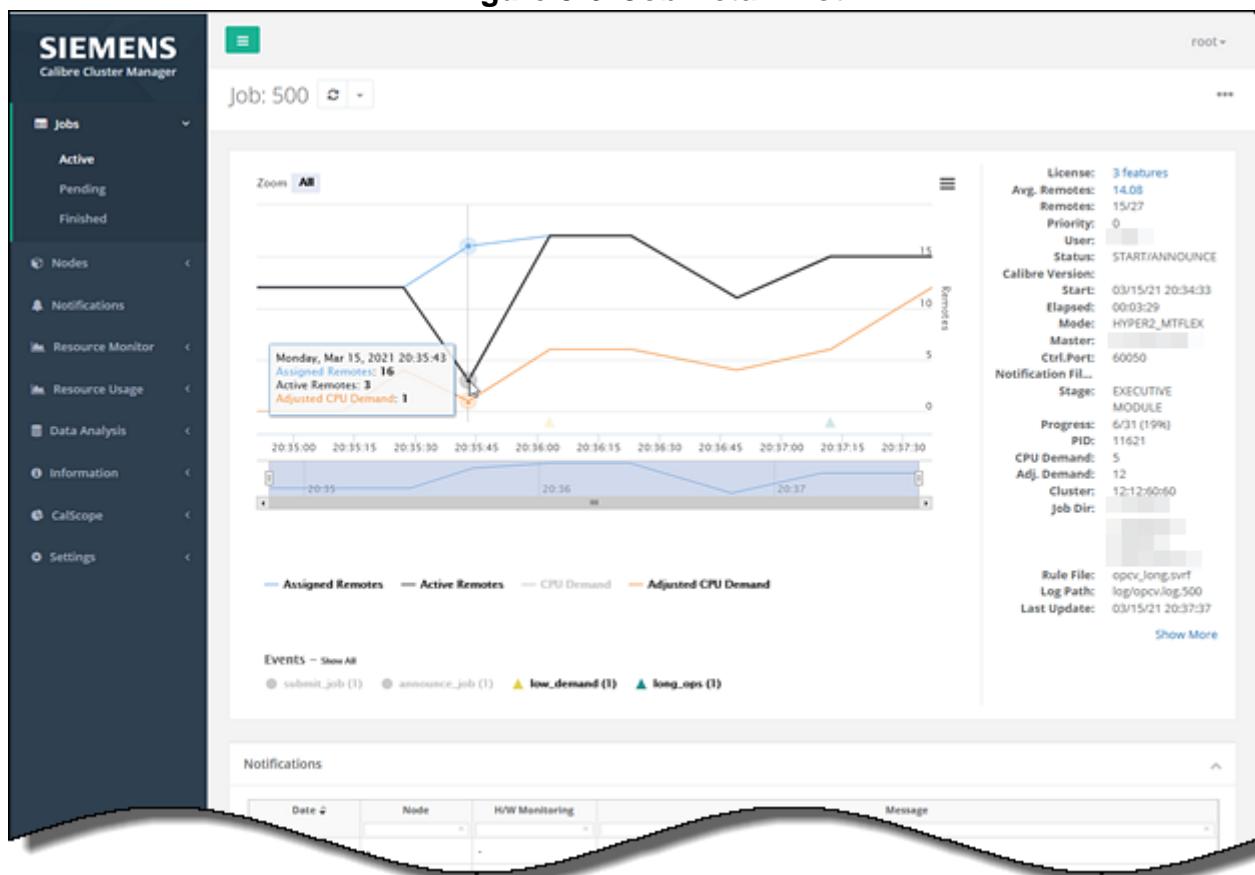
This displays the CalCM dashboard web application and defaults to the Active Jobs page in the **Jobs** menu item in the left pane.

**Figure 3-5. Active Jobs**

Action	Job	Priority	User	Start	Elapsed	Mode	Status	Master	Remotes	Active Remotes	Allocated Remotes
	464	0		03/15/21 19:24:14	00:01:01	HYPER2_M...	START/AN...		14/12	14	12
	463	0		03/15/21 19:20:22	00:04:53	HYPER2_M...	START/AN...		12/23	12	23
	462	0		03/15/21 19:18:27	00:06:48	HYPER2_M...	START/AN...		20/20	20	20
	461	1		03/15/21 19:17:18	00:07:37	HYPER2_M...	START/AN...		6/21	0	21

You can click a job number in the first column of the table to display the detailed information for a specific job in a plot format. In addition to information on the active and assigned remotes and CPU demand, this information can also include quota values and job notifications using JOBFILTERS in the `calcm_notification_app.tcl` application. Job events occurring during the run are shown at the bottom of the plot.

**Figure 3-6. Job Detail Plot**



If you select the check box next to the job number in the Active Jobs table, the Action filter is activated with a list of command actions for the active job. If multiple jobs are selected, the list displays only the applicable actions.

This requires you to be authenticated because you are only allowed to act on your own jobs. CalCM administrator logins can also be defined with permissions to act on all jobs. See [Figure 3-40](#) on page 89 for information on setting permission levels for different groups.

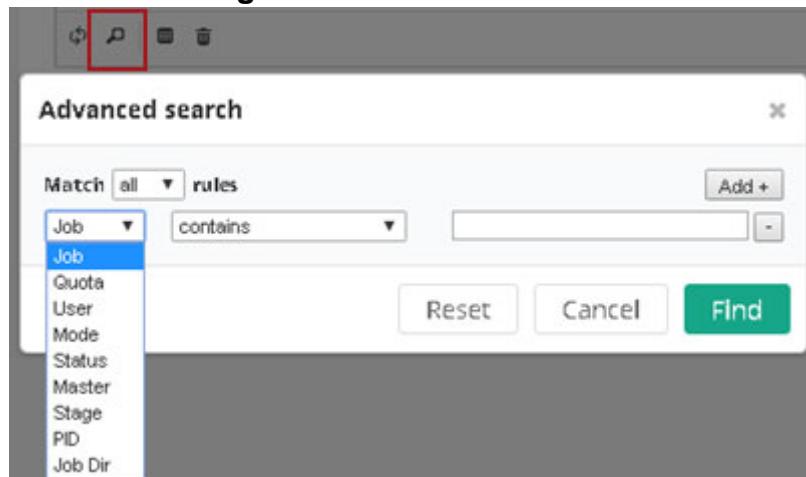
Figure 3-7. Job Action Filter

The screenshot shows the CalCM interface with the 'Active' tab selected. The main area displays a table of active jobs with columns for Quota, Priority, User, Start At, Elapsed, Mode, Status, Master, Remotes, Avg. Remotes, Notification, and Usage. Below the table is a search bar and a message indicating 1 of 8 results. The 'Action' dropdown menu is open, listing the options mentioned in the caption.

The available job actions and CalCM messaging commands are listed:

- **Kill job** — [calcm\\_kill\\_job](#)
  - **Suspend job** — [suspend\\_job](#)
  - **Prioritize job** — [prioritize\\_job](#)
  - **Adjust quota** — [adjust\\_quota](#)
  - **Adjust minmax** — [adjust\\_minmax](#)
  - **Adjust maxchange** — [adjust\\_maxchange](#)
  - **Adjust job cluster** — [adjust\\_cluster](#)
2. Click the magnifying glass icon to open a search window in the Jobs, Nodes, or Notifications web pages.

Figure 3-8. Search Window



3. Choose which columns to display and their order by clicking the table icon and updating the selections in the window.

**Figure 3-9. Reorder or Hide Columns Window**



4. Use the menu items in the left pane of the CalCM dashboard to view the Pending Jobs and Finished Jobs pages.

The columns in these web pages provide information on pending job status, job statistics, cluster information, notifications, and exit status along with other job specific information.

- In the Pending Jobs page, if you select the check box next to the job number in the first column, the Action filter is also activated, and you can choose from a list of possible command actions that apply to the pending jobs.

## Deploying and Running CalCM

### Executing Calibre Jobs to Run Under CalCM with the Dashboard

**Figure 3-10. Pending Jobs**

- In the Finished Jobs page, the exit status, run times, modes, primary (master) and remotes, notifications, and cluster information is displayed for the completed jobs. You can select the check box next to a job number in the first column to display the information for a specific job in a plot format.

**Figure 3-11. Finished Jobs**

**Figure 3-12. Finished Job Information**

5. You can execute jobs to run under CalCM by first specifying your working directory in a shell.

```
CALCM_HOME/jobqueue
```

This directory should contain the calcm\_submit\_job script, which is automatically created when you start the CalCM daemon.

6. Enter the following command to submit job1:

```
./calcm_submit_job $PWD/job1/job.conf
```

7. In the browser window **Jobs** menu item, select the **Active** and **Pending** menu options and verify that job1 appears in the Pending Jobs or the Active Jobs category.

Depending on various factors, such as the job configuration or resource status, it may take a few minutes for the browser to update and display the job.

8. In a shell, enter the following command to submit job2:

```
./calcm_submit_job $PWD/job2/job.conf
```

9. In the browser window **Jobs** menu item, select the **Active** and **Pending** menu options and verify that job2 appears in the Pending Jobs or the Active Jobs category.

10. Select the check box next to the job number in the first column to view the current state of a job.

## Results

You have reviewed the CalCM dashboard job web pages and executed two jobs to run under CalCM. The dashboard web application updates to display the status of the jobs.

# Monitoring CalCM Notifications with the Dashboard

This procedure describes how to use the Notifications web page in the CalCM dashboard application to view notifications and information for your running jobs.

## Prerequisites

- The CalCM daemon (calcmd) and the CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Jobs must be running under CalCM as described in “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.
- Notification filters are configured by your CalCM administrator in the [calcm\\_notification\\_app.tcl](#).

## Procedure

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (*calcmd.conf*). For example,

```
DASHBOARDPORT = 9902
```

enter in the browser,

```
http://server_name:9902
```

2. Select the **Notifications** menu item in the left pane of the dashboard web page.
3. Review the notifications shown for each job ID in the Message column. The notifications are job specific and matched to the logged-in user.

**Figure 3-13. Notifications — Job ID Filtered Messages**

Date	Type	Job	Node	User	Message
02/17/20 23:...	Job	474			job(474): tag(demand):The demand was less than 0 for 10 update cy...
02/17/20 23:...	Job	474			job(474): expr [nemo:job_number_of_masters \$[jobObj] > 0 -> expr 1...
02/17/20 23:...	Job	474			job(474): tag(long_op): The present operation is ongoing for 7 hour...
02/17/20 23:...	Job	474			job(474): tag(notUpToDate): The job has not responded for 10 update...
02/17/20 23:...	Job	475			job(475): tag(demand):The demand was less than 0 for 10 update cy...
02/17/20 23:...	Job	475			job(475): expr [nemo:job_number_of_masters \$[jobObj] > 0 -> expr 1...
02/17/20 23:...	Job	475			job(475): tag(long_op): The present operation is ongoing for 7 hour...
02/17/20 23:...	Job	475			job(475): tag(notUpToDate): The job has not responded for 10 update...
02/17/20 23:...	Job	476			job(476): tag(demand):The demand was less than 0 for 10 update cy...
02/17/20 23:...	Job	476			job(476): expr [nemo:job_number_of_masters \$[jobObj] > 0 -> expr 1...
02/17/20 23:...	Job	476			job(476): tag(long_op): The present operation is ongoing for 7 hour...
02/17/20 23:...	Job	476			job(476): tag(notUpToDate): The job has not responded for 10 update...

4. Click a job ID to display the job detail page.

**Figure 3-14. Notifications — Job ID Resource Plot and Information**



A Notifications table with node names and messages is below the chart area along with the Masters (primaries) and Remotes information tables.

Notifications		
Date	Node Name	Message
02/17/20 23:24:05	-	job(475): tag(demand):The demand was less than 0 for 10 update cycles (-358).
02/17/20 23:24:05	-	job(475): expr [nemo::job_number_of_masters \$jobObj] > 0 -> expr 1 > 0
02/17/20 23:24:05	-	job(475): tag(long_ops): The present operation is ongoing for 7 hours with under 10 cores.
02/17/20 23:24:05	-	job(475): tag(notUpdated):The job has not responded for 10 update cycles.
02/17/20 23:24:05	[REDACTED]	master([REDACTED]): tag(memory): The free memory is less than 100000 mb.

Masters																
HDB	Node	H/W Mon	PID	CPU Tim	Heap Usr	Heap Allt	Heap Mi	Operation	Progre	Thread	Queue Cr	Queue Str	OP I/O Rx	OP I/O Se	Tot I/O R	Tot I/O Se
0	[REDACTED]	VM	1...	00:04...	351	361	361	prime(DRC:11)	50%	12	318	0	0	0	0	0

Remotes									
HDB	Node	H/W Monitoring	PID	CPU Time	Heap Use	Heap Alloc	Heap Max	Tot I/O Recv	Tot I/O Sent
0	node10002	cloud	123680...	143d 03:33:20	0	0	0	0	0
0	node10003	clust1	123680...	143d 03:33:21	0	0	0	0	0

## Results

You have now used the Notifications web page in the CalCM dashboard web application to view the notifications and information for your jobs.

## Monitoring CalCM Resources with the Dashboard

This procedure describes how to use the CalCM dashboard Nodes web pages to view primary and remote resource information collected by the Resource Manager application. Using this information, you can review how resources are allocated for CalCM jobs and make adjustments to the resources, as needed.

### Prerequisites

- The CalCM daemon (`calcmd`) and the CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Jobs must be running under CalCM as described in “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.

### Procedure

- Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (`calcmd.conf`). For example,

```
DASHBOARDPORT = 9902
```

enter in the browser,

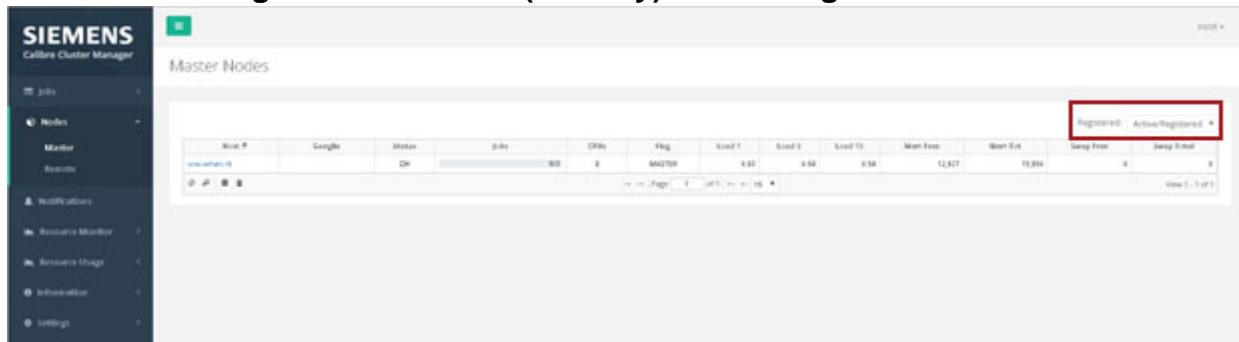
```
http://server_name:9902
```

- Click the **Nodes** menu item in the left pane.

This accesses the Master Nodes (primary nodes) and Remote Nodes information and displays the host, ganglia, status, jobs, CPUs, flag, load, and memory information for the primary and remote nodes that are available for CalCM jobs.

You can use the Registered filter in the upper right corner to view all resources or only those that are registered and active. If a node is unregistered, in order to use the node for a CalCM job, you must register it using the [register\\_node](#) command.

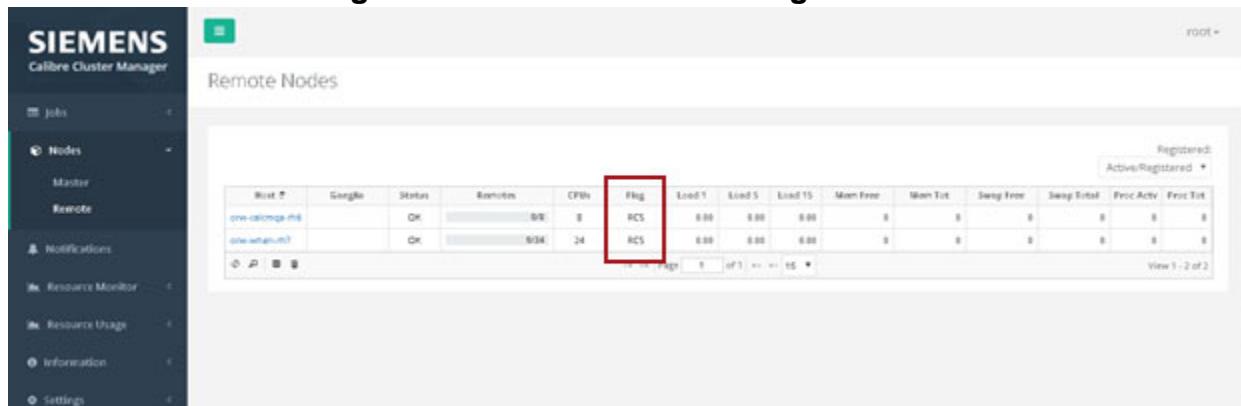
**Figure 3-15. Master (Primary) Nodes Registered Filter**



The Flag column displays the following values to indicate the node status:

- UNKNOWN — Identifies an unregistered node. In order to use this node for a CalCM job, you must register it using the [register\\_node](#) command.
- MASTER — Identifies a primary node. The primary node is assigned by the [MASTER HOST](#) statement in the CalCM cluster configuration file.
- RCS — Identifies a Remote Compute Server (RCS) node. An RCS node is assigned by the [REMOTE HOST](#) in the CalCM cluster configuration file.
- RDS — Identifies a Remote Data Server (RDS) node. An RDS node is assigned by an [RDS HOST](#) in the CalCM cluster configuration file.

**Figure 3-16. Remote Nodes Flag Column**



Host #	Google	Status	Remote	CPIs	Flag	Load 1	Load 5	Load 15	Mem Free	Mem Tot	Swap Free	Swap Total	Proc Actv	Proc Tot
one-calcme.mil		OK	8/36	8	RCS	0.00	0.00	0.00	0	0	0	0	0	0
calcme.mil		OK	8/36	24	RCS	0.00	0.00	0.00	0	0	0	0	0	0

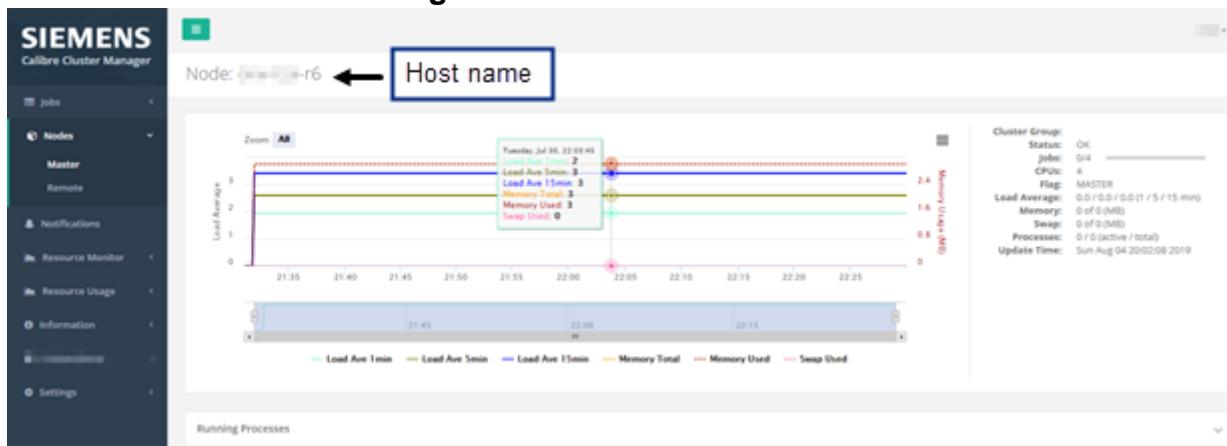
3. Click the name of a host in the Host column of the Master Nodes (primary nodes) page.

**Figure 3-17. Master (Primary) Nodes Host Column**



Host #	Google	Status	Remote	CPIs	Flag	Load 1	Load 5	Load 15	Mem Free	Mem Tot	Swap Free	Swap Total	Proc Actv	Proc Tot
one-calcme.mil		OK	8/36	8	MASTER	0.00	0.00	0.00	12,827	13,899	0	0	0	0

This displays detailed information for the selected the host.

**Figure 3-18. Host Information**

## Monitoring Jobs Running Under CalCM with the Dashboard

This procedure describes how to use the Resource Monitor web pages in the CalCM dashboard application to view resource, license, and job utilization information collected by the resource monitoring application.

### Prerequisites

- The CalCM daemon (`calcmd`) and the CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Jobs must be running under CalCM as described in “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.

### Procedure

- Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (`calcmd.conf`). For example,

```
DASHBOARDPORT = 9902
```

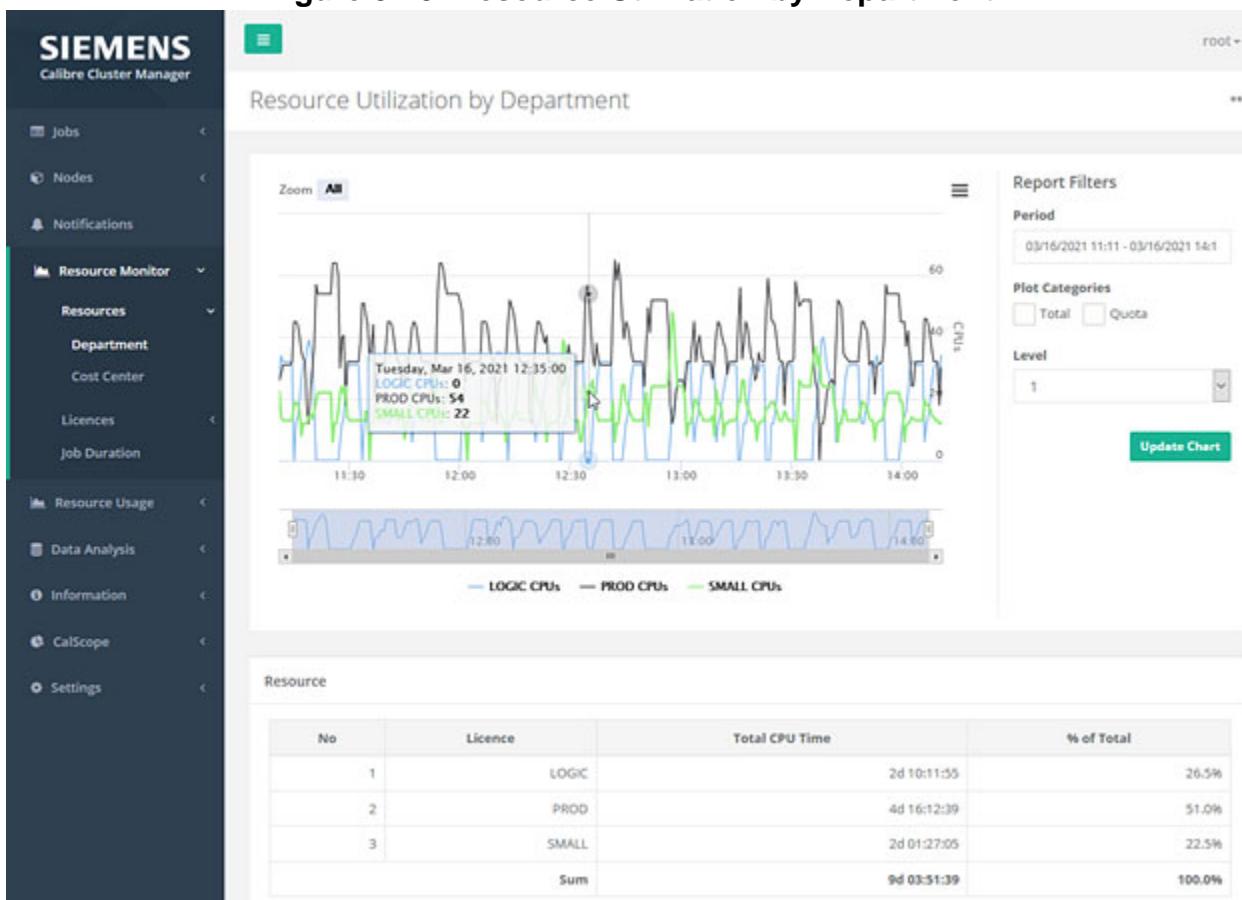
enter in the browser,

```
http://server_name:9902
```

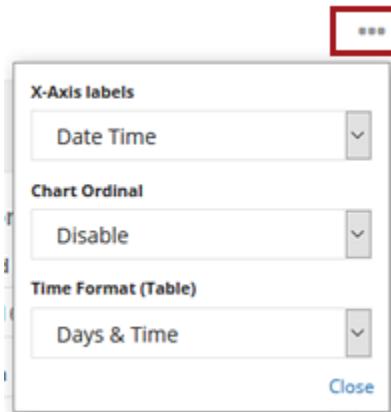
- Select the **Resource Monitor** menu item.

This accesses the Resource Monitor web pages. The left pane displays menu items and the right pane displays report information according to the menu item selected in the left pane. The default view is Resource Utilization by Department.

**Figure 3-19. Resource Utilization by Department**



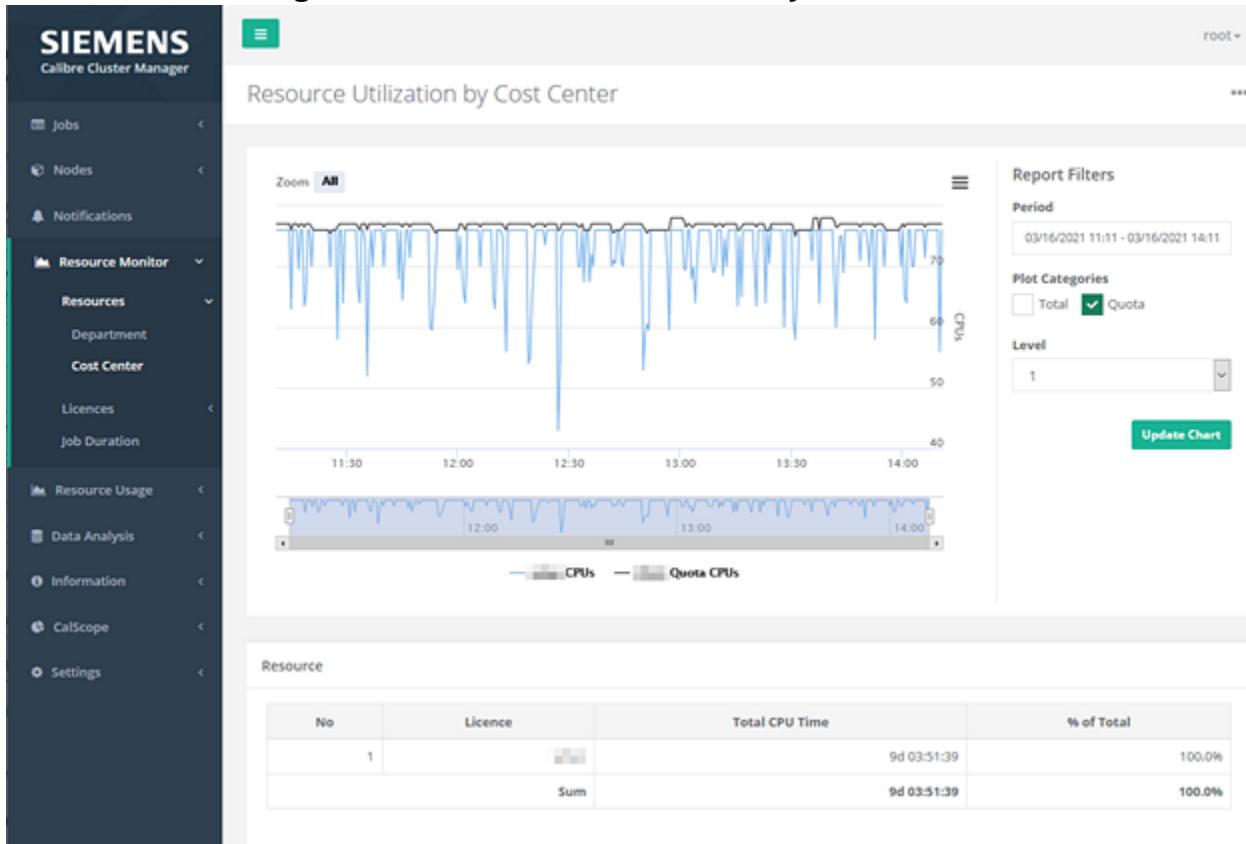
3. In the upper-right right, click the three dots to display the chart options, which include X-Axis labels, Chart Ordinal, and Time Format (Table) settings.



4. Scroll down in the window to view information and try out the controls that are available for the Resource Utilization by Department page.
5. In the left pane, select the **Cost Center** menu option.

This accesses the Resource Utilization by Cost Center page, which displays the CPU usage by department and by cost center for the specified reporting period. This page includes the same options as those found on the Resource Utilization by Department page, with the exception that the Department Filters are replaced by Cost Center Filters.

**Figure 3-20. Resource Utilization by Cost Center**

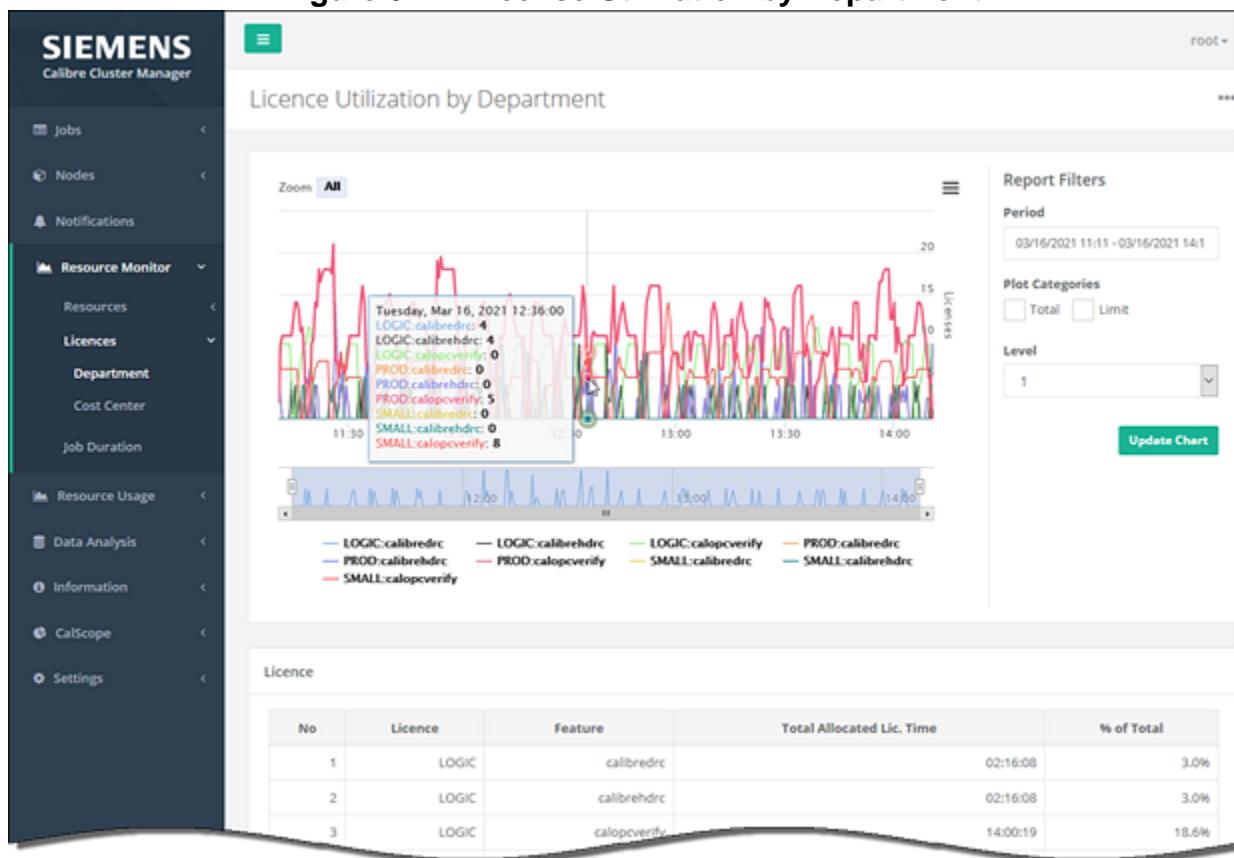


6. In the left pane, select the **Licenses** menu option.

This displays the License Utilization by Department page.

7. Scroll down in the window to view information and try out the controls that are available for the License Utilization by Department page.

**Figure 3-21. License Utilization by Department**



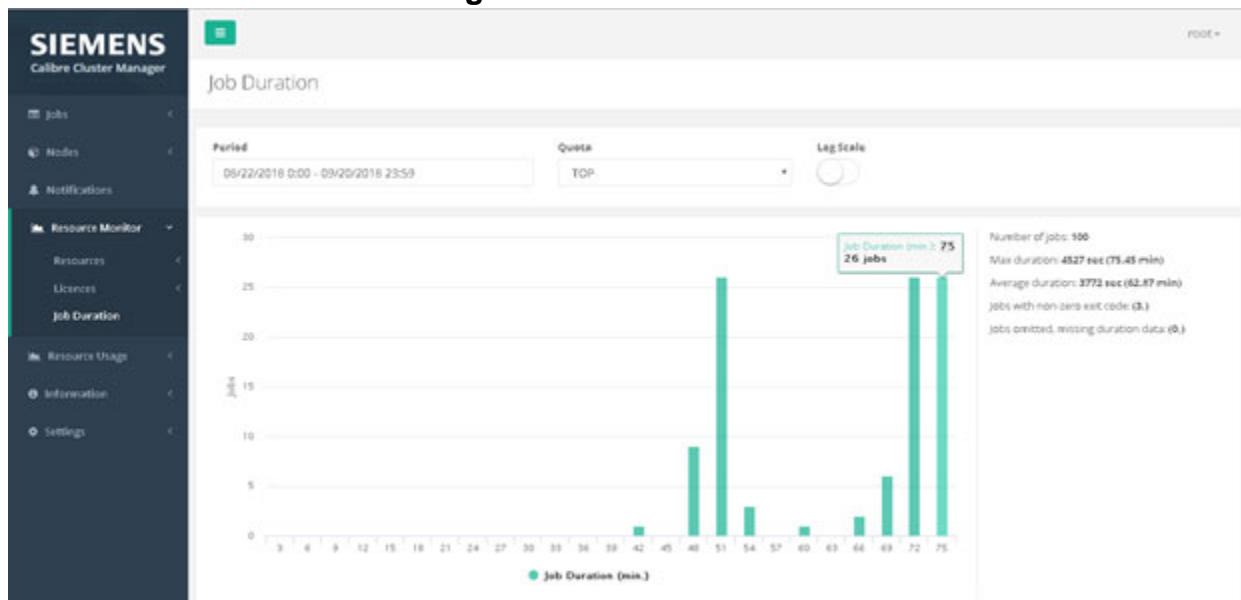
8. In the left pane, select the **Cost Center** menu option.

This displays the License Utilization by Cost Center page. This page includes the same options as those found on the License Utilization by Department page, with the exception that the Department/Feature Filters are replaced by Cost Center/Feature Filters.

9. In the left pane, select the **Job Duration** menu option.

This displays the Job Duration page and a histogram plot showing the distribution of the duration of finished jobs.

**Figure 3-22. Job Duration**



To the right of the plot, the following information is displayed for jobs matching the report period, department, or cost center:

- Number of jobs
- Maximum job duration
- Average job duration
- Jobs with no-zero exit code
- Jobs omitted, missing duration data

10. Review and try out the following options on the Job Duration page:
  - Period — Use these controls to specify a time frame for displaying the job duration information.
  - Quota — Use to select the job quota group.
  - Log Scale — Use this control to change the display from a histogram plot to a logarithmic y-axis scale.

## Results

You have now used the Resource Monitor web pages in the CalCM dashboard web application to view the resources, license utilization, and duration for your jobs.

# Monitoring Resource Usage and Allocation with the Dashboard

This procedure describes how to use the Resource Usage web pages in the CalCM dashboard application to view the CPU allocation by job and quota for a specified report period. You can also use the Resource Usage web pages to view the license usage by feature and group.

## Prerequisites

- The CalCM daemon (`calcmd`) and the CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Jobs must be running under CalCM as described in “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.

## Procedure

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (`calcmd.conf`). For example,

```
DASHBOARDPORT = 9902
```

enter in the browser,

```
http://server_name:9902
```

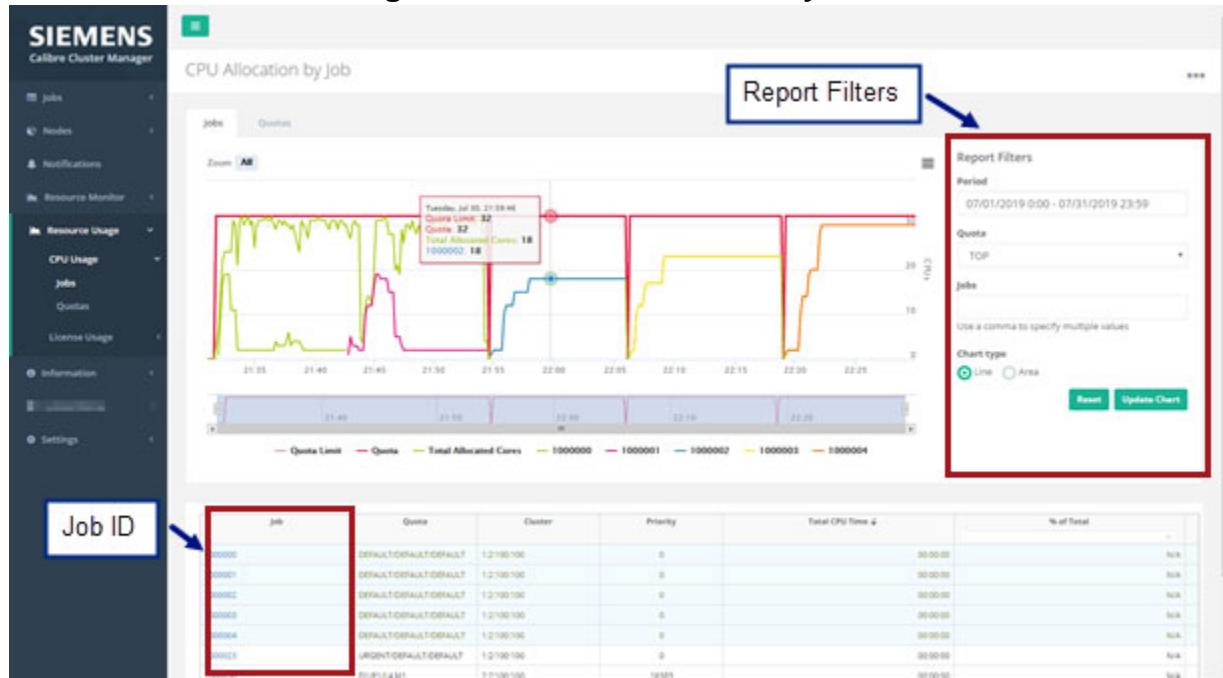
2. Select the **Resource Usage** menu item in the left pane of the dashboard.

This accesses the Resource Usage web application. The left pane displays menu items and the right pane displays a report. The default view is CPU Allocation by Job.

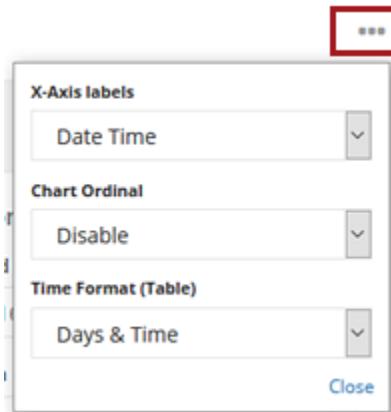
3. Scroll down in the window to view information and try out the controls that are available for the CPU Allocation by Job page.

You can use the Report Filters pane on the right-side of the web page to select the time period that you want for the displayed information. This enables you to review the resource usage information for the finished jobs.

**Figure 3-23. CPU Allocation by Job**



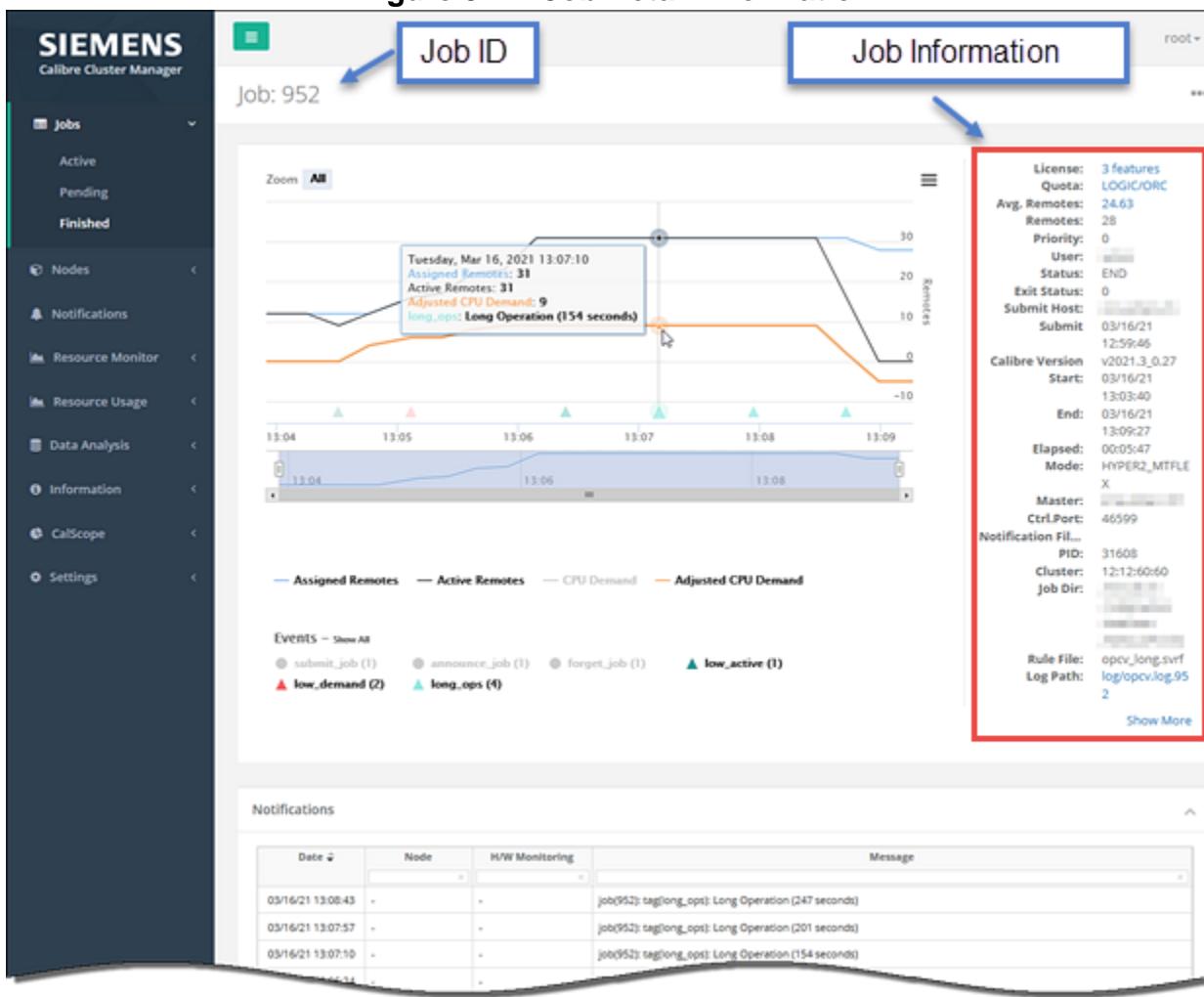
4. In the upper-right right, click the three dots to display the chart options, which include X-Axis labels, Chart Ordinal, and Time Format (Table) settings.



5. Click a job ID in the Job column of the table to open the job detail page.

The job detail page displays a plot for the selected job with the CPU demand and remote allocations for the given report period. The right-side pane of the job detail page displays detailed information for the job.

**Figure 3-24. Job Detail Information**

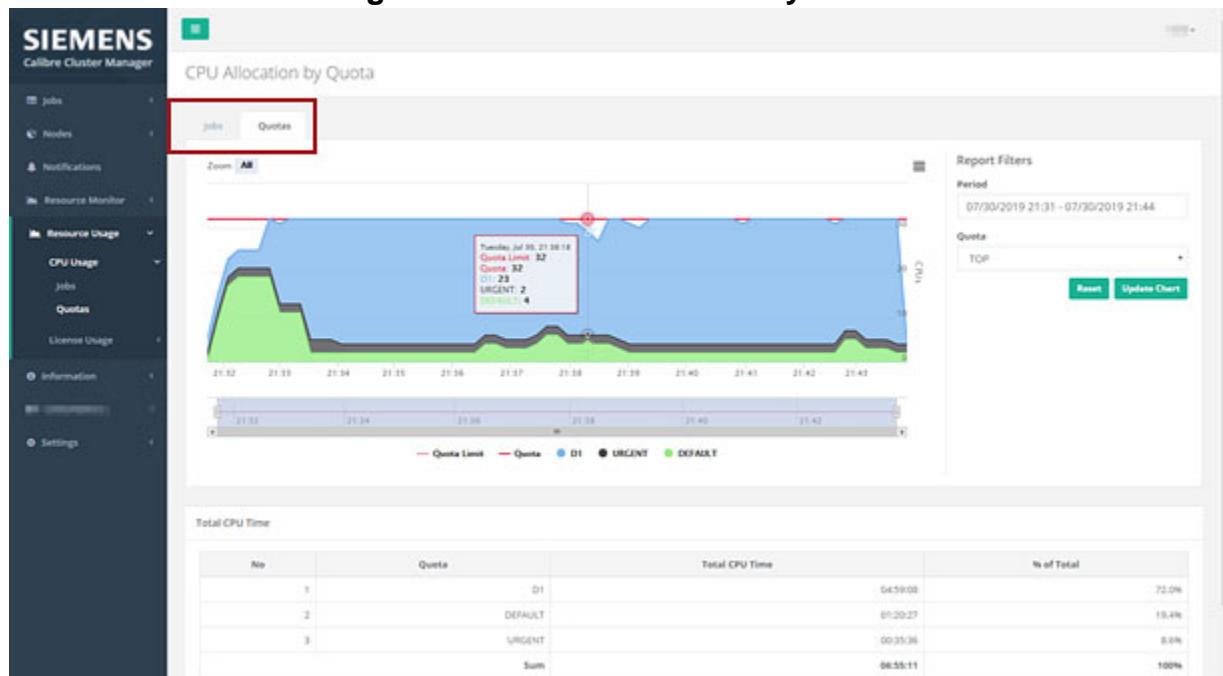


- Click on the Avg. Remotes number in the job information in right-side pane of the job detail page to display plots of CPU allocation by job and by quota.

There are also links to License, Quota, and Log Path information in this pane. You can click the Show More link at the bottom of the pane to display more information for the selected job.

- Select the **Resource Usage** menu item and then click the **Quotas** menu option to display plots of CPU allocation by quota groups and jobs for the given report period.

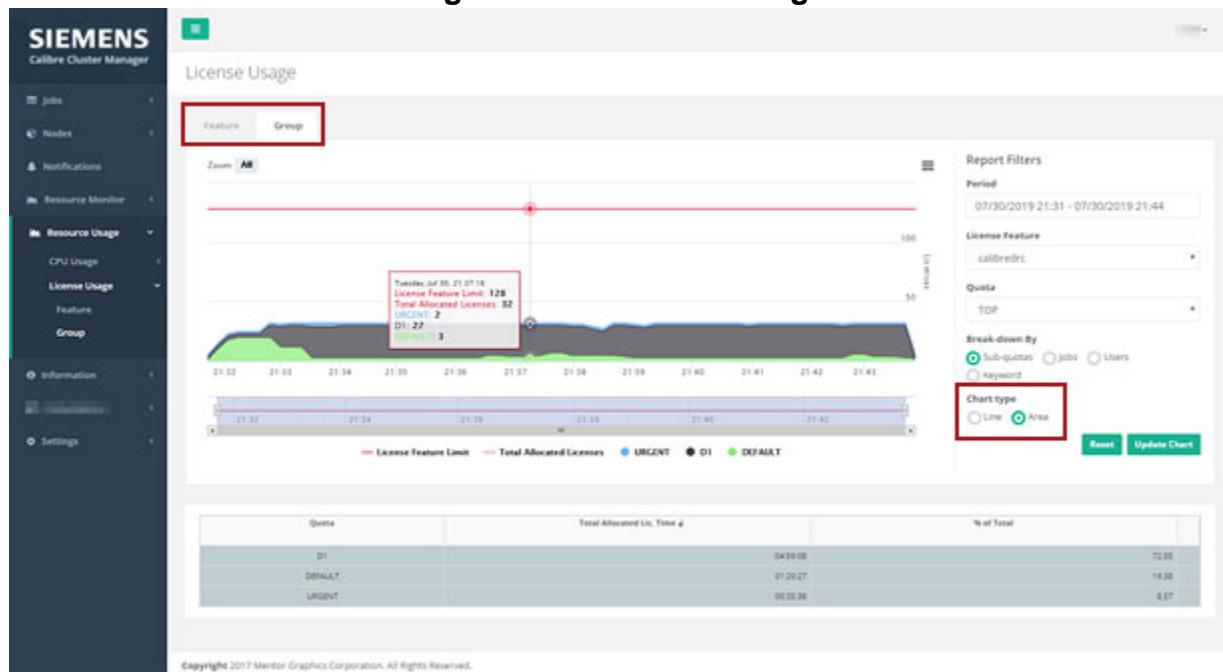
**Figure 3-25. CPU Allocation by Quota**



8. Select the **Resource Usage** menu item and then click the **License Usage** menu option to display plots of license feature and group usage for the given report period.

You can choose between Line and Area chart types to help you visualize the data.

**Figure 3-26. License Usage**



## Results

You have now used the Resource Usage web pages in the CalCM dashboard web application to view CPU allocation by job and quota along with license usage by feature and group for a specified report period.

# Displaying System Information in the CalCM Dashboard

This procedure shows how to access the system utilization and application information in the CalCM dashboard web application.

## Prerequisites

- The CalCM daemon (`calcmd`) and CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Jobs must be running under CalCM as described in “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.

## Procedure

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (`calcmd.conf`). For example,

```
DASHBOARDPORT = 9902
```

enter in the browser,

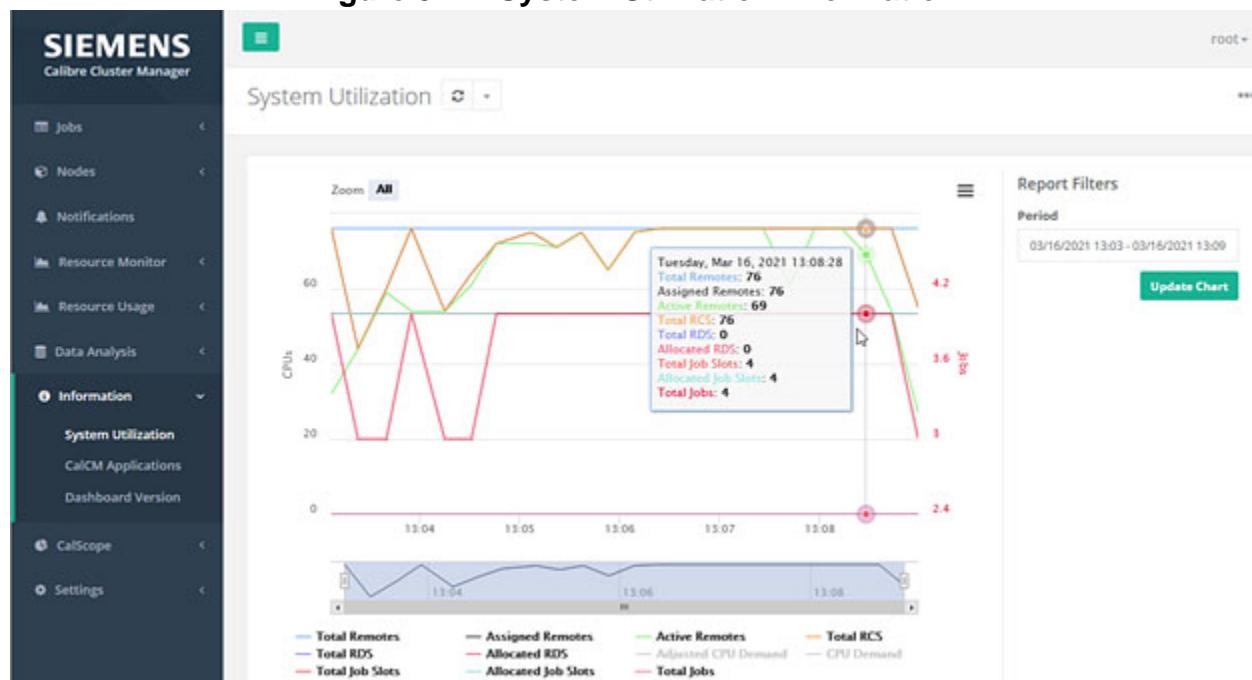
```
http://server_name:9902
```

2. Select the **Information** menu item in the left pane of the dashboard.
3. Click **System Utilization** in the dropdown menu.

This displays a plot with total CPU and RCS information along with the number of assigned and active remotes, the total number of job slots and jobs, as well as the allocated RDS and job slots. You can move the cursor over the plot lines to show the information at a specific time.

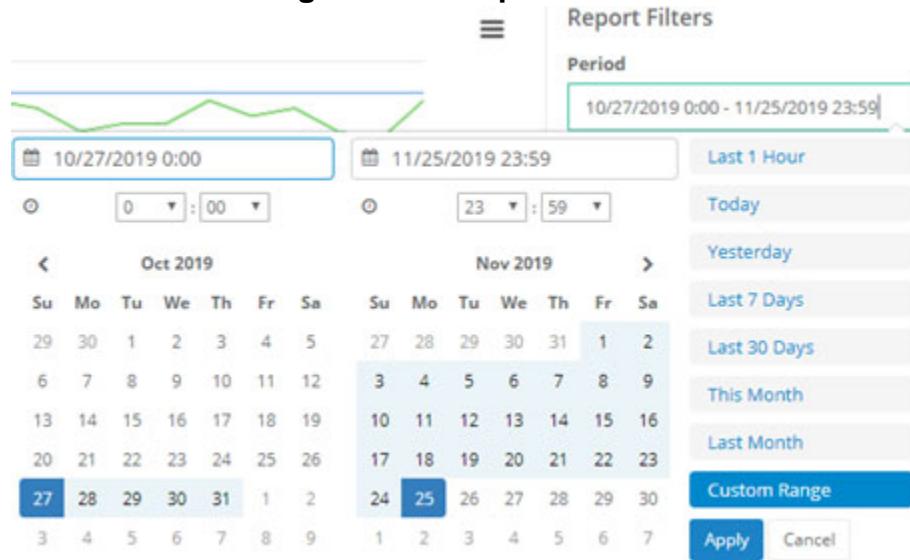
The “Chart context menu” is located in the upper right-corner of the plot area of the chart (the three-bars icon). This menu has view, print, download, and preference options for the displayed plot.

**Figure 3-27. System Utilization Information**



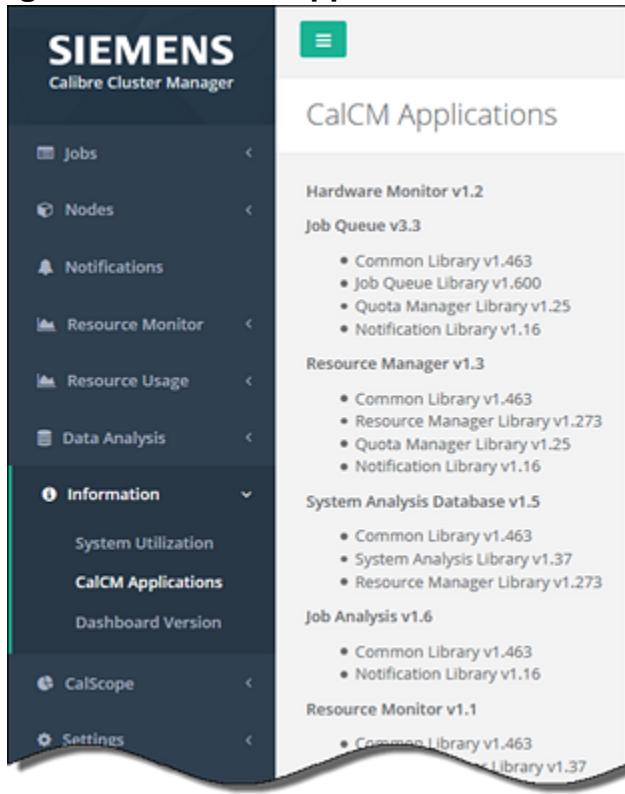
The “Report Filters” option in the far upper-right of the plot window enables you to display information for a selected time period (the default is current date and time).

**Figure 3-28. Report Filters**



4. Click **CalCM Applications** in the dropdown menu in the left pane of the dashboard. This displays a bulleted list of the CalCM application libraries and versions.

Figure 3-29. CalCM Applications Information



5. Similarly, click **Dashboard Version** to display the CalCM Web Dashboard Version information.

Figure 3-30. CalCM Dashboard Version Information



## Results

You have used the **Information** menu in the CalCM dashboard to display system utilization and CalCM applications information.

# Viewing Chart Information in the CalCM Dashboard

This procedure describes how you can use chart menu functions to customize the displayed job data in the CalCM dashboard web application.

## Prerequisites

- The CalCM daemon (`calcmd`) and CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Browser access to the CalCM dashboard web application.
- You are logged into the CalCM dashboard web application with authentication information that allows you access to the menu items on the left-side of the dashboard.

### Note

 The CalCM dashboard displays a web login page requesting authentication information (user name and password). Some menus and actionable items are only accessible at certain user levels. For more information on user authentication in the CalCM dashboard, see the LDAP-related configuration keywords in the [`calc\_http\_server\_app.tcl`](#) application.

---

## Procedure

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (`calcmd.conf`). For example,

```
DASHBOARDPORT = 9902
```

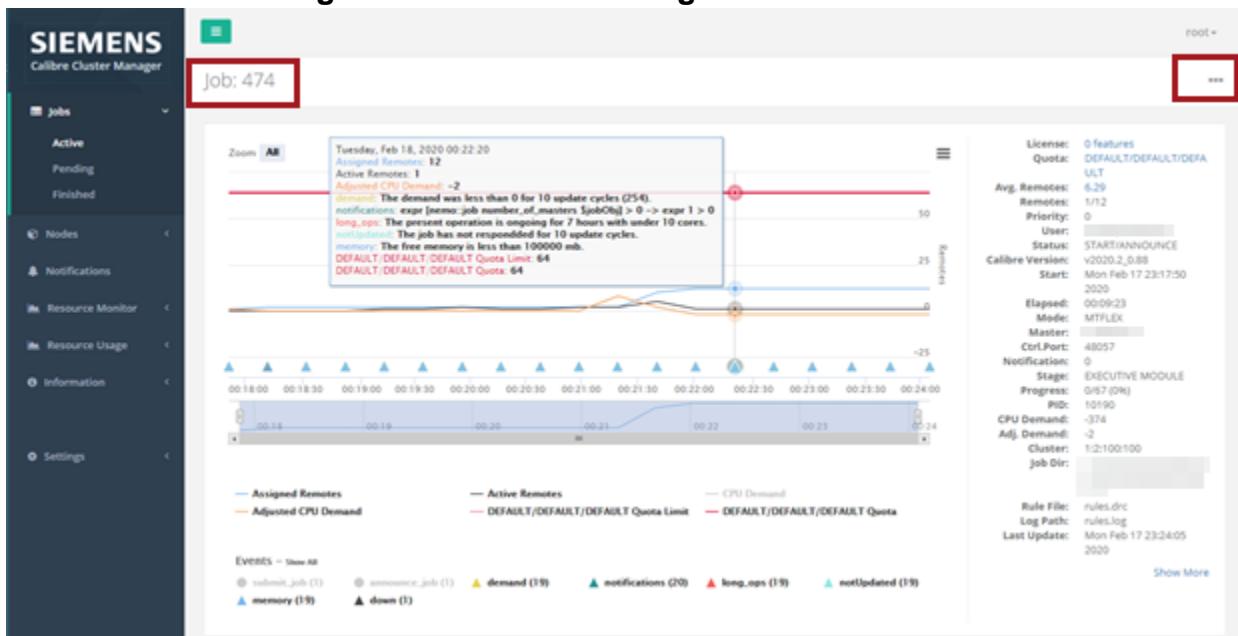
enter in the browser,

```
http://server_name:9902
```

2. In the Active Jobs page, click a job ID in the Job column of the table.

The chart information with a plot of the selected job ID appears in the job detail page.

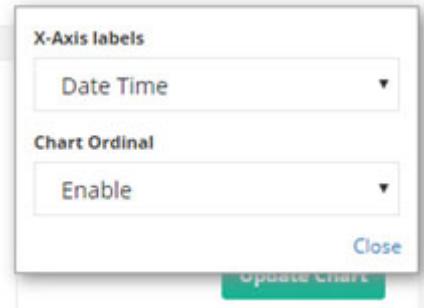
**Figure 3-31. Job Detail Page and Chart Menu**



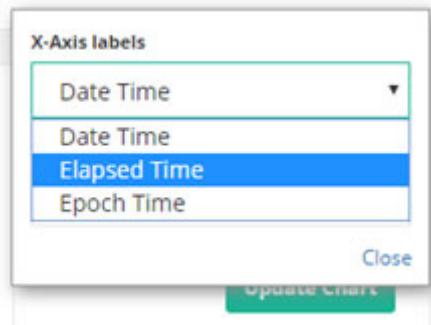
3. To set the display options for the chart, click the **Chart** menu (the three dots icon) in the upper-right corner of the job detail page.

The **Chart** menu icon only appears if the page has a chart.

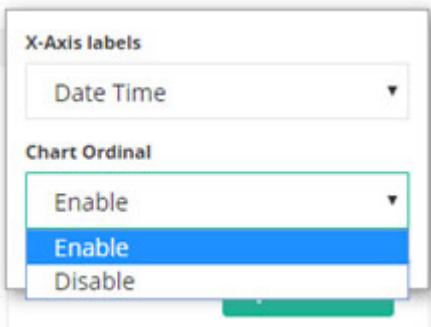
This opens a menu with **X-Axis labels** and **Chart Ordinal** menu selections.



4. In the **X-Axis labels** menu, select your preferred time format for the x-axis of the job plot. You can choose from **Date Time** (the default), **Elapsed Time**, and **Epoch Time**.



5. In the **Chart Ordinal** menu, select your preference to either **Enable** (the default) or **Disable** the ordinal x-axis display.



An ordinal axis in a chart displays regular increments for the scale. Disabling the ordinal axis can result in a more realistic display of data that accounts for shorter and longer time periods and gaps in the sampled data.

## Results

You have used the chart menu functions to set the x-axis label and scale preferences for displaying chart data in the CalCM dashboard web application. These preferences are saved with your user login for the next time you access the CalCM dashboard.

# Comparing Job Information in the CalCM Dashboard

This procedure shows you how to control the number of active job records to display and how to easily compare chart information for multiple jobs in order to analyze runtime and scalability.

## Prerequisites

- The CalCM daemon (calcmd) and CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.

- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Jobs must be running under CalCM as described in “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.

## Procedure

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (*calcmd.conf*). For example,

```
DASHBOARDPORT = 9902
```

enter in the browser,

```
http://server_name:9902
```

2. In the Active Jobs page, click the dropdown icon at the bottom-center of the table to select the number of job records to display as shown in [Figure 3-32](#). The default is 30 job records maximum per page.

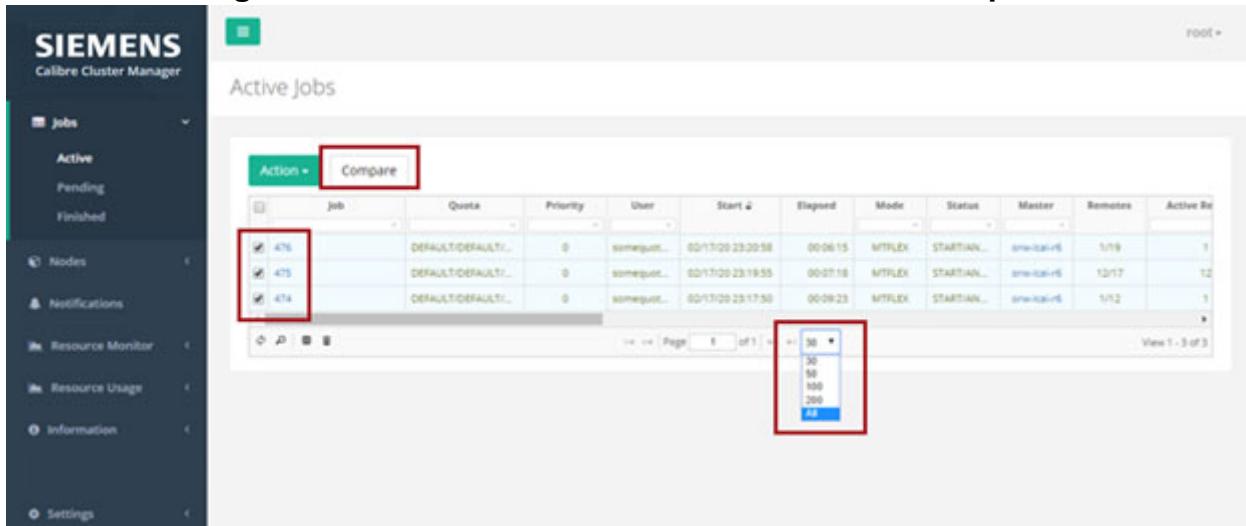
This functionality is also available in the Finished Jobs page, but it is not available in the Pending Jobs tables because of the potentially large number of records in this category.

3. To compare detailed job information for multiple jobs, choose several job IDs in the Job column of the table and click the **Compare** button.

### **Note**

The **Compare** button is available in the Active Jobs and Finished Jobs pages and is only enabled when two or more job IDs are selected. The job selections are persistent, even when the web browser back-arrow is clicked.

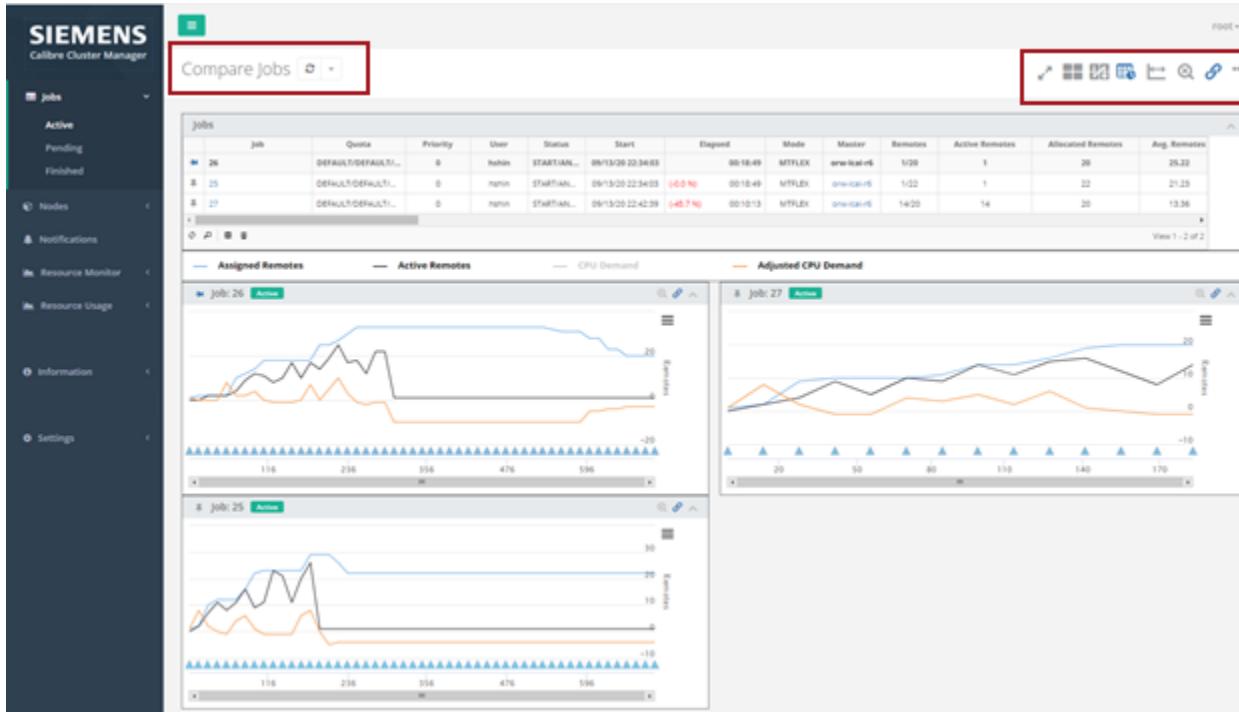
**Figure 3-32. Active Jobs Record Selection and Compare**



The screenshot shows the CalCM Active Jobs page. On the left, a sidebar menu includes 'Jobs' (selected), 'Active', 'Pending', 'Finished', 'Notes', 'Notifications', 'Resource Monitor', 'Resource Usage', 'Information', and 'Settings'. The main area is titled 'Active Jobs' and contains a table with columns: Action, Compare, Job, Quota, Priority, User, Start, Elapsed, Mode, Status, Master, Remotes, and Active. Three rows are selected, indicated by checkboxes in the 'Job' column. A red box highlights the 'Compare' button in the 'Action' column. Another red box highlights the dropdown menu for selecting the number of records per page, currently set to 30. The dropdown menu also includes options for 50, 100, 200, and All.

The Compare Jobs page opens and displays the information for the selected job IDs.

**Figure 3-33. Compare Jobs Page Functions**

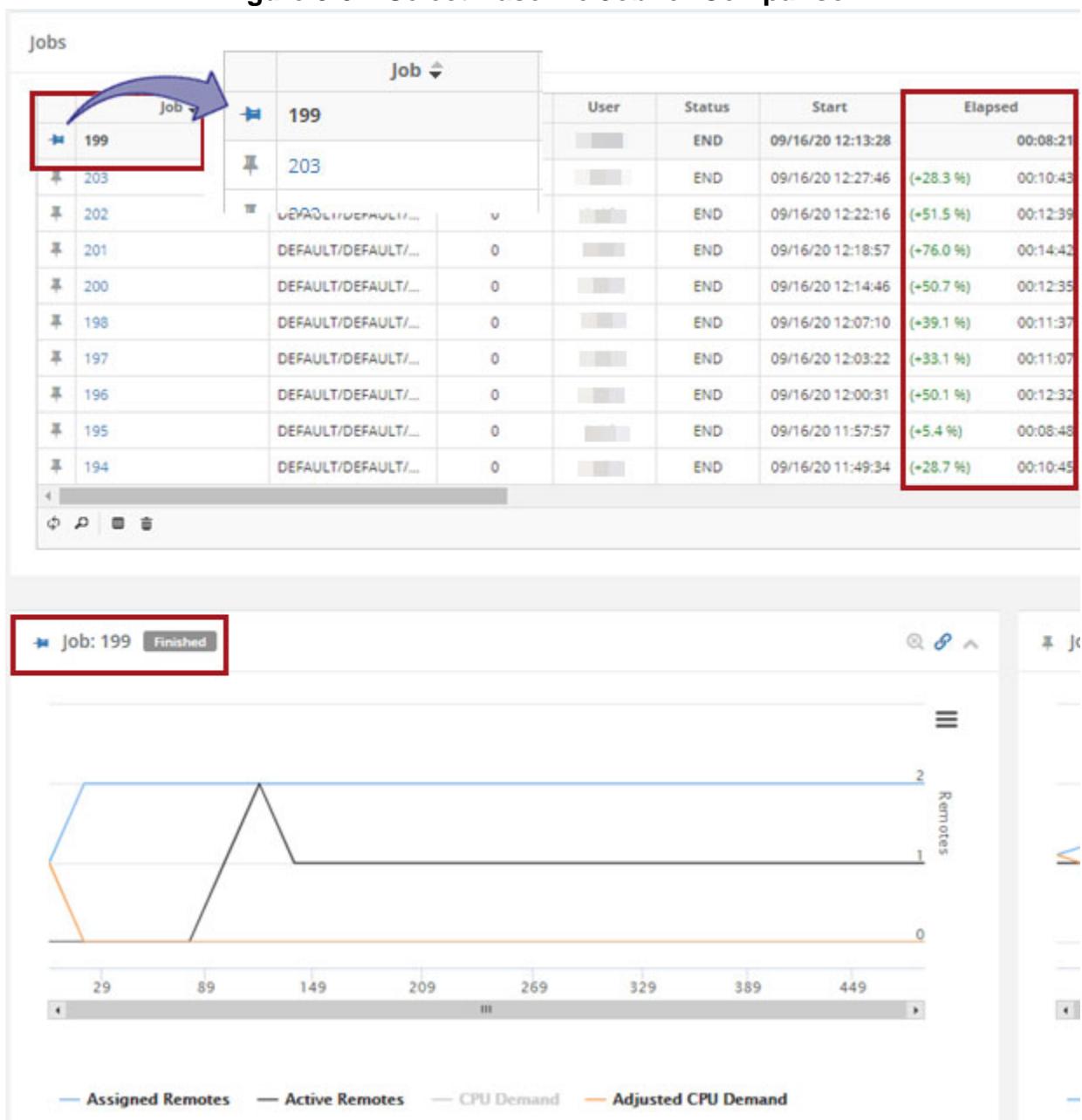


4. In the jobs table, click the pin icon (next to the job ID) to select a baseline job for comparing information across all the jobs in the table and charts.

#### **Note**

If you sort the table columns or charts, the baseline job stays “pinned” as the first table entry and the top chart position.

**Figure 3-34. Select Baseline Job for Comparison**

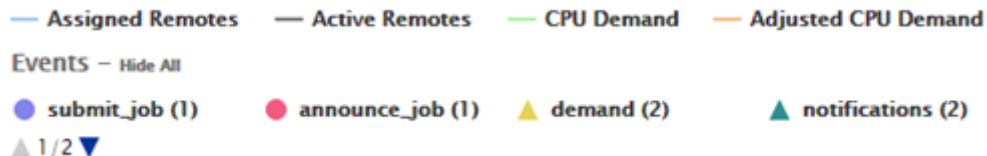


5. Control the chart features using the icons in the upper-right corner of the Compare Jobs page (shown left to right):

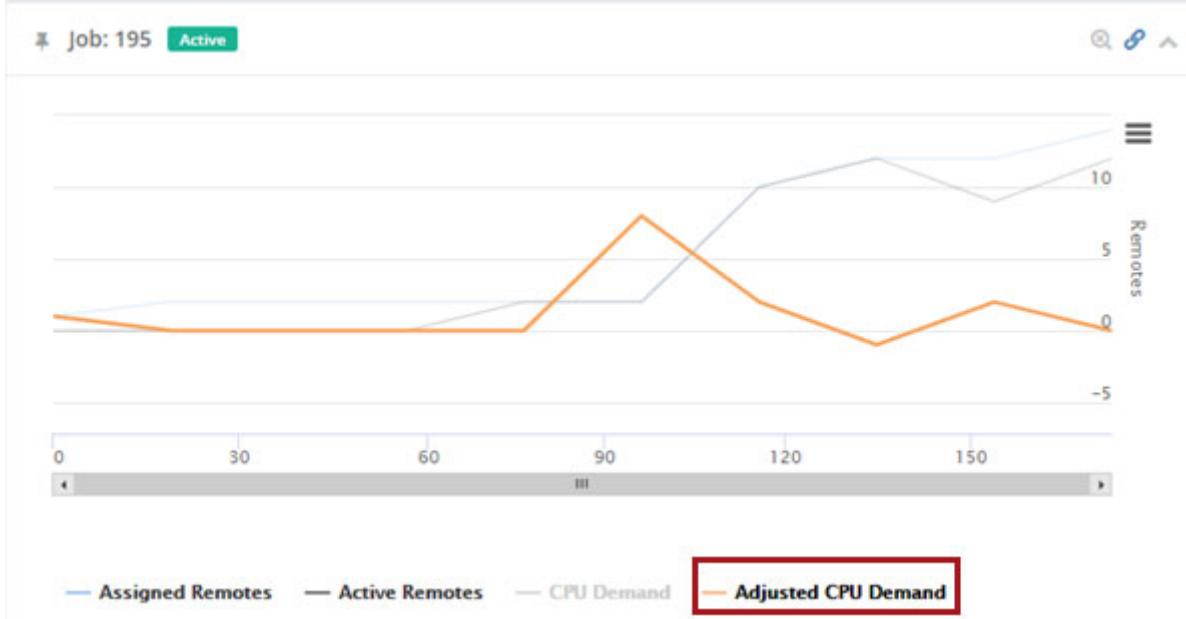


- a. **Compact view** — Toggles charts from normal to compacted view.

- b. **Switch view** — Switches the chart views between horizontally and vertically stacked orientation.
  - c. **Switch Chart Sorting** — Changes the order of the charts between horizontal and vertical ordering.
  - d. **Sync Order Between Table and Charts** — Synchronizes the sort order between the jobs table and the charts.
  - e. **Common Axis** — Toggles the display of the x- and y-axis scales for runtime and scalability comparison.
  - f. **Zoom out** — Resets the zoom (out) across all charts.
  - g. **Link Charts** — Toggles the linking functionality across all charts.
  - h. **Chart (ellipses (...))** — Access **X-axis labels** and **Chart Ordinal** menus. See “[Viewing Chart Information in the CalCM Dashboard](#)” on page 78 for menu detail.
6. Use the chart legend labels to toggle the display of plot-lines and event markers.



Hover your cursor over a plot label to highlight the corresponding plot line.



## Results

You have used table and chart functions in the CalCM dashboard to selectively compare job data and control how it is displayed.

# Refreshing Chart Data in the CalCM Dashboard

This procedure describes how to perform a live update of the page and chart information for active jobs in the Compare Jobs page in the CalCM dashboard web application.

## Prerequisites

- The CalCM daemon (`calcmd`) and CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Jobs must be running under CalCM as described in “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.

## Procedure

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (`calcmd.conf`). For example,

```
DASHBOARDPORT = 9902
```

enter in the browser,

```
http://server_name:9902
```

2. In the Active Jobs page, choose the job records to compare.
3. Click the **Compare** button to display plots for the selected records.

**Figure 3-35. Selected Job Records for Comparison**

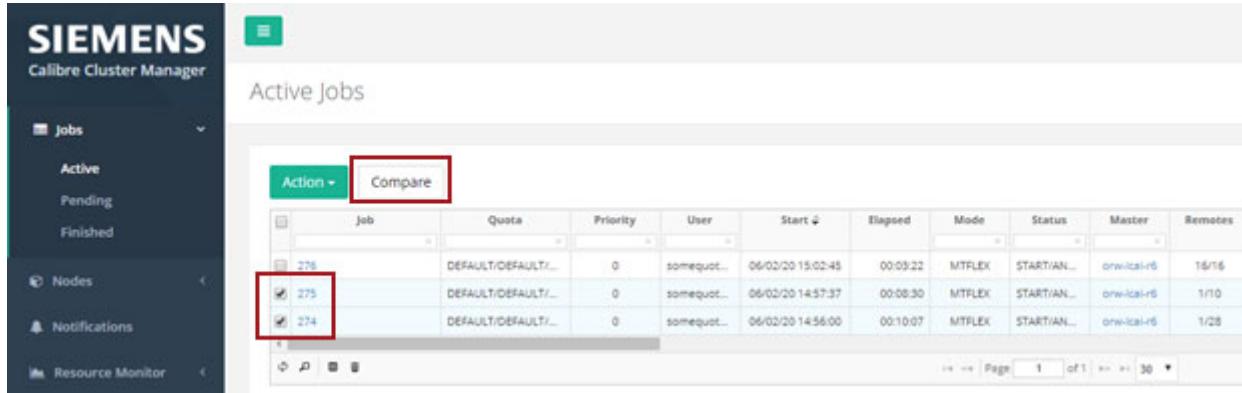
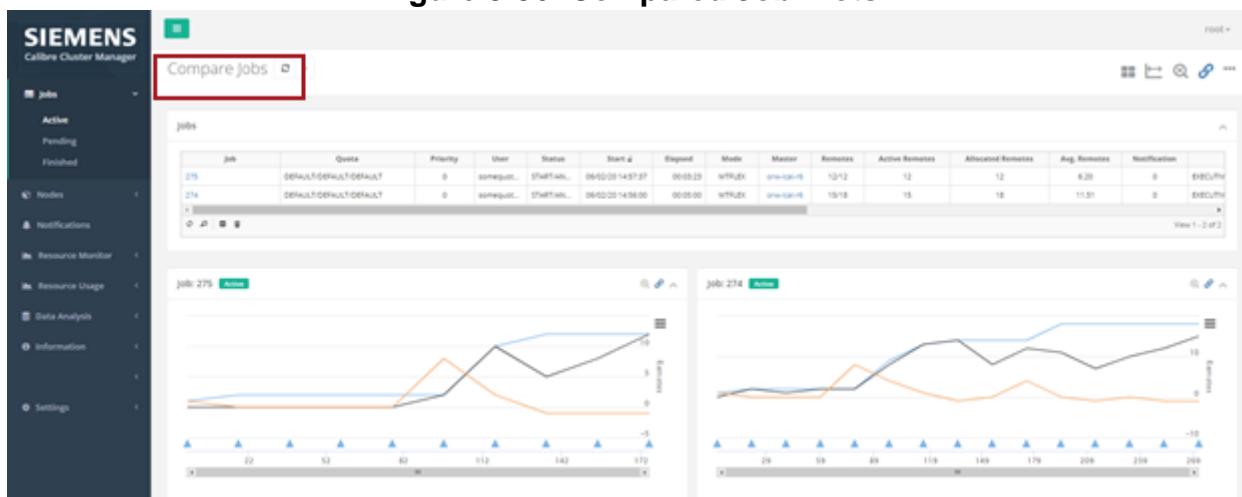


Figure 3-36. Compared Job Plots

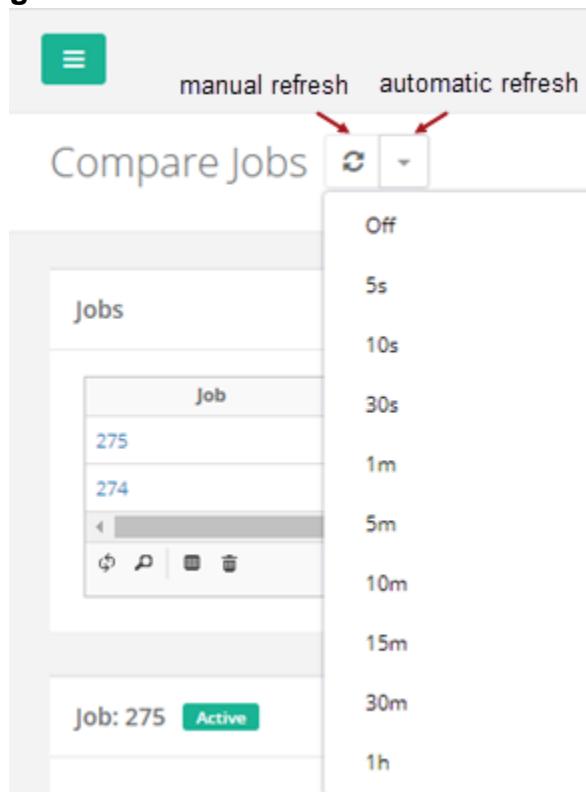


4. Refresh the chart data (without reloading the page) in one of two ways:

**Automatically** — Click the dropdown menu in the upper-left corner of the Compare Jobs page, and select the desired interval time for automatic refresh of the chart data. In the Compare Jobs page, refresh indicators appear when the page and individual charts are updating.

**Manually** — Click the **Refresh Dashboard** icon to the left of the dropdown menu to manually refresh the chart data.

Figure 3-37. Refresh Chart Data Functions



## Results

You have refreshed the chart data in the Compare Jobs page. The live update functionality is also available from the Active Jobs page and the System Utilization page.

# Accessing and Updating Settings in the CalCM Dashboard

This procedure describes how you can access and update the Settings pages in the CalCM dashboard web application.

## Prerequisites

- The CalCM daemon (calcmd) and CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Browser access to the CalCM dashboard web application.
- You are logged into the CalCM dashboard web application with authentication information that allows you access to the **Settings** menu item on the left-side of the dashboard.

**Note**

 The CalCM dashboard displays a web login page requesting authentication information (user name and password). Some menus and actionable items are only accessible at certain user levels. For more information on user authentication in the CalCM dashboard, see the LDAP-related configuration keywords in the [calcm\\_http\\_server\\_app.tcl](#) application.

**Procedure**

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (*calcmd.conf*). For example,

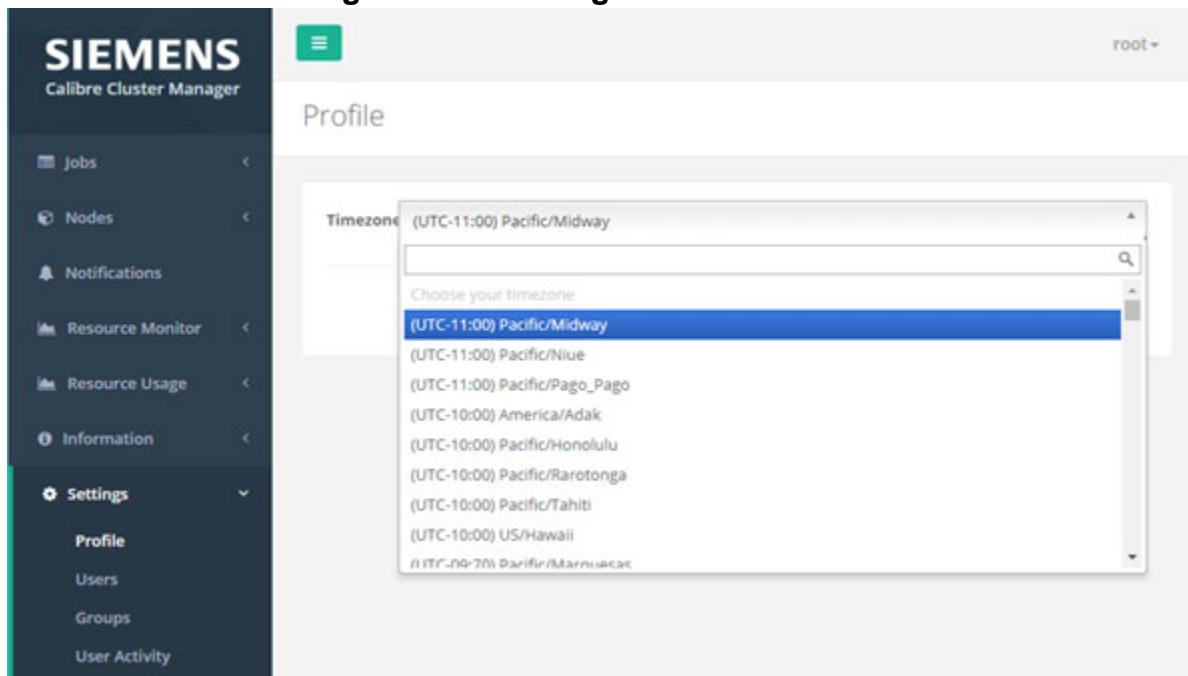
```
DASHBOARDPORT = 9902
```

enter in the browser,

```
http://server_name:9902
```

2. Select the **Settings** menu item in the left pane of the CalCM dashboard and set your preferred timezone by clicking **Profile** in the dropdown menu.

**Figure 3-38. Settings Profile Timezone**



3. Add a new user:
  - a. Click **Settings > Users**.
  - b. Click the **Add** button and fill in the user information fields.

If the “Add to Groups” field does not display the correct group, you can create a group and update the user information as described in the following step.

- c. Click **Save**.

**Figure 3-39. Settings Add New User**

Add New User

ID  
new\_user ✓  
*Username is available!*

Password      Confirm Password

First Name      Last Name

Email  
new\_user@domain.com

Add to Groups  
admin

**Save**

4. Create a new group and add a user:

- a. Click **Settings > Groups** to display the user groups and information.

**Figure 3-40. Settings Groups**

Groups

Group #	Page Access	Permission	Permission Level	Last Update
admin	all	all	TOP	11/03/19 22:23:36
anonymous	Information > System Utilization, jobs, ...			11/03/19 22:23:36
default	Information > System Utilization, jobs, ...	adjust_job_launch_order, kill job, resu...		11/03/19 22:23:36

- b. Click the **Add** button and fill in the group information fields.

Use the dropdown menu items to control the page accessibility and the types and levels of permissions for the new group.

You can also click on an existing group name and update the page access and permissions in the Edit Group window.

- c. Click **Save**.

**Figure 3-41. Settings Add New Group**

**Add New Group**

---

Name  
test\_group

Add Users to this Group  
[empty input field]

Page Accessibility  
Jobs X Notifications X Resource Monitor > Job Duration X  
Nodes X

Permission  
edit user X

Permission Level  
TOP/DEFAULT X

Cancel Save

5. Click **User Activity** to view a history of user actions:

**Figure 3-42. Settings User Activity**

The screenshot shows the 'User Activity' section of the CalCM dashboard. On the left, there's a sidebar with various navigation options like 'Jobs', 'Nodes', 'Notifications', etc., and a 'Settings' section which is currently selected. Under 'Settings', there are four sub-options: 'Profile', 'Users', 'Groups', and 'User Activity'. The main area is titled 'User Activity' and contains a table with 44 rows of data. The columns are labeled: ID, Time, User, Type, Result, Failed Reason, and Called Args. Most entries show a 'User' named 'root' performing a 'LOGIN' action at various times between 11/22/18 and 11/23/18, with a 'Result' of 'SUCCESS'. There are also several entries for 'GROUP\_DELETE', 'GROUP\_EDIT', and 'GROUP\_ADD' actions, all successful, involving the same user and group IDs mentioned in the table.

ID	Time	User	Type	Result	Failed Reason	Called Args
44	11/22/18 18:42:37	root	LOGIN	SUCCESS		
45	11/22/18 17:48:55	root	LOGIN	SUCCESS		
42	11/22/18 17:08:06	root	LOGIN	SUCCESS		
41	11/22/18 16:47:09	root	LOGIN	SUCCESS		
40	11/22/18 16:33:38	root	LOGIN	SUCCESS		
38	11/22/18 15:53:48	root	LOGIN	SUCCESS		
39	11/22/18 15:35:52	root	LOGIN	SUCCESS		
37	11/22/18 15:18:53	root	GROUP_DELETE	SUCCESS		{group_id: 'test_group'}
36	11/22/18 15:19:19	root	GROUP_EDIT	SUCCESS		{group_id: 'test_group', 'permissions': ['perform_edit_user']}
35	11/22/18 15:12:53	root	GROUP_ADD	SUCCESS		{group_id: 'test_group', 'permissions': ['perform_edit_user']}
34	11/22/18 15:09:34	root	LOGIN	SUCCESS		
39	11/22/18 14:52:21	root	LOGIN	SUCCESS		
32	11/22/18 11:49:51	root	LOGIN	SUCCESS		
31	11/22/18 11:01:49	root	LOGIN	SUCCESS		
30	11/22/18 10:23:02	root	LOGIN	SUCCESS		

## Results

You have accessed and updated the user and group settings in the CalCM dashboard and viewed the record of user activities.

# CalCM+ Advanced Features for Data Analysis in the CalCM Dashboard

---

With the Calibre Cluster Manager Plus (CalCM+) advanced features licenses, you can perform statistical analysis of job data and cluster utilization in the CalCM dashboard.

---

## Note

---

 CalCM+ advanced features licenses are required to enable the data analysis functionality in the CalCM dashboard. Refer to the “[Calibre Cluster Manager Plus \(CalCM+\)](#)” section in the *Calibre Administrator’s Guide* for complete licensing information.

---

The CalCM+ advanced features provide capabilities beyond the basic CalCM functionalities to enable data analysis from high-level to detailed information. This enhanced functionality gives insight for job administrators, tapeout engineers, rule developers, and other design team members to identify potential run issues. You can choose the parameters for filtering job and cluster information for the specified time period. The aggregated results of these parameters display as plots and table information that can be used to analyze statistical job data and cluster utilization.

You can view the resulting plots and tables of the aggregated statistics from the **Data Analysis** menu item. This menu provides access to the Job Statistics and Cluster Utilization pages of the CalCM dashboard.

- **Job Statistics Page** — Displays the aggregate values of filtered parameters such as technology, layer, module, keyword value, and more. The chart format makes it easy to identify average values and outliers relevant to runtime performance, maximum memory, and run scalability.

Job data comes from the JOB INFO data provided with each CalCM job, such as tech\_node, layer, module, and so on. Refer to “[JOB INFO](#)” on page 236 for details.

- **Cluster Utilization Page** — Displays the aggregate values of filtered parameters such as module, layer group, mask set, keyword, keyword version, and more. The color-coded plots and charts provide information for the total and allocated remotes and the number of CPUs.

<b>Using Job Statistics for Data Analysis . . . . .</b>	<b>92</b>
<b>Using Cluster Utilization Information for Data Analysis . . . . .</b>	<b>98</b>

## Using Job Statistics for Data Analysis

As part of the Calibre Cluster Manager Plus (CalCM+) advanced features, you can perform data analysis on your jobs by displaying the statistical data in an aggregated plot and tabular format for a specified period.

## Restrictions and Limitations

- CalCM+ advanced features licenses are required to enable the data analysis functionality in the CalCM dashboard. See the [Calibre Administrator's Guide](#) for complete licensing information.

## Prerequisites

- The CalCM daemon (`calcmd`) and CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Jobs must be running under CalCM as described in “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.
- The following CalCM applications must be enabled in the `calcmd.conf` configuration file:
  - `calcml_jobanalysis_app.tcl`
  - `calcml_notification_app.tcl` (This Notification application is responsible for the remote parsing of the Calibre log file from the Calibre execution host. This distributed architecture avoids any bottleneck on the CalCM server.)
  - `calcml_jobqueue_app.tcl`
- In the `calcml_jobqueue_app.tcl` application, the parameter `SCANJOBINFO` must be set to 1. This enables CalCM to scan the Calibre output transcripts to gather detailed job information and record it to the CalCM database.

## Procedure

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (`calcmd.conf`). For example,

```
DASHBOARDPORT = 9902
```

enter in the browser,

```
http://server_name:9902
```

2. Log into the CalCM dashboard web application with your authentication information and verify that you have access to the **Data Analysis** pages in the Groups settings. See “[Accessing and Updating Settings in the CalCM Dashboard](#)” on page 87.
3. In the left-side menu of the CalCM dashboard, click the **Data Analysis** menu item.
4. Select Job Statistics to open the page in the CalCM dashboard.

**Figure 3-43. Job Statistics Page**

- In the Job Statistics page, click inside the Period field to select the date range for your job data for your analysis and click **Apply**.

Choose the time period that is specific to your job data. A message appears warning that the data may take longer to load if the period is over one month.

**Figure 3-44. Job Statistics Period**

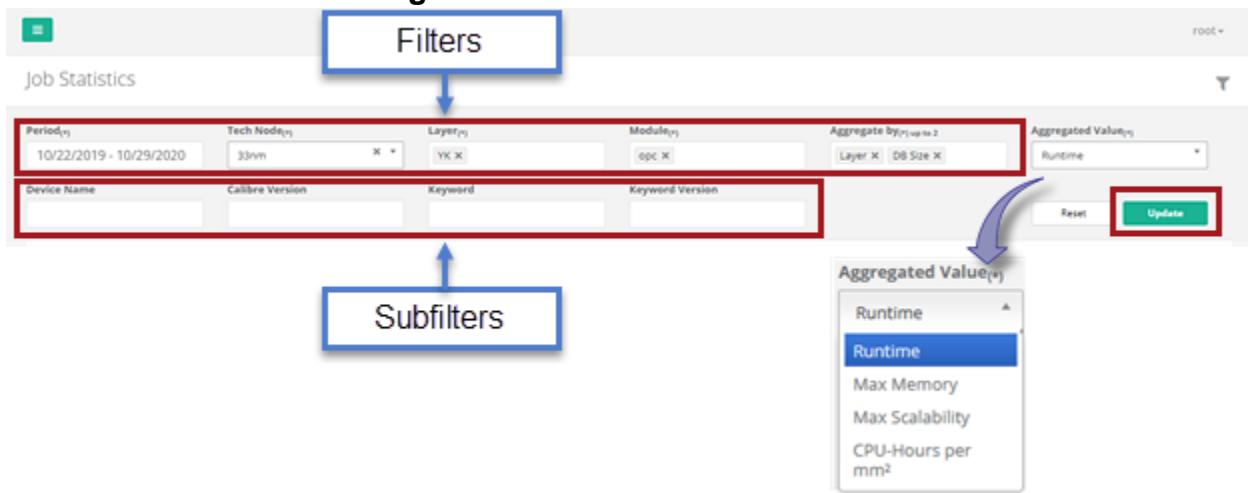
#### **Note**

Fields with an asterisk ( \* ) denote required fields.

- Filter the data for your job analysis at the top of the Job Statistics page by selecting field entries for the Tech Node, Layer, Module, and Aggregate by up to 2. You can optionally select subfilter fields for Device Name, Calibre Version, Keyword, and Keyword Version for additional refinement.

Analyze the job data for different scenarios by choosing from Runtime, Max Memory, and Max Scalability in the Aggregated Value field. For example, a tapeout engineer with a long-running job may filter and aggregate the data by layer and database size and then analyze the resulting runtime value. Another scenario may require the analysis of maximum memory, maximum scalability, or an aggregate of these values.

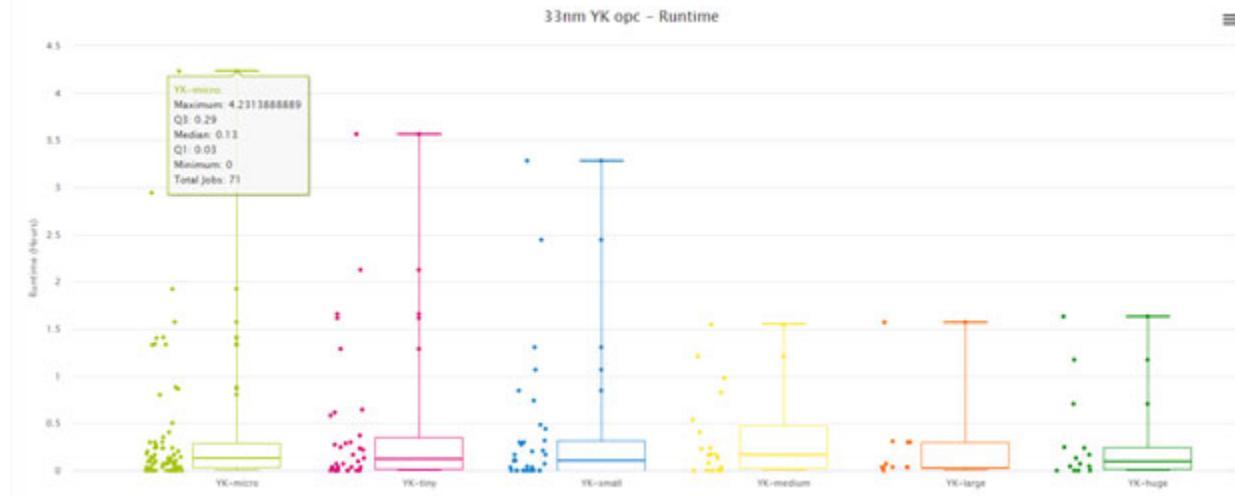
**Figure 3-45. Job Statistics Filters**



- Click **Update** to display a box and whiskers plot of the filtered job data.

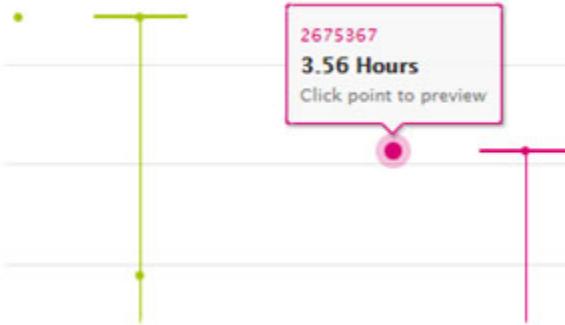
In this style of plot, you can see the spread of the data set by easily identifying cluster centers (boxes) and outliers (whiskers). Hovering your cursor over a particular box and whisker set in the plot displays the Minimum, Maximum, and Mean values along with the Total Jobs for that data set.

**Figure 3-46. Job Statistics Plot**



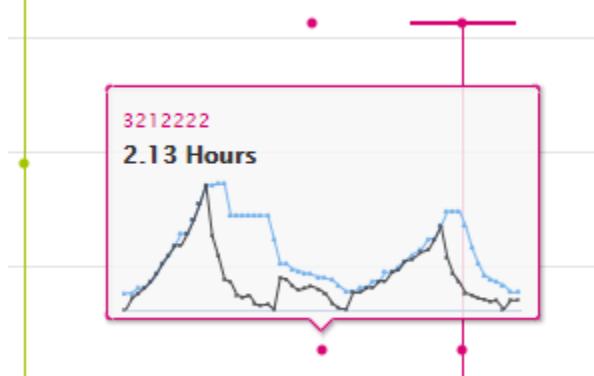
- Hover your cursor over an outlier data point to see the job ID and value (for example, the runtime) for that point.

**Figure 3-47. Job Statistics Data Point Value**



9. Click a data point to preview a time-series plot for the job ID. You can view the associated job detail page by clicking a plot line.

**Figure 3-48. Job Statistics Data Point Plot**



The information for each job ID also displays in a tabular format in the Filtered Jobs table located below the main plot window. You can click a job ID in the table to open the associated job detail page.

**Figure 3-49. Job Statistics Filtered Jobs Table**

Filtered jobs													
Job	Submit Time	Start Time	Device Name	Node	Tech Node	Module	DB Area	Categorized DB Size	Layer	Layer Revision	Layer Group	Run time	
7246145	08/10/20 18:37:31	08/10/20 18:38:31	Eigra	0	33nm	opc	728.00	huge	VK	3	middleend	0.00	
4896643	08/10/20 17:38:13	08/10/20 17:38:13	WeissenburginBayern	0	33nm	opc	87.00	small	VK	1	middleend	0.00	
9622327	08/10/20 17:38:13	08/10/20 17:38:13	Zwijndrecht	0	33nm	opc	18.00	tiny	VK	1	middleend	0.00	
5331380	08/10/20 17:29:11	08/10/20 17:30:11	Goethe	0	33nm	opc	29.00	small	VK	2	middleend	0.00	
8977158	08/09/20 10:29:43	08/09/20 10:28:40	Eigra	0	33nm	opc	728.00	huge	VK	3	middleend	0.17	

A screenshot of the CalCM dashboard showing a filtered jobs table. The table lists several job entries. A purple arrow points from the word "Job" in the first column to the first row of the table. The first row is highlighted with a pink background. The other rows have white backgrounds. The columns are labeled: Job, Submit Time, Start Time, Device Name, Node, Tech Node, Module, DB Area, Categorized DB Size, Layer, Layer Revision, Layer Group, and Run time.

10. Similarly, examine the data clusters defining the box portion of the job statistics plot.

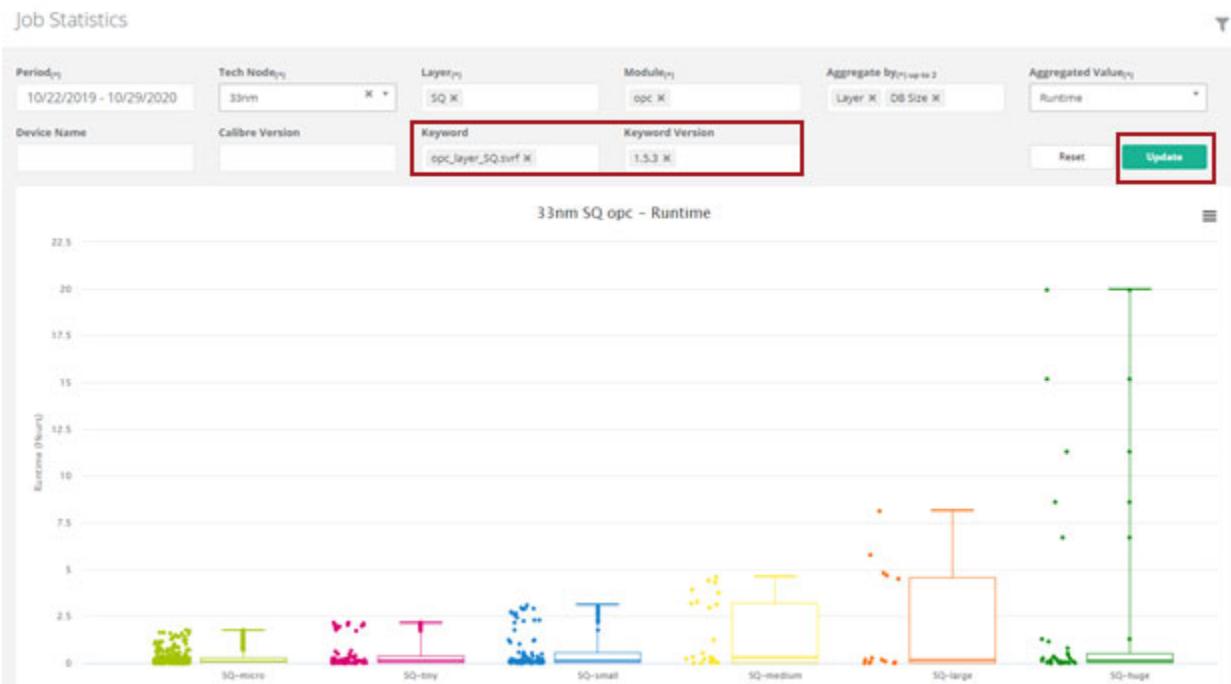
The cluster centers are represented by the box. The median of the cluster shows as the center-line between the lower (Q1) and upper (Q3) quartiles that divide the data into the 25th and 75th percentiles, respectively.

**Figure 3-50. Job Statistics Cluster Centers**

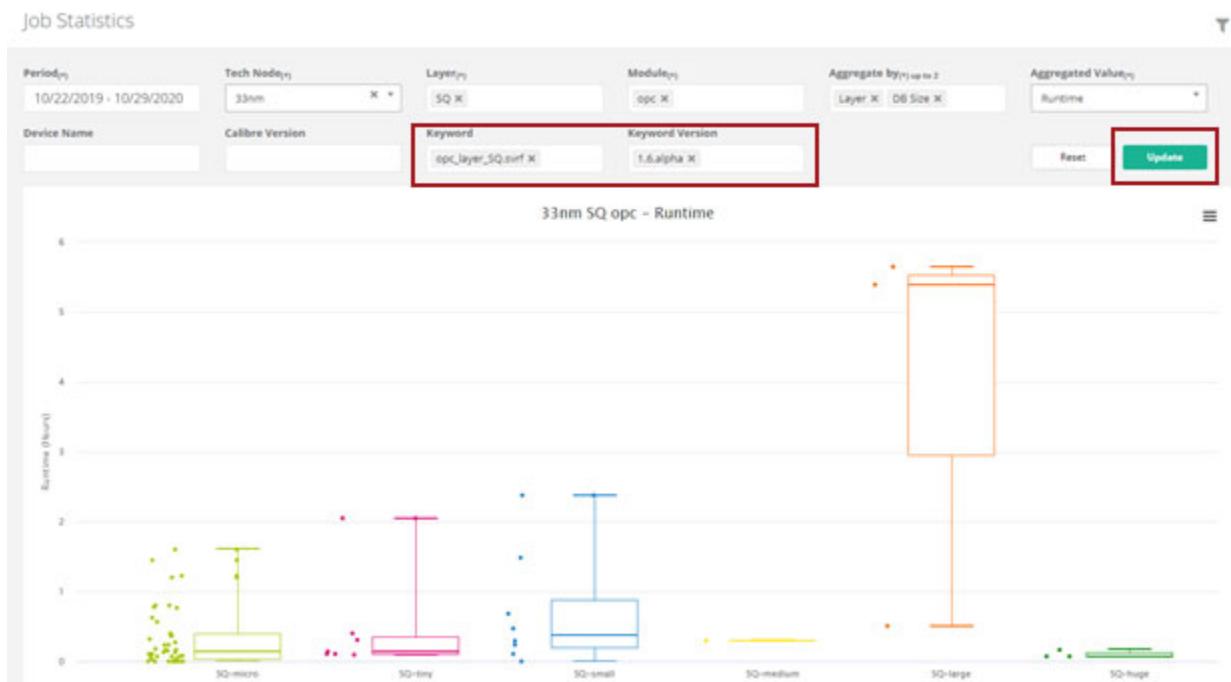


11. Use the subfilter fields to perform further analysis of your data. For example, a rule developer may filter by SVRF rule keyword version aggregated by layer and database size and then compare the runtime values for analysis.

**Figure 3-51. Job Statistics Filter by Keyword Version (1)**



**Figure 3-52. Job Statistics Filter By Keyword Version (2)**



## Results

You have performed statistical data analysis on your job by filtering and plotting aggregate values of the job data using CalCM+ advanced features in the CalCM dashboard.

## Using Cluster Utilization Information for Data Analysis

As part of the Calibre Cluster Manager Plus (CalCM+) advanced features, you can perform data analysis on your cluster utilization by displaying the statistical data in an aggregated plot and chart format for a specified period.

### Restrictions and Limitations

- CalCM+ advanced features licenses are required to enable the data analysis functionality in the CalCM dashboard. See the [Calibre Administrator's Guide](#) for complete licensing information.

### Prerequisites

- The CalCM daemon (calcmd) and CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Jobs must be running under CalCM as described in “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.

## Procedure

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (*calcmd.conf*). For example,

```
DASHBOARDPORT = 9902
```

enter in the browser,

```
http://server_name:9902
```

2. Log into the CalCM dashboard web application with your authentication information and verify that you have access to the Data Analysis pages in the Groups settings. See “[Accessing and Updating Settings in the CalCM Dashboard](#)” on page 87.
3. In the left-side menu of the CalCM dashboard, click the **Data Analysis** menu item.
4. Select Cluster Utilization to open the page in the CalCM dashboard.
5. In the Cluster Utilization page, click the Period field on the right-side of the page to set the date range for the data analysis and click **Apply**.

Choose the time period that is specific to your cluster data. A message appears warning that the data may take longer to load if the period is over one month, but you can choose a very long period of time of multiple years to plot long-term statistics. Look at the progress bar to know when the plot is ready.

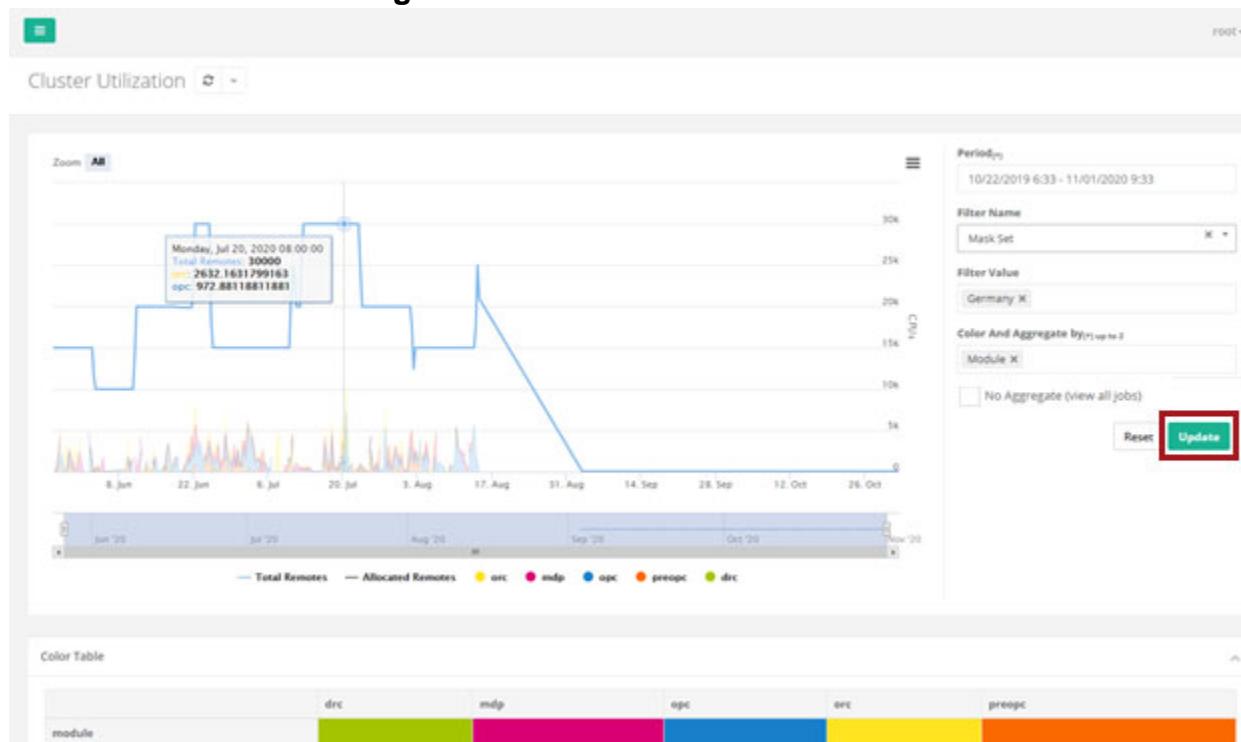
6. Click the Filter Name field and select **Mask Set** or other filter name from the dropdown menu. You can also leave this field blank, in which case all jobs will show.
7. Click the Filter Value field and select a value for the filter name.
8. Click the Color And Aggregate by field and select Module.

You can optionally select two menu items to color and aggregate.

9. Check No Aggregate (View all Jobs) to view all jobs. It is recommended to specify a relatively small period range when selecting No Aggregate to prevent large numbers of jobs to be displayed.
10. Click **Update** to create a color-coded plot of the aggregated data over the specified period.

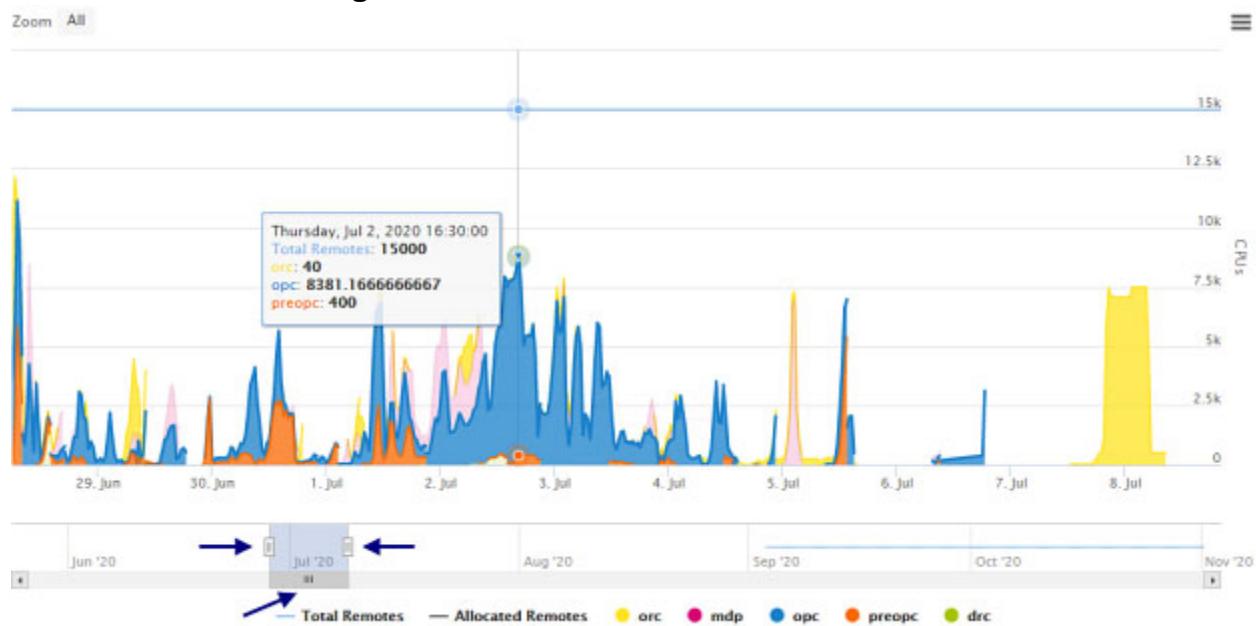
You can hover over a plot line to display information for a certain time. Plot detail includes the total number of remotes and CPUs (in this case, by module). The Color Table below the main plot window shows the colors used for the aggregated data.

**Figure 3-53. Cluster Utilization Plot**



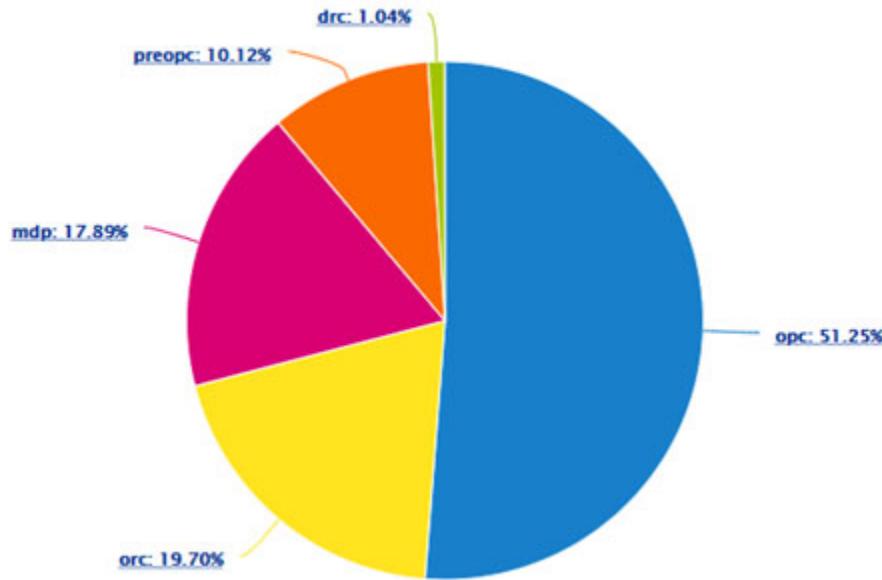
11. Use the slider at the bottom of the plot to adjust the x-axis for specific dates during the analysis period. The slider can be clicked and dragged at three points. Dragging either the left or right edge points of the slider adjusts the range displayed (zoom level). Dragging the bottom of the slider shifts the display along the time line while preserving the zoom level.

**Figure 3-54. Cluster Utilization Plot Detail**



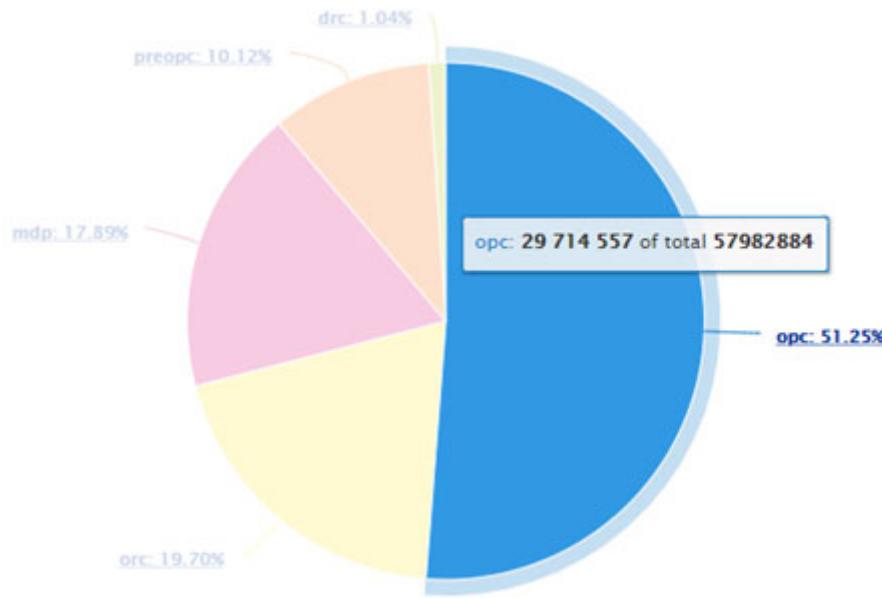
12. At the bottom of the Cluster Utilization page, the Top Ten Remote Usage information displays as a percentage value in the pie chart.

**Figure 3-55. Cluster Utilization Top Ten Remote Usage**



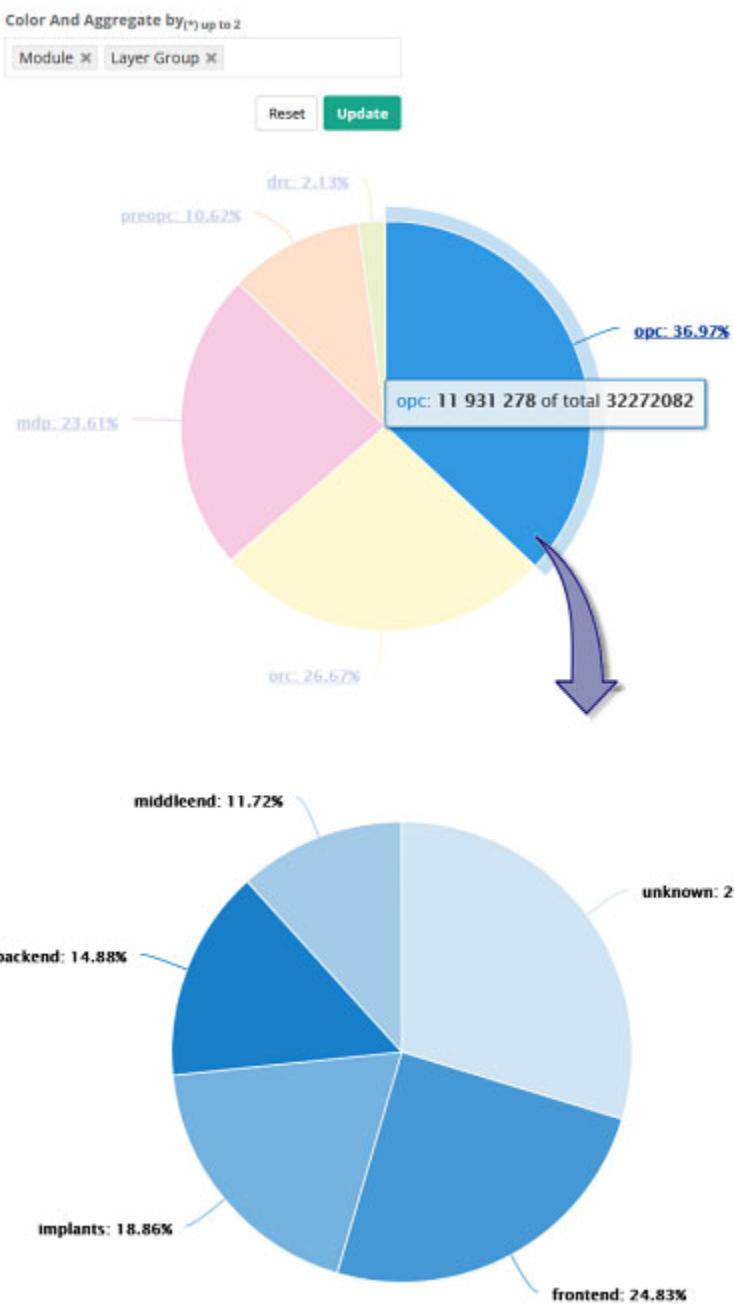
You can hover over a color sector of the pie chart to display the total remote usage for the aggregated data.

**Figure 3-56. Cluster Utilization Top Ten Remote Usage Detail**



13. When two aggregate factors are used, you can click in the pie chart to see the distribution over the second criteria.

**Figure 3-57. Viewing Second Criteria Distribution With Two Aggregates**



## Results

You have performed data analysis on the cluster utilization for your job by filtering and plotting aggregate values using CalCM+ advanced features in the CalCM dashboard.

# CalCM+ Advanced Features for Accessing CalScope in the CalCM Dashboard

As part of the Calibre Cluster Manager Plus (CalCM+) advanced features, you can access the Calibre® System Monitoring Solution (CalScope) software tool to view job and hardware monitoring data for your Calibre run.

---

## Note

 For complete details on CalScope, refer to “[CalScope Usage and Reference](#)” on page 391.

---

## Restrictions and Limitations

- CalCM+ advanced features licenses are required to access the CalScope software tool in the CalCM dashboard. See the [Calibre Administrator’s Guide](#) for complete licensing information.

## Prerequisites

- The CalCM daemon (calcld) and CalCM dashboard web application must be running as described in “[Starting the CalCM Daemon](#)” on page 49.
- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#)” on page 51.
- Jobs must be running under CalCM as described in “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.
- The `calcm_hmonitor_app.tcl` application must be configured to enable the CalScope tool for hardware monitoring, synchronizing of host status, and the command-line interface (CLI) under CalCM. For example:

```
APPLICATION TCL calcld_hmonitor_app.tcl FILE [
    ENABLECALSCOPE = 1
    CALSCOPECLI = "$MGC_HOME/bin/calscope"
    CALSCOPEADD = 3
    CALSCOPEREMOVE = 1
]
```

- The `calcm_http_server_app.tcl` application must be configured to use CalScope menus in the dashboard. For example:

```
APPLICATION TCL calcld_http_server_app.tcl FILE [
    ...
    ENABLECALSCOPE = 1
    LOGVIEWDB = postgresql+psycopg2://cscope:\n        cscope@<server_name>:<port>/cscope_cluster
    CALCMONDB = postgresql+psycopg2://cscope:\n        cscope@<server_name>:<port>/cscope_cluster
    CALAMSDB = postgresql+psycopg2://cscope:\n        cscope@<server_name>:<port>/cscope_cluster
    ...
]
```

## Procedure

1. Open a browser and enter the server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM configuration file (*calcmd.conf*). For example:

```
DASHBOARDPORT = 9902
```

enter in the browser,

```
http://server_name:9902
```

2. Log into the CalCM dashboard web application with your authentication information and verify that you have access to the CalScope Jobs page in the Groups settings. See “[Accessing and Updating Settings in the CalCM Dashboard](#)” on page 87.
3. In the left-side menu of the CalCM dashboard, click the **CalScope** menu item to open the Logview/Monitor Host Job List page.

From this page, you can view the monitoring recipe, Calibre version, command-line arguments, and operating mode parsed from your Calibre transcript.

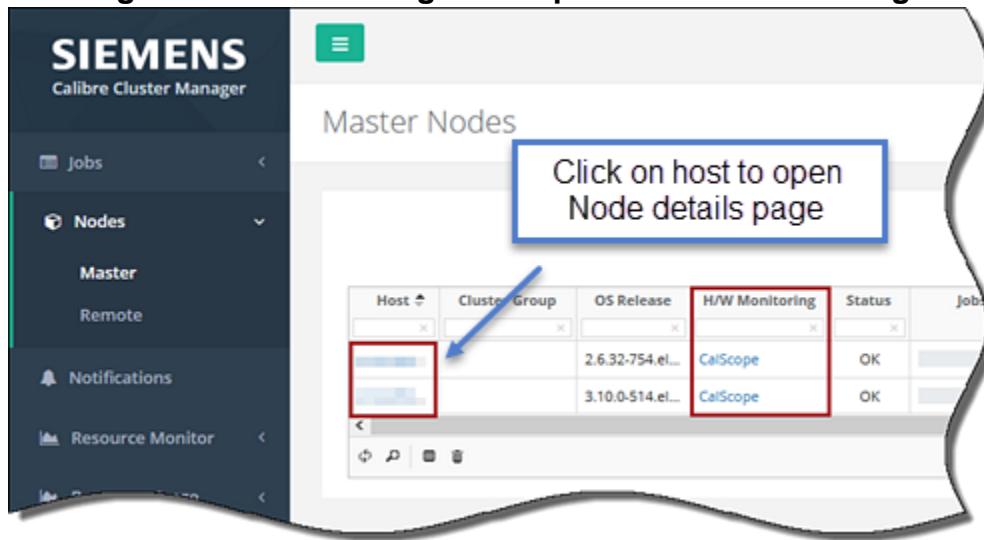
**Figure 3-58. Logview/Monitor Host Job List**

Job	Monitoring Recipe	Log File Name	Log File Path	Version	Command Line
1	UNKNOWN	opcv.log.1	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
10	UNKNOWN	opcv.log.10	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
100	UNKNOWN	opcv.log.100	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
1000	UNKNOWN	opcv.log.1000	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
1001	UNKNOWN	opcv.log.1001	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
1002	UNKNOWN	opcv.log.1002	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
1003	UNKNOWN	opcv.log.1003	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
1004	UNKNOWN	opcv.log.1004	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
1005	UNKNOWN	opcv.log.1005	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
1006	UNKNOWN	opcv.log.1006	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
1007	UNKNOWN	opcv.log.1007	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
1008	UNKNOWN	opcv.log.1008	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/
1009	UNKNOWN	opcv.log.1009	v2021.3_0...	v2021.3_0...	// Running Mgc_home/pkgs/

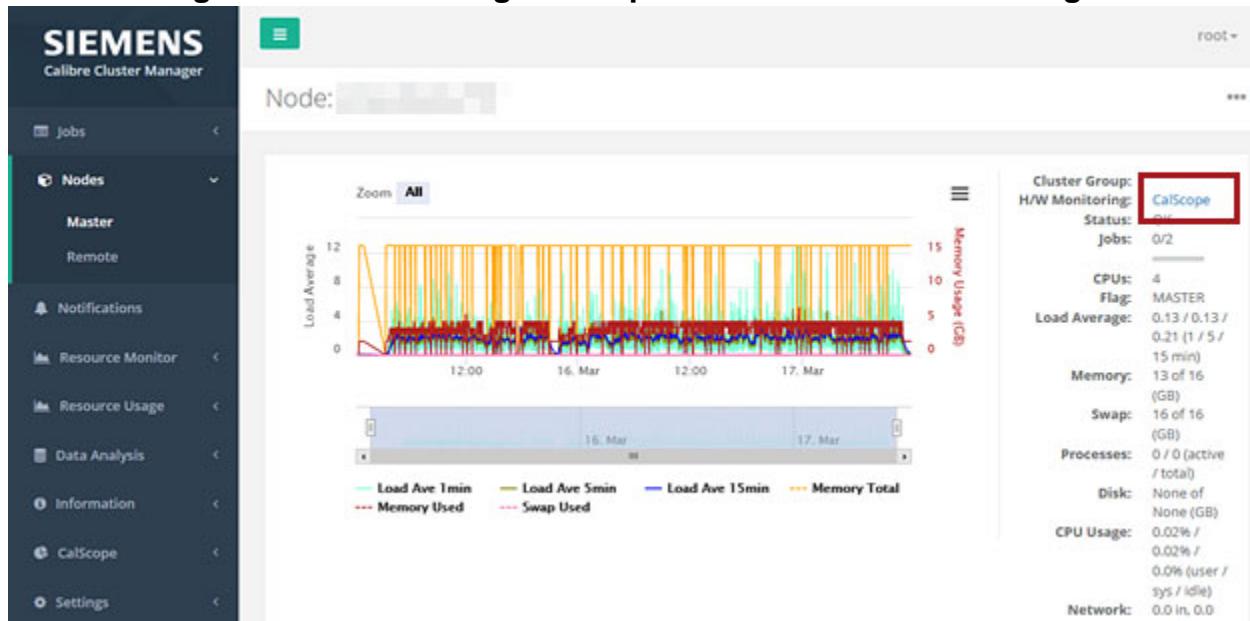
4. Click a job ID in the table to open the job detail page and display plots of hardware and system monitoring metrics.

5. You can directly access CalScope hardware and system monitoring metrics from the Nodes tables and the Node detail page (click on the host link in the Nodes table). Clicking these CalScope links from these pages takes you to the Cluster page with the time span for that host.

**Figure 3-59. Accessing CalScope From the Nodes Page**



**Figure 3-60. Accessing CalScope From the Node Details Page**



6. Similarly, you can access the Cluster page with the time span of a job for a particular host by clicking the CalScope link in the Notifications table of the Job details page.

**Figure 3-61. Accessing CalScope From the Job Details Notifications Table**

Date	Node	H/W Monitoring	Message
03/17/21 10:43:...	-	CalScope	job(1593): tag(low_demand): The demand was 0 or less for 5 update cycles. (0 0 0 0 0)
03/17/21 10:43:...	-	CalScope	job(1593): tag(long_ops): Long Operation (327 seconds)
03/17/21 10:42:...	-	CalScope	job(1593): tag(long_ops): Long Operation (281 seconds)
03/17/21 10:42:...	-	CalScope	job(1593): tag(low_demand): The demand was 0 or less for 5 update cycles. (0 0 0 0 0)
03/17/21 10:41:...	-	CalScope	job(1593): tag(long_ops): Long Operation (235 seconds)
		CalScope	job(1593): tag(long_ops): Long Operation (235 seconds)
		CalScope	job(1593): tag(long_ops): Long Operation (235 seconds)
		CalScope	job(1593): tag(long_ops): Long Operation (235 seconds)

7. Use the command-line interface to add or remove hosts and query hardware monitoring data from the CalScope monitoring session. The default path is `$MGC_HOME/bin/calscope`. For example:

```
calscope hw_monitor add -hostnames host01 host02  
calscope hw_monitor remove -hostnames host03 host04
```

## Results

You have enabled the CalScope software tool to view hardware monitoring and job information for your Calibre run using CalCM+ advanced features in the CalCM dashboard.

Refer to “[CalScope Usage and Reference](#)” on page 391 for complete details on CalScope.

# Running CalCM with a Backup Daemon

Running CalCM with a backup daemon is necessary to ensure the continuous operation of CalCM in the event the primary daemon fails or is terminated. After launching the primary daemon, you can launch the backup daemon using the same CalCM configuration. The backup daemon then waits until the primary daemon fails or is terminated.

## Prerequisites

- Primary and backup hosts for running multiple CalCM daemons.

## Procedure

1. Open the CalCM configuration file (`calcmd.conf`) in an ASCII editor and replace the LICENSE PORT statement with the **LICENSE SERVER** statement, substituting the strings shown in italics as described below.

```
LICENSE SERVER primary:port;backup:port
```

- *primary* — Replace this string with the hostname of the license server running the primary CalCM daemon.

- *backup* — Replace this string with the hostname of the license server running the backup CalCM daemon.
  - *port* — Replace this string with the port numbers on the primary and backup servers to be used for the License Broker Daemon (LDB).
2. Run the following command on the primary and backup hosts to start the CalCM daemon (calcmd):  

```
$CALIBRE_HOME/bin/calcmd -config calcmd.conf
```
  3. Verify that the backup daemon is waiting for the primary daemon.  

```
Locking "calcm/calcmd.conf" on backup with the process, pid.  
WARNING: already locked on primary by other process, pid.  
WARNING: waiting for the lock...
```

4. Use the Linux kill command to kill the primary daemon:

```
kill -HUP {primary pid}
```

5. Use the Linux ps command to verify that the backup daemon is launched.

## Results

The backup daemon is now responsible for handling the dispatching and monitoring of Calibre jobs launched using CalCM.

When killing CalCM with a backup daemon already waiting, the backup should be killed first, followed by killing the primary daemon.



# Chapter 4

## CalCM Command Line and Tcl Application Reference Dictionary

---

Reference information for the CalCM environment variables, command line, messaging commands, and Tcl applications is provided.

Refer to “[Syntax Conventions](#)” on page 22 for the command syntax conventions.

<b>CalCM Environment Variables .....</b>	<b>110</b>
<b>CalCM Command Line Reference.....</b>	<b>122</b>
<b>Messaging Commands.....</b>	<b>130</b>
<b>CalCM Tcl Application Reference.....</b>	<b>161</b>

## CalCM Environment Variables

---

This section provides reference information on the CalCM environment variables.

<b>CALCMD_CONF_PATH</b> .....	<b>111</b>
<b>CALCMD_LOGLEVEL</b> .....	<b>112</b>
<b>CALCMD_LOG_TIMESTAMPS</b> .....	<b>113</b>
<b>CALCMD_REMOTE_COMMAND_DYNAMIC_LOG_LEVEL</b> .....	<b>114</b>
<b>CALIBRE_ENABLE_SETLAYER_ECP</b> .....	<b>119</b>
<b>CALIBRE_ENABLE_SETLAYER_ERT</b> .....	<b>120</b>
<b>CALIBRE_LBD_LOG_TIMESTAMPS</b> .....	<b>121</b>

## **CALCMD\_CONF\_PATH**

Environment variable specifying the path to the CalCM configuration file.

### **Usage**

`CALCMD_CONF_PATH path`

### **Arguments**

- *path*

Specifies the path to the CalCM configuration file.

### **Description**

This environment variable specifies the path to the CalCM configuration file. When you invoke CalCM, the daemon searches for a CalCM configuration file in the following locations and order listed below:

1. A file defined by the `-config` command line option.
2. A file defined by the `CALCMD_CONF_PATH` environment variable.
3. The `calcmd.conf` file in the current working directory.
4. The `.calcmd.conf` file (note the period) in your home directory.
5. The `calcmd.conf` file in the system `/etc` directory.

### **Examples**

```
setenv CALCMD_CONF_PATH $CALCM_HOME/config/calcmd.conf
```

## **CALCMD\_LOGLEVEL**

Environment variable specifying the level of information that is output to the transcript.

### **Usage**

`CALCMD_LOGLEVEL [LOG | ERROR | WARNING | MESSAGE | NOTE]`

### **Arguments**

- **LOG**  
Outputs basic level of information.
- **ERROR**  
Outputs error and log information.
- **WARNING**  
Outputs warning, error, and log information.
- **MESSAGE**  
Outputs message, error, warning, and log information (default).
- **NOTE**  
Outputs all levels of information.

### **Description**

This environment variable controls the level of information that is output to the transcript by the CalCM daemon.

### **Examples**

```
setenv CALCMD_LOGLEVEL ERROR
```

## **CALCMD\_LOG\_TIMESTAMPS**

Environment variable that enables printing of timestamps to the transcript.

### **Usage**

`CALCMD_LOG_TIMESTAMPS value`

### **Arguments**

- *value*

Set this variable to “1” to enable the timestamps printing feature.

### **Description**

This environment variable enables the printing of timestamps to the transcript for the CalCM daemon and the LBD. If not set, the timestamps printing feature is not activated (default). Banner and configuration lines in the transcript are exempt from timestamps printing. The timestamps format is [year-month-day hour:minute:second].

### **Examples**

```
setenv CALCMD_LOG_TIMESTAMPS 1
```

## **CALCMD\_REMOTE\_COMMAND\_DYNAMIC\_LOG\_LEVEL**

Environment variable specifying the amount of information that is output to the transcript when REMOTE COMMAND DYNAMIC is enabled.

### **Usage**

```
CALCMD_REMOTE_COMMAND_DYNAMIC_LOG_LEVEL [ 0 | 1 ]
```

### **Arguments**

- 0  
Disables verbose mode.
- 1  
Enables verbose mode (default).

### **Description**

This environment variable controls the amount of information that is output to the transcript when the REMOTE COMMAND DYNAMIC statement is used. Note that REMOTE COMMAND DYNAMIC is automatically set when launching a job to be governed by CalCM. Following is the syntax and a description of the information that is transcribed when verbose mode is enabled:

```
<DCA <timestamp> 0 <remote_cpu_timestamp> N
  {<hostname>|<ip_addr>} R <np> <nt> <nc> <ns> <nd> T <ntc> <ntl>
  <nar> A <nat> <nmt> <pid1> [.. <pidN>] >
```

In hyperthread mode, the format of the process ID (PID) syntax is specified as follows:

```
<pid>(V) [<sibling_pid>] <pid>(P)
```

**Table 4-1. Description of Verbose Output from CALCMD\_REMOTE\_COMMAND\_DYNAMIC\_LOG\_LEVEL**

<b>Field</b>	<b>Value Returned</b>	<b>Description</b>
timestamp	int >0	Wall clock seconds elapsed since the start of the Calibre job.
<i>unused</i>	0	This field is not used.
remote_cpu_timestamp	int >0	CPU processing seconds elapsed on all of the remote CPU cores since the start of the Calibre job.
N	string	Literal character indicating the hostname or IP address.

**Table 4-1. Description of Verbose Output from  
CALCMD\_REMOTE\_COMMAND\_DYNAMIC\_LOG\_LEVEL (cont.)**

<b>Field</b>	<b>Value Returned</b>	<b>Description</b>
hostname	string	Remote host system name.
ip_addr	string	Remote host IP address will be displayed if the remote host's name is unknown. <sup>1</sup>
R	string	Literal character indicating the start of the remote cpu core and remote compute server (RCS) information.
np	int >0	The number of “physical” CPU cores available on the remote host.
nt	int >=0	The number of “virtual” CPU cores available on the remote host.
nc	int >0	The number of “chip socket packages” on the remote host. <sup>2</sup>
ns	int >0	The number of RCS running on the remote host. <sup>3</sup>
nd	[ - ]int >0	Change, or delta, in CPU cores connected or disconnected for the job. A positive number indicates the number of CPU cores that were connected, while a negative number indicates the number of CPU cores that were disconnected.
T	string	Literal character indicating the start of the total remote CPU cores information.
ntc	int >0	Total number of connected cores on all remote hosts.
ntl	int >0	Total number of licensed cores on all remote hosts.
nar	int >0	Total number of active remotes for the job. This value is equivalent to the MON:L number5 element as stated in the <i>Calibre Administrator's Guide</i> .
A	string	Literal character indicating the start of the information pertaining to the number and type of threads on the primary.
nat	int >=nar	Total number of awake threads for the job.

**Table 4-1. Description of Verbose Output from  
CALCMD\_REMOTE\_COMMAND\_DYNAMIC\_LOG\_LEVEL (cont.)**

Field	Value Returned	Description
nmt	int >0	Total number of maximum (max) threads for the job.
<pid1> [... <pidN>]	int>0	Process ID of the remote rcalibre process. PID1 is required to be output. If more than one rcalibre process is connected or disconnected, then a PID space delimited list should be appended to the output. The number of PIDs will equal the absolute value of nd.
<pid>(V) [sibling_pid] <pid>(P)	int >0	Process ID of the remote rcalibre process in hyperthreading mode. In this mode, <pid>(V) is the virtual core's PID, where [sibling_pid] is its matching physical core's PID, and <pid>(P) is a physical core's PID.

1. Calibre MTflex remote configuration files support both mechanisms of identifying remote hosts. When using LAUNCH AUTOMATIC, both “REMOTE HOST node3001” and “REMOTE HOST 147.34.120.105” are equally valid. DNS is not required.
2. Large Calibre primaries may have 4 or 8 chip packages, but remote hosts with more than 2 packages are not recommended for OPC due to poor performance. The number of packages may be unknown on older systems, in which case this value defaults to 1.
3. ns can be larger than np when hyperscaling is enabled.

## Examples

### Example 1

1. In this example, the CALCMD\_REMOTE\_COMMAND\_DYNAMIC\_LOG\_LEVEL is defined as follows:

```
setenv CALCMD_REMOTE_COMMAND_DYNAMIC_LOG_LEVEL 1
```

2. At one hour into the job, CalCM adds four remote cores to the job with 10 active remotes and writes the following text to the transcript:

```
<DCA 3600 0 845612 N node10602 R 4 0 2 4 4 T 35 35 10 A 15 35
16732 16730 16728 16726>
```

At t=3600 seconds, the remote host “node10602” is shown to have:

- o 4 physical cores and 0 virtual cores = 4 total cores
- o 2 physical chips
- o 4 remote compute servers (RCS) connected and running
- o 4 CPU additional cores were connected to the job

The job has a total of:

- 35 connected cores
- 35 licensed cores
- 10 active remotes
- 15 awake threads
- 35 max threads

Remote process connected has:

- Remote process ids 16732 16730 16728 16726

#### Example 2

1. In this example, the CALCMD\_REMOTE\_COMMAND\_DYNAMIC\_LOG\_LEVEL is defined as follows:

```
setenv CALCMD_REMOTE_COMMAND_DYNAMIC_LOG_LEVEL 1
```

2. At 2 hours into the job, CalCM removes two remote cores from the job and writes the following text to the transcript:

```
<DCA 7200 0 548566 N node10735 R 4 0 2 2 -2 T 15 15 3 A 3 15  
7859 7860>
```

At t=7200 seconds, the remote host “node10735” is shown to have:

- 4 physical cores and 0 virtual cores = 4 total cores
- 2 physical chips
- 2 remote compute servers (RCS) connected and running
- 2 CPU cores were disconnected from the job

The job has a total of:

- 15 connected cores
- 15 licensed cores
- 3 active remotes
- 3 awake threads
- 15 max threads

Remote processes disconnected has:

- Remote process ids 7859 7860

### Example 3

1. In this example, the CALCMD\_REMOTE\_COMMAND\_DYNAMIC\_LOG\_LEVEL is defined as follows:

```
setenv CALCMD_REMOTE_COMMAND_DYNAMIC_LOG_LEVEL 1
```

2. At 1 hour into the job, there are four physical and four virtual cores connected. CalCM adds four remote cores to the job and writes the following text to the transcript:

```
<DCA 3600 0 854216 N node7083 R 4 4 1 4 4 T 20 10 12 A 13 20  
51689(V) [51527] 51527(P) 51531(P) 51686(V) [51531]>
```

At t=3600 seconds, the remote host “node7083” is shown to have:

- o 4 physical cores and 4 virtual cores = 8 total cores
- o 1 physical chip
- o 4 remote compute servers (RCS) connected and running
- o 4 additional CPU cores were connected to the job

The job has a total of:

- o 20 connected cores
- o 10 licensed cores
- o 12 active remotes
- o 13 awake threads
- o 20 max threads

Remote process has two pairs of remote compute servers (RCS) connected:

- o 51527 (physical) and 51689 (virtual)
- o 51531 (physical) and 51686 (virtual)

# **CALIBRE\_ENABLE\_SETLAYER\_ECP**

Environment variable that enables the setlayer progress meter.

## **Usage**

`CALIBRE_ENABLE_SETLAYER_ECP value`

## **Arguments**

- *value*

Enables the setlayer progress meter. Set this variable to “1” to enable this feature.

## **Description**

This environment variable is used to enable the setlayer progress meter for tracking the estimated completion percentage of a Calibre job. This information is then used by the CalCM TAT application to determine whether the priority should be adjusted for the job.

This variable is used to enable the ECP without modifying the rule file. This variable does not enable the Setlayer ERT feature.

## **Examples**

```
setenv CALIBRE_ENABLE_SETLAYER_ECP 1
```

## **Related Topics**

[CALIBRE\\_ENABLE\\_SETLAYER\\_ERT](#)

## **CALIBRE\_ENABLE\_SETLAYER\_ERT**

Environment variable that enables setlayer ERT features.

### **Usage**

`CALIBRE_ENABLE_SETLAYER_ERT [1|2]`

### **Arguments**

- `[1|2]`

Enables the Setlayer ERT features.

1 — Enables the Setlayer ERT features only for operations that turn on the progress meter.

2 — Enables the Setlayer ERT features for all setlayer operations.

### **Description**

This environment variable enables the setlayer ERT features for calculating the estimated remaining time for a Calibre job. This information is then used by the CalCM TAT application to make resource allocation decisions for the job. If not set, the setlayer ERT features are not activated.

### **Examples**

```
setenv CALIBRE_ENABLE_SETLAYER_ERT 2
```

# **CALIBRE\_LBD\_LOG\_TIMESTAMPS**

Environment variable that controls the printing of timestamps to the transcript for the LBD.

## **Usage**

`CALIBRE_LBD_LOG_TIMESTAMPS [0 | 1]`

## **Arguments**

- `[0 | 1]`

Controls the printing of timestamps to the transcript for the LBD.

`0` — Disables the printing of timestamps for the LBD.

`1` — Enables the printing of timestamps for the LBD.

## **Description**

This environment variable is used to control the timestamps printing feature for the LBD only. Set this variable to “0” to override the printing of timestamps for the LBD to the transcript when `CALCMD_LOG_TIMESTAMPS` is set. By default, the timestamps for the LBD are not printed to the transcript unless `CALCMD_LOG_TIMESTAMPS` is set. The timestamps format is [year-month-day hour:minute:second].

## **Examples**

```
setenv CALIBRE_LBD_LOG_TIMESTAMPS 1
```

## CalCM Command Line Reference

---

This section provides information on the command line syntax for the available CalCM commands and related scripts.

<b>calcmd</b> .....	<b>123</b>
<b>calcめ_kill_job</b> .....	<b>126</b>
<b>calcめ_send_message</b> .....	<b>127</b>
<b>calcめ_submit_job</b> .....	<b>129</b>

# calcmd

Invokes the CalCM daemon.

## Usage

**calcmd** {-config *path*} [-log *path* [-logsize *size[unit]*]] [-simulation]

## Arguments

- **-config *path***

Required argument that specifies the path to the configuration file to be used by the CalCM daemon.

- **-log *path***

By default, calcmd writes all messages to the standard output device (usually console). However, if -log is specified, the output is redirected to the specified path. The special character '%' can be used to convert and format the current time using a Linux strftime function. For example, you can use '%s' to specify time in the number of seconds since the epoch time:

```
$MGC_HOME/bin/calcmd -config mycalcmd.conf -log mycalcmd.log.%s
```

Detailed information for the strftime function parameters can be found in the Linux manual pages.

- **-logsize *size[unit]***

Specifies the maximum file size for the calcmd log file when -log is used. Once the file size limit is reached, CalCM archives the log file and starts a new one. The new log file continues the transcript information from the point where the archived log file ends. File name collisions are avoided by the automatic appending of the '.' and epoch time at the end of the log file name. The new log file name is appended with the epoch time from the archived log file and the epoch time when the new log file starts. The file size unit specifications are 'k', 'm', 'g', or 't'. If *unit* is not specified, the default unit is byte. By default, CalCM does not archive and start a new calcmd log file unless -logsize is specified.

- **-simulation**

Provides a limited debugging capability to test that CalCM is running and the applications are compiling. This does not provide a full simulation mode.

---

### Caution



Running -simulation may cause the database to become corrupted. If this occurs, you should delete the database or revert back to a previous version before running CalCM.

---

## Description

The **calcmd** command invokes the CalCM daemon which runs continuously in the background on a machine that has network connection to all Calibre primary and remote hosts.

The current directory of the CalCM daemon is “/” and all program output is completely suppressed unless the path to a log file is provided explicitly using the `-log path` command line option.

The SIGINT (interrupt process) or SIGTERM (terminate process) signal should be used to stop the daemon.

## Examples

### Example 1

In this example, you invoke calcmd with no options. The CalCM daemon searches for a configuration file in the standard locations. The log file is output to standard out and the application runs as a regular console application.

```
$MGC_HOME/bin/calcmd
```

### Example 2

In this example, you invoke calcmd and specify the path to the configuration file. Log information is output to standard out and the application runs as a regular console application.

```
$MGC_HOME/bin/calcmd -config calcmd.conf
```

### Example 3

In this example, you invoke calcmd and specify the path to the configuration file. Log information is output to a log file named “mylogfile.log”.

```
$MGC_HOME/bin/calcmd -config mycalcmd.conf -log mylogfile.log
```

### Example 4

In this example, you invoke calcmd and specify the path to the configuration file. You also specify the `-log` option and a log file size limit. Log information is output to a log file named “mylogfile.log”. When the log file size limit is met, the log file is automatically archived and a new log file is started.

```
$MGC_HOME/bin/calcmd -config mycalcmd.conf -log mylogfile.log -logsize 2g
```

### Example 5

In this example, you invoke calcmd and specify the path to the configuration file. You also specify the `-log` option with the special characters ‘.%s’ at the end of the output log file name. Log information is output to a log file named “mylogfile.log”. The time in the number of seconds since the epoch time is appended to the end of the log file name.

```
$MGC_HOME/bin/calcmd -config mycalcmd.conf -log mylogfile.log.%s
```

## Related Topics

- [Starting the CalCM Daemon](#)
- [Product Requirements](#)
- [CalCM Configuration Files](#)

## calcm\_kill\_job

Kills one or more jobs running under CalCM.

---

### Note

 You can also access this command from the CalCM dashboard web interface with the required authentication.

---

### Usage

**calcm\_kill\_job *jobid* [*jobid*...]**

### Arguments

- *jobid*

Specifies the job ID of the job to kill. You can specify more than one job ID. A value of 0 can be used by a superuser to kill all jobs.

### Description

The **calcm\_kill\_job** script is used to kill an active job or remove a pending job from a job queue in CalCM. The user executing the script must be defined using the JOB USER statement in the *job.conf* file. The user that started the CalCM daemon (calcmd) is considered a superuser and can kill all jobs using a job ID of 0.

The status of a job at the time this command is issued determines the information that appears in the transcript as shown in [Table 4-2](#).

**Table 4-2. Transcript Messages for calcm\_kill\_job**

When the job...	Issuing the command ...	Generates the following message in the transcript...
Is in the queue	calcm_kill_job 6	Removing job “6”.
Is running	calcm_kill_job 1	Killing job “1”.
Does not exist	calcm_kill_job 8	Returns with no output message.

### Examples

/calcm/jobqueue/calcm\_kill\_job 100000 100001

### Related Topics

[JOB USER](#)

## [calcm\\_send\\_message](#)

Sends a message to CalCM.

### Usage

**calcm\_send\_message *message***

### Arguments

- ***message***

Specifies a messaging command to be processed by CalCM. Refer to “[Messaging Commands](#)” for information on the valid commands.

### Description

The **calcm\_send\_message** script is used to send a message to CalCM. The script is generated in the directory specified by the QUEUEDIR parameter in the [calcm\\_jobqueue\\_app.tcl](#) application.

The message is processed by the CalCM applications specified in the CalCM configuration file. Only the application that solicits the message will accept and process the message.

### Examples

```
/calcm/jobqueue/calcm_send_message prioritize_job 1 254
```

### Related Topics

[adjust\\_cluster](#)  
[adjust\\_maxchange](#)  
[adjust\\_minmax](#)  
[adjust\\_quota](#)  
[build\\_jobanalysis](#)  
[check\\_cluster\\_limit](#)  
[adjust\\_rampupdown](#)  
[close\\_node](#)  
[log\\_level](#)  
[log\\_timestamp](#)  
[open\\_node](#)  
[prioritize\\_job](#)  
[queue\\_status](#)  
[read\\_conf](#)

[register\\_node](#)  
[renew\\_license](#)  
[reset\\_license](#)  
[reset\\_jobanalysis](#)  
[reset\\_notification](#)  
[reset\\_quota](#)  
[resume\\_job](#)  
[set\\_budget](#)  
[set\\_priority\\_bump](#)  
[span\\_log](#)  
[suspend\\_job](#)  
[unregister\\_node](#)

## **calcm\_submit\_job**

Submits a job to CalCM.

### Usage

**calcm\_submit\_job** [-force\_launch] [*conf\_path*] [*rule\_index*]

### Arguments

- **-force\_launch**  
Specifies the job will be launched in the next update cycle.
- ***conf\_path***  
Specifies the absolute path to the job configuration file that is used to start a job.
- ***rule\_index***  
Specifies the index to the rule when several rules are specified in the job configuration file.  
The default value is 0.

### Description

The **calcm\_submit\_job** script is used to submit a job to CalCM. The script is generated in the directory specified by the QUEUEDIR parameter in the [calcm\\_jobqueue\\_app.tcl](#) application.

When the **calcm\_submit\_job** script is invoked with a path to the job configuration files, the job related files (*job.conf* and *job.env*) are copied to CONFDIRNAME in [calcm\\_jobqueue\\_app.tcl](#). By default, CONFDIRNAME is “.\$jobID” and is created under the job directory. This behavior is controlled by the COPYCONFFILES parameter in [calcm\\_jobqueue\\_app.tcl](#). The COPYCONFFILES parameter is enabled (“true”) by default, and if disabled (“false”), then the job configuration files remain in their original specified path.

Upon submitting a job, a job ID is automatically assigned to the job and is displayed in the transcript. The job ID is used by CalCM when making changes to a specific job. For example, some messaging commands (such as **prioritize\_job**) require you to specify the job ID when executing the command.

### Examples

```
/calcm/jobqueue/calcm_submit_job /calcm/jobqueue/job1/job.conf
```

Upon submitting the job, the transcript echoes the following information:

```
# .../.../jobqueue/calcm_submit_job 'pwd' /job.conf
Job "1" is submitted.
```

### Related Topics

[calcm\\_jobqueue\\_app.tcl](#)

# Messaging Commands

---

This section provides reference information on the messaging commands you can use with the calcm\_send\_message script. These messaging commands enable you to change aspects, such as the budget or priority, of a job that is currently running. Certain commands (as noted) are also accessible from the CalCM dashboard web interface if you supply the required authentication.

<b>adjust_cluster</b> . . . . .	<b>131</b>
<b>adjust_maxchange</b> . . . . .	<b>133</b>
<b>adjust_minmax</b> . . . . .	<b>134</b>
<b>adjust_quota</b> . . . . .	<b>135</b>
<b>adjust_rampupdown</b> . . . . .	<b>136</b>
<b>build_jobanalysis</b> . . . . .	<b>137</b>
<b>check_cluster_limit</b> . . . . .	<b>138</b>
<b>close_node</b> . . . . .	<b>139</b>
<b>log_level</b> . . . . .	<b>140</b>
<b>log_timestamp</b> . . . . .	<b>141</b>
<b>open_node</b> . . . . .	<b>142</b>
<b>prioritize_job</b> . . . . .	<b>143</b>
<b>queue_status</b> . . . . .	<b>144</b>
<b>read_conf</b> . . . . .	<b>146</b>
<b>register_node</b> . . . . .	<b>147</b>
<b>renew_license</b> . . . . .	<b>148</b>
<b>reset_jobanalysis</b> . . . . .	<b>150</b>
<b>reset_license</b> . . . . .	<b>151</b>
<b>reset_notification</b> . . . . .	<b>153</b>
<b>reset_quota</b> . . . . .	<b>154</b>
<b>resume_job</b> . . . . .	<b>155</b>
<b>set_budget</b> . . . . .	<b>156</b>
<b>set_priority_bump</b> . . . . .	<b>157</b>
<b>span_log</b> . . . . .	<b>158</b>
<b>suspend_job</b> . . . . .	<b>159</b>
<b>unregister_node</b> . . . . .	<b>160</b>

## adjust\_cluster

Related application: [calcm\\_rmanager\\_app.tcl](#)

Adjusts the cluster string for the specified job.

---

### Note

 You can also access this command from the CalCM dashboard web interface with the required authentication.

---

### Usage

**adjust\_cluster *job\_id* *cluster\_string***

### Arguments

- ***job\_id***

Specifies the id of the job whose budget you want to change.

- ***cluster\_string***

A string specifying the adjustment in resources for the specified ***job\_id***. The format of the ***cluster\_string*** is:

***count:min:max:count\_max***

***count*** — The initial number of remotes. This value is initially defined by the **REMOTE COUNT** statement in the job configuration file.

***min*** — The minimum number of remotes. This value is initially defined by the **REMOTE MIN** statement in the job configuration file.

***max*** — The maximum number of remotes. This value is initially defined by the **REMOTE MAX** statement in the job configuration file.

***count\_max*** — The maximum number of remotes in a Calibre MTflex configuration. This value is initially defined by the **REMOTE COUNT\_MAX** statement in the job configuration file.

The wildcard character (\*) can be used in place of a value and specifies to use the existing number of remotes for that value.

### Description

Use **adjust\_cluster** with the `calcm_send_message` script to adjust the minimum and maximum number of resources for the specified job. This information is passed to the `calcm_rmanager_app.tcl` application which adjusts the job resources.

### Examples

#### Example 1

This example sends a message to adjust the resources for job ID 192 by changing the initial number of remotes to 1, the minimum number of remotes to 2, the maximum number of remotes to 100, and the maximum number of Calibre MTflex remotes to 100.

```
calcm_send_message adjust_cluster 192 1:2:100:100
```

### Example 2

This example sends a message to adjust the resources for job ID 192 by changing the initial number of remotes to 2 and using the existing values for the minimum, maximum, and maximum Calibre MTflex number of remotes.

```
calcm_send_message adjust_cluster 192 2:*:*:*
```

### Related Topics

[calcm\\_send\\_message](#)

# **adjust\_maxchange**

Related application: [calcm\\_rmanager\\_app.tcl](#)

Adjusts the maximum number of remotes to supply for a job.

---

## **Note**

 You can also access this command from the CalCM dashboard web interface with the required authentication.

---

## **Usage**

**adjust\_maxchange *job\_id value***

## **Arguments**

- ***job\_id***

Specifies the ID of the job whose maximum number of remotes you want to change.

- ***value***

The maximum number of remotes to supply for a job. Valid values include those specified for [REMOTE MAXCHANGE](#).

## **Description**

Use **adjust\_maxchange** with the `calcm_send_message` script to adjust the maximum number of remotes available in an update cycle for the specified job ID. The `adjust_maxchange` value overrides the value specified for `REMOTE MAXCHANGE` in the `job.conf` file. If the job ID is specified with a wildcard (\*) value, the global `MAXCHANGE` value is set.

## **Examples**

This example sends a message specifying that the maximum number of remotes supplied for job1 is to be automatically determined.

```
calcm_send_message adjust_maxchange job1 -1
```

## **Related Topics**

[calcm\\_send\\_message](#)

## adjust\_minmax

Related application: [calcm\\_rmanager\\_app.tcl](#)

Adjusts the minimum and maximum number of remote hosts for the specified job operation.

---

### Note

 You can also access this command from the CalCM dashboard web interface with the required authentication.

---

### Usage

**adjust\_minmax *job\_id minmax\_string***

### Arguments

- ***job\_id***

Specifies the ID of the job whose minimum and maximum number of remote hosts you want to change.

- ***minmax\_string***

A string specifying the number of minimum and maximum remote hosts for a list of job operations for the specified job ID.

### Description

Use **adjust\_minmax** with the `calcm_send_message` script to adjust the MINMAXOP string for the specified job ID. This information is passed to the `calcm_rmanager_app.tcl` application which adjusts the number of minimum and maximum remote hosts for the specified job operation list. The wildcard (\*) can be specified for the job operation type.

### Examples

```
calcm_send_message adjust_minmax job1 DRC:*=256/256,LITHO:*=128/1024
```

### Related Topics

[calcm\\_send\\_message](#)

## adjust\_quota

Related application: [calcm\\_rmanager\\_app.tcl](#)

Adjusts the quota for the specified job.

---

### Note

 You can also access this command from the CalCM dashboard web interface with the required authentication.

---

### Usage

**adjust\_quota** *job\_id* *string*

### Arguments

- *job\_id*

Specifies the id of the job whose budget you want to change.

- *string*

A string specifying the quota for the job. Use the following syntax to specify the quota string:

**GROUP[/GROUP]**

Refer to the [JOB QUOTA](#) description for information on the format for this string.

### Description

Use **adjust\_quota** with the calcm\_send\_message script to specify a new quota value for a job. This information is passed to the calcm\_rmanager\_app.tcl application which updates the quota hierarchy for the specified job.

### Examples

```
calcm_send_message adjust_quota job2 D2/P3
```

### Related Topics

[calcm\\_send\\_message](#)

## adjust\_rampupdown

Related application: [calcm\\_rmanager\\_app.tcl](#)

Adjusts the RAMPUPDOWN string of the specified job.

### Usage

**adjust\_rampupdown *job\_id updown\_string***

### Arguments

- ***job\_id***

Specifies the ID of the job whose ramp up/down ratio you want to change.

- ***updown\_string***

A string specifying the up/down ratio that is multiplied in calculating the job CPU demand for a list of job operations for the specified job ID.

### Description

Use **adjust\_rampupdown** with the calcm\_send\_message script to adjust the RAMPUPDOWN string for the specified job ID. This information is passed to the calcm\_rmanager\_app.tcl application, which adjusts the job's resource demand calculation based on the value of the up/down ratio for the specified job operation class list. The wildcard (\*) can be specified for the job operation type or to keep existing up/down ratio values. The format of the string is the same as the RAMPUPDOWN filter string.

### Examples

```
calcm_send_message adjust_rampupdown job1 DRC:*=*/0.5,LITHO:*=1.5/*
```

### Related Topics

[calcm\\_send\\_message](#)

# **build\_jobanalysis**

Related application: [calcm\\_jobanalysis\\_app.tcl](#)

Launches the categorization process immediately (instead of waiting for the next DBSTARTTIME) to rebuild the info\_analysis table.

## **Usage**

**build\_jobanalysis**

## **Arguments**

None.

## **Examples**

```
% calcm_send_message build_jobanalysis
Analyzing info_jobs table (17749)...
```

## **Related Topics**

[calcm\\_send\\_message](#)

## check\_cluster\_limit

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Previews the result of a cluster limit adjustment for the specified job.

### Usage

**check\_cluster\_limit** *count*

### Arguments

- *count*

Specifies a cluster limit number to simulate.

### Description

Use **check\_cluster\_limit** with the `calcm_send_message` script to preview the result of adjusting the CLUSTERLIMIT value. The command calculates the number of jobs that can be launched in the next cycle with the specified cluster count. The results of the calculation are based on the current resource conditions. If the quota configuration is specified, the change in the quota is also displayed. Refer to “[Quota Configuration File \(calcm\\_quota.conf\)](#)” on page 29.

### Examples

This example sends a message to simulate a cluster limit adjustment that increases the current cluster limit to 2500 remotes maximum.

```
calcm_send_message check_cluster_limit 2500
```

### Related Topics

[calcm\\_send\\_message](#)

# **close\_node**

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Closes slots in the specified nodes.

## **Usage**

**close\_node** *node\_name* [*node\_name...*]

## **Arguments**

- ***node\_name***

Specifies the name of the node with the slots that you want to close.

## **Description**

Use **close\_node** with the calcm\_send\_message script to close the slots in the specified node(s) and prevent the launching of remotes and primaries. Once the slots are closed, they must be re-opened with [open\\_node](#) in order to be used again. You can specify more than one node. The wildcard (\*) and (?) expressions are supported for type matching.

## **Examples**

```
calcm_send_message close_node node123 node45*
```

## **Related Topics**

[calcm\\_send\\_message](#)

## **log\_level**

Related application: CalCM daemon and all CalCM Tcl applications

Changes the level of information that is output to the CalCM daemon transcript.

### **Usage**

**log\_level** *level*

### **Arguments**

- **level**

Specifies the level of information that is output to the CalCM daemon transcript. Allowed levels include:

- MESSAGE** — Outputs message information. This is the initial value of the level of information.
- LOG** — Outputs all levels of information.
- ERROR** — Outputs error, warning, message, and note information.
- WARNING** — Outputs only warning, message, and note information.
- NOTE** — Outputs only note information.
- DEBUG** — Outputs only debugging information.

### **Description**

Use **log\_level** with the `calcm_send_message` script to dynamically change the level of information that is output to the transcript by the CalCM daemon. The LBD transcript is not affected by this messaging command.

### **Examples**

In this example, the `calcm_send_message` script is used to change the level of information in the CalCM daemon transcript to output error, warning, message, and note information.

```
calcm_send_message log_level ERROR
```

### **Related Topics**

[calcm\\_send\\_message](#)

# log\_timestamp

Related application: CalCM daemon and all CalCM Tcl applications

Changes whether timestamp information is output to the CalCM daemon transcript.

## Usage

`log_timestamp flag`

## Arguments

- *flag*

Allows dynamic control of the output of timestamp information to the CalCM daemon transcript.

**1** — Enables the output of timestamp information to the transcript.

## Description

Use **log\_timestamp** with the `calcm_send_message` script to dynamically change whether timestamp information is output to the transcript of the CalCM daemon. By default, timestamp information is not output. The timestamp information format is [year-month-day hour:minute:second]. The LBD transcript is not affected by this messaging command.

## Examples

In this example, the `calcm_send_message` command is used to enable the output of timestamp information to the CalCM daemon transcript.

```
calcm_send_message log_timestamp 1
```

## Related Topics

[calcm\\_send\\_message](#)

## **open\_node**

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Opens closed slots in the specified nodes.

### **Usage**

**open\_node *node\_name* [*node\_name...*]**

### **Arguments**

- ***node\_name***

Specifies the name of the node with the closed slots that you want to open.

### **Description**

Use **open\_node** with the calcm\_send\_message script to re-open closed slots from [close\\_node](#) in the specified node(s). This enables the launching of remotes and primaries. You can specify more than one node. The wildcard (\*) and (?) expressions are supported for type matching.

### **Examples**

```
calcm_send_message open_node node123 node45*
```

### **Related Topics**

[calcm\\_send\\_message](#)

# prioritize\_job

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Changes the priority for the specified job.

---

## Note

 You can also access this command from the CalCM dashboard web interface with the required authentication.

---

## Usage

**prioritize\_job** *job\_id* *priority*

## Arguments

- *job\_id*  
Specifies the id of the job whose priority you want to change.
- *priority*  
A string specifying the priority for the job.

## Description

Use **prioritize\_job** with the calcm\_send\_message script to change the priority for the specified CalCM job. This information is passed to the calcm\_jobqueue\_app.tcl application to assign the new priority to the job.

## Examples

```
calcm_send_message prioritize_job job1 3
```

## Related Topics

[calcm\\_send\\_message](#)

[JOB PRIORITY](#)

## queue\_status

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Changes or reports the job queue status for the specified quota.

### Usage

**queue\_status** [*quota*] [*status*] [*recursive*]

### Arguments

- *quota*

The specified quota group for a job. The following syntax is used to specify the quota group:  
**GROUP[/GROUP]**

Quota group specifications are hierarchical with the first group considered as the parent of the next group. A quota group automatically inherits the queue status from the parent (previous) quota group. For example, the quota group *D1/P1* inherits the parent *D1* queue status along with its specified status.

The wildcard character (\*) escaped by the backslash (\) can replace the quota group and be used to specify the whole queue (\\*).

- *status*

The queue status of the specified quota group is defined by the following bit-field combinations and decimal values:

**Table 4-3. Queue Status Bit-Fields**

Bit 1	Bit 0	Queue Status	Value (decimal representation)
0	0	active:open	0
0	1	active:closed	1
1	0	inactive:open	2
1	1	inactive:closed	3

For example, the queue status value of “3” means that the queue status is inactive (per *Bit 1*) and closed (per *Bit 0*).

- *recursive*

If this argument is set to “1”, the queue status is cleared for the sub-quotas of the specified quota group.

### Description

Use **queue\_status** with the `calcm_send_message` script to change or report the job queue status for the specified quota group. If no argument is specified, all job queues with a closed or

inactive status are reported. This information is passed to the calcm\_jobqueue\_app.tcl application which manages the job queue.

## Examples

### Example 1

The following example shows quota group D1 with status “1” which results in a queue status value of “1”, active:closed.

```
calcm_send_message queue_status D1 1
```

### Example 2

The following example shows quota group D1/P1 that inherits status “1” (from parent D1) along with its status “2” which results in a queue status value of “3”, inactive:closed.

```
calcm_send_message queue_status D1 1
calcm_send_message queue_status D1/P1 2
```

### Example 3

The following example resets the status for all queues to a queue status value of “0”, active:open.

```
calcm_send_message queue_status \* 0 1
```

## Related Topics

[calcm\\_send\\_message](#)

## **read\_conf**

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Reads the cluster configuration file and updates the cluster.

### **Usage**

**read\_conf**

### **Arguments**

None.

### **Description**

Use **read\_conf** with the calcm\_send\_message script to read the cluster configuration file and update the cluster. This information is passed to the calcm\_jobqueue\_app.tcl application, which then updates the cluster.

The remotes on unregistered nodes can still be active. To address this problem, you can set the REVOKEUNREGISTERED flag in the [calcm\\_rmanager\\_app.tcl](#) application to “1” to revoke them.

---

#### **Note**

 The `read_conf` *kill* argument and options are deprecated as of the 2020.1 release. You can use the REVOKEUNREGISTERED flag in the `calcm_rmanager_app.tcl` to indicate how unregistered remotes should be handled.

---

### **Examples**

```
calcm_send_message read_conf
```

### **Related Topics**

[calcm\\_send\\_message](#)

# register\_node

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Registers a specified remote node with the cluster manager along with an optional number of remote host job slots.

## Usage

**register\_node** *node\_name* ... [*count*]

## Arguments

- ***node\_name***  
Specifies the name of the remote node to register with the cluster manager. A cluster named in the *cluster.conf* file can also be used.
- ***count***  
Specifies an optional number of remote host job slots to register on *node\_name*.

## Description

Use **register\_node** with the `calcm_send_message` script to register a remote node and optionally specify the number of remote host job slots for the cluster manager. Multiple node names (space separated) can be specified with a single *count* option at the end of the argument.

## Examples

### Example 1

This example sends a message to register four remote host job slots on node “node123”.

```
calcm_send_message register_node node123 4
```

### Example 2

This example sends a message to register eight remote host job slots on node “node123” and node “node456”.

```
calcm_send_message register_node node123 node456 8
```

## Related Topics

[calcm\\_send\\_message](#)

## renew\_license

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Dynamically renews licenses from the CalCM license pool.

### Usage

**renew\_license {*license\_name* | “\*”}**

### Arguments

- ***license\_name***

Required argument that specifies to renew an individual (atomic) license name in the CalCM license pool. When this argument is specified, licenses of that name (feature) in the CalCM license pool are released and reacquired.

- “\*”

Required argument enclosed in quotation marks (“ ”) that specifies to renew all licenses in the CalCM license pool. When this argument is specified, all licenses in the CalCM license pool are released and reacquired.

### Description

Use **renew\_license** with the `calcm_send_message` script to renew an individual license feature name or to renew all licenses in the Calibre License Broker Daemon (LBD) license pool.

The check in and check out behavior between the LBD and the license manager with `renew_license` is as follows:

- When an existing license feature name is specified, the LBD checks back in or out the license feature from the license manager with the same original count.
- When “\*” is specified, the LBD checks back in all of its licenses to the license manager and then checks back out the same license features and the same count from the license manager.

Refer to the [LICENSE FEATURES](#) statement for more information on the license definition syntax.

### Examples

For example, to renew the `calnmopc` license feature, run the `calcm_send_message` command.

```
calcm_send_message renew_license calnmopc
```

This returns the following message.

```
Renewing atomic license feature, "calnmopc"
```

The CalCM daemon transcript (`calcmd.log`) shows the following information.

```
[2018-01-05 10:11:04] Received connection request from my_server-rh7:56092
[2018-01-05 10:11:04] my_server-rh7:56092/my_user: renew_license calnmopc
[2018-01-05 10:11:04] Renewing license feature, "calnmopc"
[2018-01-05 10:11:04] (LBD) [T2.C25481] Attempting atomic license re-read
for calnmopc, quantity 0
[2018-01-05 10:11:04] (LBD) 100 calnmopc licenses released.
[2018-01-05 10:11:04] (LBD) 100 calnmopc (2020.120) licenses acquired.
```

## Related Topics

[calcm\\_send\\_message](#)

## reset\_jobanalysis

Related application: [calcm\\_jobanalysis\\_app.tcl](#)

Updates the configuration for *calcm\_jobanalysis.tcl* without restarting the CalCM daemon. To be used with the CONFPATH parameter.

### Usage

**reset\_jobanalysis** [*configuration*]

### Arguments

- *configuration*

Optional argument to specify the configuration for *calcm\_jobanalysis.tcl*.

### Examples

```
% more conf/calcmd.conf
...
APPLICATION TCL $CALCM_APP/calcm_jobanalysis_app.tcl FILE [
    CONFPATH = $CALCM_HOME/conf/jobanalysis.conf
]
...
%
% ./jobqueue/calcm_send_message reset_jobanalysis

Reading job analysis configuration from \
<CALCM_HOME>/conf/jobanalysis.conf...
// -----
// Calibre Cluster Manager Configuration for JobAnalysis App
// This file is called by calcmd.conf -> calcm_jobanalysis_app.tcl ->
// CONFPATH
// -----
GROUPS = tech_node,module,layer,calibre_version,db_range(db_area)
COLUMNS = count(*),avg(runtime),max(runtime),max(max_nremotes), \
          max(max_master_lvheap)
RANGES = {db_range micro=0,5 tiny=5,20 small=20,100 medium=100,300 \
          large=300,500 huge=500,*} \
          {runtime_range short=0,1 medium=1,10 long=10,100 huge=100,*}
DBSTARTTIME = 01:00
```

### Related Topics

[calcm\\_send\\_message](#)

## reset\_license

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Dynamically adds or removes licenses from the CalCM license pool.

### Usage

**reset\_license** *license\_config\_file*

### Arguments

- *license\_config\_file*

Required argument that specifies the path to the CalCM license configuration file. A warning is issued if the filename is not specified.

### Description

Use **reset\_license** with the `calcm_send_message` script to modify the License Broker Daemon (LBD) license pool so that it matches the contents of the specified license configuration file. The syntax of the license definitions in the license configuration file is:

***feature\_name* [exact\_access\_date] = *number***

Refer to the [LICENSE FEATURES](#) statement for more information on the license definition syntax. Here is an example of a license configuration file:

```
calopcpco = 5
calmtopcpco = 5
calopcsbar = 10
calibredrc = 10
calibrehdrc = 10
calnmopc = 10
```

---

#### Caution

 If a license in the license pool is omitted from the license configuration file specified by `reset_license`, then the LBD removes the license from the pool. For example, if the license pool includes 5 *calfracturej* licenses and you do not define the *calfracturej* license in the license configuration file, then these licenses are removed from the license pool.

---

Licenses in use by Calibre jobs running under CalCM cannot be removed from the license pool. For example, if the license pool contains 50 *calnmopc* licenses and 10 of those licenses are in use, then the maximum number of *calnmopc* licenses that can be removed from the pool is 40.

In the event licenses are being added and sufficient quantity is not immediately available, the `reset_license` command will fail and the LBD will maintain the pool unchanged. The `reset_license` command does not make any attempt to queue or retry for licenses.

## Examples

In the following example, the license pool for jobs running under CalCM includes 50 *calopcpro*, 50 *calmtopcpro*, 75 *calibredrc*, 75 *calibrehdrc*, and 10 *calnmopc* licenses. You create a license configuration file (*license.conf*) that specifies the following licenses for the license pool:

```
calopcpro = 5
calmtopcpro = 5
calopcsbar = 10
calibredrc = 10
calibrehdrc = 10
calnmopc = 10
```

Use the following command to submit the license configuration file (*license.conf*):

```
calcm_send_message reset_license license.conf
```

The transcript identifies the changes made by the License Broker Daemon (LBD) to the license pool. The *calopcpro* and *calmtopcpro* licenses are each reduced by 45, 10 *calopcsbar* licenses are added, the *calibredrc* and *calibrehdrc* licenses are each reduced by 65, and there is no change to the *calnmopc* licenses:

```
...
(LBD) [T3] Updating license pool:
(LBD) [T3] - calopcpro: 50 -> 5 (-45)
(LBD) [T3] - calmtopcpro: 50 -> 5 (-45)
(LBD) [T3] - calopcsbar: 0 -> 10 (+10) [NEW]
(LBD) [T3] - calibredrc: 75 -> 10 (-65)
(LBD) [T3] - calibrehdrc: 75 -> 10 (-65)
(LBD) [T3] - calnmopc: 10 -> 10 (+0) [NO CHANGE]
(LBD) 10 calopcsbar (2016.020) licenses acquired
(LBD) 45 calopcpro licenses released
(LBD) 45 calmtopcpro licenses released
(LBD) 65 calibredrc licenses released
(LBD) 65 calibrehdrc licenses released
(LBD) [T3] License pool update complete.
...
```

## Related Topics

[calcm\\_send\\_message](#)

# reset\_notification

Related application: [calcm\\_notification\\_app.tcl](#)

Reconfigures the attribute trackers and filters used for CalCM notifications.

## Usage

**reset\_notification** [*file\_path*]

## Arguments

- *path*

Optional argument that specifies the path to a notifications configuration file.

## Description

Use **reset\_notification** with the `calcm_send_message` script to read a file that reconfigures the attribute trackers and filters used for notifications. The following attribute trackers and filters are used in the next update cycle:

- ATTRTRACKERS
- DAEMONFILTERS
- JOBFILTERS
- MASTERFILTERS
- REMOTEFILTERS
- VERSIONFILTERS

If *file\_path* is not specified, the filename specified in the [calcm\\_notification\\_app.tcl](#) NOTIFYCONF configuration variable is used.

---

### Note

 By specifying notifications outside of the main CalCM configuration file (`calcconf.conf`), as with `reset_notification` and NOTIFYCONF, you can update the notification settings without restarting the CalCM daemon.

---

## Examples

```
calcm_send_message reset_notification notifications.conf
```

## Related Topics

[calcm\\_send\\_message](#)

## reset\_quota

Related application: [calcm\\_rmanager\\_app.tcl](#)

Reads the quota configuration and re-calculates the quota.

### Usage

**reset\_quota *quota\_configuration***

### Arguments

- ***quota\_configuration***

Specifies the path to the quota configuration table. The default name is *calcm\_quota.conf*.

### Description

Use **reset\_quota** with the `calcm_send_message` script in order to read the specified quota configuration and re-calculate the quota. This information is passed to the `calcm_rmanager_app.tcl` application which resets the quota hierarchy.

### Examples

```
calcm_send_message reset_quota calcm_quota.conf
```

### Related Topics

[calcm\\_send\\_message](#)

# resume\_job

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Resumes a suspended job under CalCM.

## Usage

**resume\_job** *job\_id*

## Arguments

- *job\_id*

Specifies the id of the job that you want to resume.

## Description

Use **resume\_job** with the calcm\_send\_message script to resume a CalCM job that is suspended with the [suspend\\_job](#) command. The normal Dynamic CPU Allocation (DCA) for the job is also resumed. This information is passed to the calcm\_jobqueue\_app.tcl application which resumes the specified job.

## Examples

```
calcm_send_message resume_job 12345
```

## Related Topics

[calcm\\_send\\_message](#)

## **set\_budget**

Related application: [calcm\\_tat\\_app.tcl](#)

Sets or changes the budget for the specified job.

### **Usage**

**set\_budget** *job\_id* *budget\_string*

### **Arguments**

- ***job\_id***

Specifies the id of the job whose budget you want to change.

- ***budget\_string***

A string specifying the name of the operation and an associated budget value (specified in minutes). The format of *budget\_string* is similar to that used for the [JOB BUDGET](#) configuration statement.

### **Description**

Use **set\_budget** with the calcm\_send\_message script to set or change the budget for one or more operations associated with a specific job. This information is passed to the calcm\_tat\_app.tcl application which assigns the budget to the specified job.

### **Examples**

```
calcm_send_message set_budget 7783 "nmopc=300; _EXTRA_=30"
```

### **Related Topics**

[calcm\\_send\\_message](#)

## **set\_priority\_bump**

Related application: [calcm\\_tat\\_app.tcl](#)

Sets or changes the priority for the specified job.

### **Usage**

**set\_priority\_bump *job\_id priority\_bump***

### **Arguments**

- ***job\_id***

Specifies the id of the job whose priority you want to change.

- ***priority\_bump***

Specifies a priority for a job. The priority for the specified job changes if this value is greater than the original priority.

### **Description**

Use **set\_priority\_bump** with the calcm\_send\_message script to set or change the priority for one or more operations associated with a specific job. This information is passed to the calcm\_tat\_app.tcl application which changes the priority bump for the specified job.

### **Examples**

```
calcm_send_message set_priority_bump 8663 40
```

### **Related Topics**

[calcm\\_send\\_message](#)

## span\_log

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Signals calcmd to span a log file as specified.

### Usage

**span\_log** [*log\_filename* [*log\_size[unit]*]]

### Arguments

- *log\_filename*

Optional argument that specifies the name of the calcmd log file.

- *log\_size[unit]*

Optional argument that specifies the maximum log file size and the unit. The *unit* specifications are ‘k’, ‘m’, ‘g’, or ‘t’. If *unit* is not specified, the default unit is byte.

### Description

Use **span\_log** with the `calcm_send_message` script and the **calcmd** -log command option to signal the calcmd to archive the current log file and start a new log file with the specified file name and file size limit. The new log file continues the transcript information from the point where the archived log file ends. File name collisions are avoided by the automatic appending of the ‘.’ and epoch time at the end of the log file name. The time appended is the epoch time when the new log file was created. If specified, *log\_filename* and *log\_size* are inherited in the spanned log file(s).

### Examples

In this example the calcmd log file is spanned into a new log file named “mylog” with a maximum file size limit of 2GB.

```
calcm_send_message span_log mylog 2g
```

### Related Topics

[calcm\\_send\\_message](#)

# **suspend\_job**

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Suspends a CalCM job.

---

## **Note**

 You can also access this command from the CalCM dashboard web interface with the required authentication.

---

## **Usage**

**suspend\_job** *job\_id*

## **Arguments**

- *job\_id*

Specifies the id of the job that you want to suspend.

## **Description**

Use **suspend\_job** with the `calcm_send_message` script to suspend a CalCM job. By default, if the job is running (active) and a **suspend\_job** command is issued, the number of remotes assigned to the job is adjusted based on the minimum number of remotes (*min* value) allocated in the cluster string. This number can be overridden with the SUSPENDMIN argument for [calcm\\_rmanager\\_app.tcl](#). In the case of a pending job, the suspended job is not launched until it is resumed with the [resume\\_job](#) command. This information is passed to the `calcm_jobqueue_app.tcl` application which suspends the specified job.

## **Examples**

```
calcm_send_message suspend_job 12345
```

## **Related Topics**

[calcm\\_send\\_message](#)

## unregister\_node

Related application: [calcm\\_jobqueue\\_app.tcl](#)

Registers a specified node with the cluster manager.

### Usage

**unregister\_node** *node\_name* ...

### Arguments

- ***node\_name***

Specifies the name of the node to unregister with the cluster manager. A cluster name specified in the *cluster.conf* file can also be used.

### Description

Use **unregister\_node** with the `calcm_send_message` script to unregister a node and revoke the specified node from the cluster manager. Multiple node specifications (space separated) and name matching with wildcard (\*) and (?) expressions are supported.

### Examples

#### Example 1

This example sends a message to unregister and revoke node “node123”.

```
calcm_send_message unregister_node node123
```

#### Example 2

This example sends a message to unregister and revoke node “node123” and all nodes with names that match “node45\*”.

```
calcm_send_message unregister_node node123 node45*
```

### Related Topics

[calcm\\_send\\_message](#)

# CalCM Tcl Application Reference

---

This section provides reference information on the CalCM Tcl applications (default unless noted otherwise).

<b>calcmon_app.tcl</b> .....	<b>162</b>
<b>calcmon_http_server_app.tcl</b> .....	<b>166</b>
<b>calcmon_jobanalysis_app.tcl</b> .....	<b>169</b>
<b>calcmon_jobqueue_app.tcl</b> .....	<b>174</b>
<b>calcmon_notification_app.tcl</b> .....	<b>182</b>
<b>calcmon_rmanager_app.tcl</b> .....	<b>195</b>
<b>calcmon_rmonitor_app.tcl</b> .....	<b>200</b>
<b>calcmon_systemanalysis_app.tcl</b> .....	<b>203</b>
<b>calcmon_tat_app.tcl</b> .....	<b>205</b>
<b>calcmon_tcl_app.tcl</b> .....	<b>210</b>

## **calcm\_hmonitor\_app.tcl**

Retrieves the hardware information monitored in the CalScope or Ganglia server.

---

### **Note**

 Requires CalCM+ advanced features licenses. Refer to the *Calibre Administrator's Guide* for complete licensing information.

---

### **Usage**

```
APPLICATION TCL path/calcm_hmonitor_app.tcl [ FILE '['  
    [BASEURL = URL]  
    [CALSCOPEADD = flag]  
    [CALSCOPECLI = path]  
    [CALSCOPEREMOVE = boolean]  
    [CLUSTER = names]  
    [ENABLECALSCOPE = boolean]  
    [HOLDINTERVAL = sec]  
    [HOST = hostname[:port]]  
    [RESETCMD = cmd]  
    [RESETINTERVAL = sec]  
    [RESETMASTERMAX = num]  
    [RESETREMOTEMAX = num]  
    [RETRYCOUNT = num]  
']' ]
```

### **Arguments**

- **BASEURL = *URL***

Specifies the URL used to build hypertexts to the hardware information URL. The URL is used in the calcm\_http\_server\_app.tcl application only for the authorized users.

- **CALSCOPEADD = *flag***

Specifies a flag value that controls the type of node to add to CalScope for hardware monitoring. The following flag values only apply when ENABLECALSCOPE = 1 (the default flag value is 3):

- 0 — Do not add any node
- 1 — Add only the registered node
- 2 — Add only the unregistered node

3 — Add all nodes (default)

- **CALSCOPECLI = *path***

Specifies the path to the CalScope command-line interface (CLI), which enables interaction with the monitored resources. The default path is *\$MGC\_HOME/bin/calscope*.

For information on the CalScope software tool, see the *Calibre System Monitoring Solution (CalScope) User's Manual* on [Support Center](#).

- **CALSCOPEREMOVE = *boolean***

Specifies whether to use the remove command every update cycle when synchronized with the CalScope server. The default is 1, specifying to use the remove command. If the value is 0, the remove command is not used. The remove command only applies when **ENABLECALSCOPE** = 1.

- **CLUSTER = *names***

Specifies the cluster names defined in the CalScope or Ganglia server. Only hosts specified in the clusters are monitored. Multiple comma-separated clusters can be specified.

- **ENABLECALSCOPE = *boolean***

Specifies whether to use CalScope instead of Ganglia for hardware monitoring. The default is 0 (Ganglia). If specified as 1, the *calcm\_hmonitor\_app.tcl* application uses CalScope for hardware monitoring and starts synchronizing the host status with CalScope.

During the synchronization, only the hosts in CalCM are added to CalScope through the CalScope CLI add command. Likewise, only the hosts in CalScope are removed from CalScope through the CalScope CLI remove command. For example, add and then remove two monitored hosts from the CalScope session:

```
calscope hw_monitor add -hostnames host03 host04
calscope hw_monitor remove -hostnames host03 host04
```

- **HOLDINTERVAL = *sec***

Specifies a hold interval in seconds for the primary and remote hosts. If the **RESETINTERVAL** keyword is specified and the time difference between the boot time and current time is equal to or greater than the interval, the node is put on hold. During the hold, no primary or remote processes can be launched. The default hold interval is 2 \* **RESETINTERVAL** time. No hosts are held if **RESETINTERVAL** is not specified.

- **HOST = *hostname[:port]***

Specifies the host name where the Ganglia tool is installed. The optional port number can be specified if different from 8652.

- **RESETCMD = *cmd***

Specifies the command line executable to reset hosts. The default value is “\$rsh \$hostName \$rshFlags reboot &”.

**Note**



The default value for the \$rsh variable in CalCM is “ssh”.

---

- **RESETINTERVAL = sec**

Specifies the interval in seconds to reset primary and remote hosts. If the difference between the boot time and current time is equal to or greater than the interval and there is no Calibre process running, the node is reset. The default is 0, no hosts are reset.

- **RESETMASTERMAX = num**

Specifies the maximum number of master hosts that are reset during the update cycle. The default is 0, no master hosts are reset.

- **RESETREMOTEMAX = num**

Specifies the maximum number of remote hosts that are reset during the update. The default is 20. If the value is 0, no remote hosts are reset.

- **RETRYCOUNT = num**

Specifies a number for the retry count to detect whether a host is up or down. The default value is 4. The equation is as follows:

```
$host(TN) <= $host(TMAX) * $retryCount
```

where TN is the number of seconds since the metric was last updated and TMAX indicates the timeout in seconds. CalCM uses the TN and TMAX values returned by Ganglia to determine if a host needs to be reset.

## Description

The *calcm\_hmonitor\_app.tcl* application retrieves and updates the hardware monitoring information from the specified CalScope or Ganglia server.

## Examples

### Example 1

This example retrieves the hardware monitoring information from a CalScope server.

```
APPLICATION TCL calcm_hmonitor_app.tcl FILE [
    ENABLECALSCOPE= 1
    CALSCOPECLI = $MGC_HOME/bin/calscope
    CALSCOPEADD = 3
    CALSCOPEREMOVE = 0
]
```

### Example 2

This example retrieves the hardware monitoring information from a Ganglia server.

```
APPLICATION TCL calcm_hmonitor_app.tcl FILE [
    HOST= ganglia.<my_ganglia_host>.com
    CLUSTER = CalCM_Primaries,CalCM_Remotes
    BASEURL = http:/ganglia.<my_ganglia_host>.com/<path>
]
```

## **calcm\_http\_server\_app.tcl**

A GUI HTTP interface to monitor job submission and progress, node allocation and network utilization.

### **Usage**

```
APPLICATION TCL path/calcm_http_server_app.tcl [ FILE '['  
    [DASHBOARDDB = file_path]  
    [DASHBOARDPORT = port_num]  
    [DASHBOARD_SESSION_LIFETIME = time]  
    [ENABLE_DASHBOARD = flag]  
    [DISPLAYROWS = num]  
    [LDAP_EMAIL_KEY = attrib_key]  
    [LDAP_FNAME_KEY = attrib_key]  
    [LDAP_LNAME_KEY = attrib_key]  
    [LDAP_UNAME_KEY = attrib_key]  
    [LDAPSEARCHBASE = ldap_search_base]  
    [LDAPURL = ldap_url]  
    [ROOT = dir_path]  
    [USEJAVACHART = flag]  
' ] ]
```

### **Arguments**

- **DASHBOARDDB = *file\_path***  
Specifies the path to the CalCM dashboard database used for dashboard-specific data which includes user information and permission groups. If not specified, this path defaults to an SQLite®<sup>1</sup> database called *calcm\_web.db* inside the ROOT directory for the HTTP server.
- **DASHBOARDPORT = *port\_num***  
Specifies the port on which the CalCM dashboard server is run. If a port is not specified, a random port is selected and shown in the transcript.  
It is recommended that you specify a port number.
- **DASHBOARD\_SESSION\_LIFETIME = *time***  
Specifies the CalCM dashboard user-session lifetime in seconds before automatic logout due to inactivity. The default is 600.

---

1. SQLite® is a registered trademark of Hipp, Wyrick & Company, Inc.

- **ENABLE\_DASHBOARD = *flag***

Specifies whether to enable the CalCM dashboard web-based GUI HTTP interface. Valid values include:

1 — Enables the CalCM dashboard web-based GUI HTTP interface. This is the default supported interface starting CalCM version 2018.1 P6 (2018.1\_37.28).

0 — Disables the dashboard web-based GUI HTTP interface.

- **DISPLAYROWS = *num***

Specifies the number of rows used for displaying the job tables. The default is 10.

- **LDAP\_EMAIL\_KEY = *attrib\_key***

Specifies the key used by the CalCM dashboard for finding the email attribute value in an LDAP user entry. By default, email address extraction of LDAP users is disabled. Refer to your system administrator and your Siemens representative for the settings required to enable the LDAP authentication for the CalCM dashboard.

- **LDAP\_FNAME\_KEY = *attrib\_key***

Specifies the key used by the CalCM dashboard for finding the first-name attribute value in an LDAP user entry. By default, first-name extraction of LDAP users is disabled.

- **LDAP\_LNAME\_KEY = *attrib\_key***

Specifies the key used by the CalCM dashboard for finding the last-name attribute value in an LDAP user entry. By default, last-name extraction of LDAP users is disabled.

- **LDAP\_UNAME\_KEY = *attrib\_key***

Specifies the LDAP attribute key for usernames. This configuration is required to enable LDAP user management. By default, the LDAP user management is disabled.

- **LDAPSEARCHBASE = *ldap\_search\_base***

Specifies the LDAP search base to be used by the CalCM dashboard when looking for user entries. By default, LDAP user management is disabled.

- **LDAPURL = *ldap\_url***

The URL for the LDAP server used for CalCM dashboard user management. By default, if unspecified, LDAP user management is disabled.

- **ROOT = *dir\_path***

Specifies the path to the directory to be used as the root directory for the HTTP server. The default is as follows:

*/html* — CalCM dashboard interface where *html* is the HTML root directory

- USEJAVACHART = *flag*

Provides an option that allows you to disable the default usage of Highchart JS technology, in order to revert back to the older GNU plot technology when using the HTTP Server application. Valid values include:

- 0 — The Highcharts JS application is not used for plotting.
- 1 — The Highcharts JS application is used for plotting (default).

---

#### **Note**

 Starting with the 2013.4 CalCM production release, the HTTP server application incorporates Highsoft AS's Highcharts JS plotting technology by default. This alteration improves your ability to extract useful and needed information from the plots in an easy-to-use fashion via real-time, responsive interaction with the plotting objects on the webpage. The Highcharts JS software is owned and licensed by Highsoft Solution AS.

---

## Description

The *calc\_http\_server\_app.tcl* application provides a GUI HTTP interface with a configuration consisting of multiple web pages that are populated on demand with job, resource, general information, and monitoring information. The job and resource pages include links to more detailed information, including a plot from the SQL database, about a job or node.

For command statements using Lightweight Directory Access Protocol (LDAP), refer to your system administrator and your Siemens representative for the required settings to enable LDAP authentication for the CalCM dashboard web interface.

See also “[Accessing and Updating Settings in the CalCM Dashboard](#)” on page 87 for information on updating user and group settings in the dashboard.

## Examples

### Example 1

```
APPLICATION TCL calc_http_server_app.tcl FILE [
    ROOT = /calcm/public_html
]
```

### Example 2

To use the CalCM dashboard web application, customize the Tcl applications HTTP Server section of the CalCM configuration file (*calcmd.conf*).

```
// "Http Server" Tcl application.
APPLICATION TCL $CALCM_APP/calcm_http_server_app.tcl FILE [
    //Path for job database
    JOBDB = $JOB_DB
    ENABLE_DASHBOARD = 1
    DASHBOARDPORT = 9902
]
```

## calcm\_jobanalysis\_app.tcl

Categorizes the job information based on groups and calculates the result for reference. The categorized information can be retrieved using a command-line interface.

---

### Note

---

 Requires CalCM+ advanced features licenses. Refer to the [Calibre Administrator's Guide](#) for complete licensing information.

---

---

### Note

---

 To use this CalCM application, you must:

- Use it with the *calcm\_notification\_app.tcl* application
  - Define SCANJOBINFO=1 in the *calcm\_jobqueue\_app.tcl* application
- 

## Usage

```
APPLICATION TCL path/calcm_jobanalysis_app.tcl [ FILE '['  
    [CONDITIONS = sql_filter]  
    [CONFPATH = path]  
    [COLUMNS = columns]  
    [DBSTARTTIME = path]  
    [GROUPS = groups]  
    [RANGES = ranges]  
'']
```

## Arguments

- CONDITIONS = *sql\_filter*

Optional keyword that can be used to ignore specific jobs from the history when building statistics for job categorization. It accepts filters specified with the SQL language. For example, if you recently stopped using an old Calibre branch 2009.1 and want to ignore these results from your statistics for more accuracy, you would define the following:

```
CONDITIONS = "calibre_version != 'v2009.1'"
```

- CONFPATH = *path*

Optional keyword that can be used to store the configuration of this CalCM application outside of the main configuration file so that you can update the values without restarting the CalCM daemon.

- **COLUMNS = *columns***

Optional keyword to specify how the aggregated columns are calculated. The specified functions can be used as a column name in the info\_analysis table. For example (specify in a single line):

```
COLUMNS = count(*), avg(runtime), max(runtime), max(max_nremotes), \
max(max_master_lvheap)
```

The listed columns are those that the main command calcm\_jobanalysis will print when run. It is recommended to output at least count(\*) so that you know how many past jobs have been used to compute the statistics, and whether the number is high enough to be reliable.

It is also recommended to include max(runtime) and max(master\_lvheap) from the history of similar jobs.

Legal keywords: max\_master\_lvheap, max\_nremotes, remote\_cpu\_hours, master\_cpu\_hours, runtime, active\_remote\_hours, allocated\_remote\_hours.

- **DBSTARTTIME = *path***

Optional keyword to specify when the database categorization occurs. The format is as follows.

- For daily categorization, specify time as *hour:minute*. For example, to categorize the database at 1:00 a.m. each day, specify “01:00”.
- For weekly categorization, specify time as *day-of-week-hour:minute*. For example, to categorize the database on Monday at 8:00 a.m. each week, specify “Monday-08:00”.
- For monthly categorization, specify time as *week-of-the-month-day-of-week-hour:minute*. For example, to categorize the database at 1:00 a.m. on the first Sunday of each month, specify “1st-Sunday-01:00”. To categorize on the last Sunday of each month at 1:00 a. m., specify “4th-Sunday-01:00”.

- **GROUPS = *groups***

Optional keyword to specify the columns in the info\_jobs table to use for grouping during categorization. The groups are comma-separated. The range function specified in RANGES can be used. For example (specify in a single line):

```
GROUPS = tech_node,module,layer,calibre_version,db_range(db_area)
```

This keyword defines which jobs are “similar” to the one you are trying to categorize. Defining these groups is a compromise between accuracy (fewer jobs but very similar) and a larger dataset (many jobs but may include jobs different from the one you are trying to categorize).

- **RANGES = *ranges***

Optional keyword to specify the range functions to use in GROUPS and COLUMNS. The name can be used as a function, and the value within the specified range is stored as the tag. The syntax is as follows:

```
{name tag=from,to...}
```

For example:

```
RANGES = {db_range micro=0,5 tiny=5,20 small=20,100 medium=100,300 \
           large=300,500 huge=500,*} \
           {runtime_range short=0,1 medium=1,10 long=10,100 huge=100,*}
```

This CalCM application is intended to give a categorization of jobs, but it cannot make predictions. It is based on Calibre jobs that previously ran on your cluster with your designs, but not exactly the same. In order to help with categorization, it is necessary to group at least the layout sizes into “buckets,” so that you define what a “similar” job is. Without these buckets, CalCM will be unable to group together jobs for similar layout sizes.

Use mm<sup>2</sup> for db\_range. In the example above, a new layout of 70 mm<sup>2</sup> will be categorized as “small” and statistics for all “small” layouts will be used to categorize the job.

## Description

The *calcめ\_jobanalysis\_app.tcl* application categorizes the job information stored in the info\_jobs table based on the groups specified. The calculated result of the information can be retrieved from the info\_analysis table and referenced using the command-line interface created in *jobqueue/calcめ\_jobanalysis*.

Categorizing jobs provides useful data that can be used in many different ways, including the following:

- Job categorization statistics enable you to make broad predictions on the behavior of any new Calibre job, based on the behavior of similar jobs that ran previously on your cluster.
- Categorizing the maximum memory size enables you to select a master host with enough memory to avoid using swap memory or possibly to avoid a crash.
- Categorizing a relevant run time enables you to configure a relevant run time outlier alert, so that any misbehavior can be detected earlier while avoiding false warnings.
- Categorizing maximum scalability enables you to select correct CalCM parameters, such as MAXCOUNT.
- If you know a job will be “small” in terms of memory, CPU, and scalability, you can run multiple ones on the same master host to allow a global cluster utilization.

The syntax follows SQL syntax convention as follows:

```
calcめ_jobanalysis [-select <column>...] -group <group>...
```

For example:

```
calcめ_jobanalysis -group 22,MODULE,M1,v2020.3_0.69,medium
calcめ_jobanalysis -select 'count(*)', 'max(runtime)', \
    'db_range(db_area)', 'max(max_master_lvheap)' \
    -group 22,MODULE,M1,v2020.3_0.69,medium
```

You can specify a threshold to query the row where count(\*) exceeds the specified value.

The -group option must be used accordingly to the GROUPS parameter defined in the CalCM application. By default, the output of the following command will be the COLUMNS parameters defined in the CalCM application.

```
$CALCM_HOME/jobqueue/calcめ_jobanalysis -group 12NM, opc, XW, \
    v2014.1_76.63, micro
```

Multiple groups can be specified to query subsequently until the row that meets the threshold is found. The percent sign (%) can be specified in the group to match everything.

```
calcめ_jobanalysis -threshold <threshold> [-select <column>...] \
    -group <group>... [-group <group>...] ...
```

For example:

```
calcめ_jobanalysis -threshold 5 \
    -group 22,MODULE,M1,v2020.3_0.69,medium \
    -group 22,MODULE,M1,v2020.3_0.69,small \
    -group 22,MODULE,M1,v2020.3_0.69,% \
    -group 12,MODULE,M1,v2020.3_0.69,%
```

---

**Tip**  If your history of jobs is not statistically large enough to produce relevant values, the accuracy of the categorization is reduced. In this case, try to select jobs exactly similar first, then expand to similar jobs if there are not enough jobs, and finally only approximately-similar jobs.

---

The following options are supported to query and modify the info\_jobs table contents. All arguments must be specified in SQL syntax.

```
calcめ_jobanalysis -select <column>... [-where <condition>...]
    -insert <column>... -values <value>...
    -delete -where <condition>...
    -update <assignment>... -where <condition>...
```

## Examples

### Example 1

To get an idea of how a 12 nm OPC job on layer Metal1 with Calibre 2020.2 will behave on your new medium-sized chip (100-300 mm<sup>2</sup>), based on your history, you would run the following:

```
% ./jobqueue/calcm_jobanalysis -group 12nm, OPC, Metal1, \
v2020.2_58.53, medium
```

### Example 2

```
APPLICATION TCL calcm_jobanalysis_app.tcl FILE [
    GROUPS = tech_node,module,layer,calibre_version,db_range(db_area)
    COLUMNS =
        count(*),avg(runtime),max(runtime),count(*),avg(runtime),count(*),avg(runtime),
        max(runtime),max(max_nremotes),max(max_master_lvheap)
        RANGES = {db_range micro=0,5 tiny=5,20 small=20,100 medium=100,300
        large=300,500 huge=500,*} \
            {runtime_range short=0,1 medium=1,10 long=10,100 huge=100,*}
        CONDITIONS = "calibre_version != 'v2020.3'"
        DBSTARTTIME = 23:00 ]
```

## **calcm\_jobqueue\_app.tcl**

Creates a job dispatcher and the configuration files to run each job.

### **Usage**

**APPLICATION TCL** *path/calcm\_jobqueue\_app.tcl* [ FILE '['  
[ALLPENDINGREASON = [false | true]]  
[ANNOUNCERTRY = *count*]  
[CHECKREMOTEALIVE = *flag*]  
[CLUSTERCONF = *cluster\_conf\_filename*]  
[CLUSTERCOUNT = *cpu\_count*]  
[CLUSTERLIMIT = *count*]  
[CLUSTERMAX = *maxcount*]  
[CLUSTERMIN = *mincount*]  
[CONFDIRNAME = *confdir\_name[,perm]*]  
[COPYCONFFILES = [true | false]]  
[ENDGUID = *num*]  
[GUIDDIGIT = *num*]  
[INITREMOTEPERNODE = *count*]  
[JOBDB = *file\_path*]  
[JOBNOTIFYCMD = *cmd*]  
[JOBNOTIFYEMAIL = *email*]  
[KEEPINITIAL = *flag*]  
[LAUNCHDELAY = *delay\_time*]  
[NODENOTIFY = *flag*]  
[NODENOTIFYCMD = *cmd*]  
[PRUNEDBINTERVAL = *interval*]  
[PRUNEDBSAVETIME = *value*]  
[PRUNEDBSTARTTIME = *time*]  
[QUEUEDIR = *queue\_dir*]  
[QUOTALEVELLIMIT = *num*]  
[QUOTAOVERDRAFT = *num*]  
[QUOTAOVERDRAFTLEVEL = *range*]

[QUOTAPOLICY = *quota\_policy\_list*]  
[SCANJOBS = *flag*]  
[SCANJOBINFO = *flag*]  
[STARTGUID = *num*]  
[VERSIONFILTERS = *filters*]  
' ] ]

## Arguments

- **ALLPENDINGREASON** = [false | true]

Specifies whether to record the pending reasons for all jobs in the queue. The default is “false”, which specifies to record only the reasons for the jobs on the top of the queue. The pending reasons appear as messages in the CalCM daemon transcript along with the queue information.

- **ANNOUNCERETRY** = *count*

Specifies the amount of time in which the CalCM should reannounce a job to the CalCM daemon. This parameter is used when CalCM announces a job to the daemon and it fails for some reason, such as the restart of the daemon. The default is 300 seconds.

- **CHECKREMOTEALIVE** = *flag*

Specifies whether to check the status of remote hosts before allocating these hosts to the job. The remote host is not allocated if there is no response. Valid values include:

0 — No check is performed (default).

1 — Remote hosts are checked for a response. If there is no response (failed state), the remote is not allocated for the job.

-*n* — After *n* consecutive failures, the failed remote is unregistered from the node list.

- **CLUSTERCONF** = *cluster\_conf\_filename*

Specifies the filename for the cluster configuration file. The default is none, which means no cluster configuration is specified.

- **CLUSTERCOUNT** = *cpu\_count*

Specifies the total number of CPUs on the remote host(s) to which the primary host will connect. The default is 1.

- **CLUSTERLIMIT** = *count*

Specifies a limit for the number of jobs that can be launched based on the number of *max* remotes specified in the job configuration (*job.conf*) file. The job launch behavior is controlled by comparing the CLUSTERLIMIT value to a sum of the **REMOTE MAX** values as follows:

- The REMOTE MAX values (cluster string *max* values) of all running (active) jobs are summed with the REMOTE MAX value of the next job in queue.

- If this sum is larger than the CLUSTERLIMIT value, then no job is launched and the remaining jobs are suspended, even if free job slots are available on the primary host(s).
- **CLUSTERMAX = *maxcount***  
Specifies the maximum number of remote connections for the Calibre job. The default is 100.
- **CLUSTERMIN = *mincount***  
Specifies the minimum number of remote connections that must be established in order for the Calibre job to run. The default is 1.
- **CONFDIRNAME = *confdir\_name,perm***  
Specifies the directory to create run and configuration files (*calcm\_run*, *calcm\_mtflex.conf*, *calcm\_launch.tcl*, and *calcm\_announce.tcl*) for a job. The default is “*.jobID*”. Optionally the permission of the directory and the files created under it can be specified after “*,*”. This example creates the directory with the job ID and the permission read/write/execute for all:  
CONFDIRNAME = \$jobID,777.  
A Tcl expression can be used to create multiple hierarchies. For example,  
[expr \$jobID/100000\*100000]/\$jobID
- **COPYCONFFILES = [true | false]**  
Specifies the version (and behavior) of the *calcm\_submit\_job* script that is created in the *CALCM\_HOME/jobqueue* directory. By default the parameter is “*true*”, and the created version of the *calcm\_submit\_job* script copies the job related files (*job.conf* and *job.env*) to CONFDIRNAME in the *calcm\_jobqueue\_app.tcl* application. If the COPYCONFFILES parameter is set to “*false*”, the created version of the *calcm\_submit\_job* script leaves the job related files in their original directory locations.
- **ENDGUID = *num***  
Specifies the end job ID. The default is  $(10^6) - 1$ .
- **GUIDDIGIT = *num***  
Specifies the number of characters (alphanumeric) used for a job ID. In most cases, numerical job IDs are used, so the maximum job ID is  $(10^num) - 1$ . The default is 6, meaning that a job ID is stored using six characters only.  
When CalCM is used in platform mode (IBM® Platform™ LSF® (LSF) or Univa® Grid Engine® (Grid)), commonly, the platform tool’s job ID is re-used as the CalCM job ID. In this case, the GUIDDIGIT number must be large enough to store the platform tool’s job ID. The default number for LSF is six characters, but this number can be increased using the LSF MAX\_JOBID parameter. The Grid platform often uses eight characters, but the MAX\_SEQNUM constant can increase this number. See “[Platform Tool Job and Remotes Control](#)” on page 382 for more information on using platform tools with CalCM.

- **INITREMOTEPEERNODE = *count***

Specifies the maximum number of initial remotes per node. The default is 1024.

- **JOBDB = *file\_path***

Specifies the path to the SQL database file for jobs. The default is *calcm\_job.db*.

For backward compatibility, you can migrate the data from the job database to another version of the database by setting the CALCM\_MIGRATE\_DB environment variable to 1. This allows you to move to a different version of the database and not lose any data when the database schema changed. The supported changes in the schema include:

Add column

Remove column

Change the data type of a column to “TEXT”

The database file with the old schema is renamed to “*databaseFileName.db.backup*”.

- **JOBNOTIFYCMD = *cmd***

Specifies an additional command that sends a notification about the job status change and reports the notification to the [calcm\\_notification\\_app.tcl](#). The default is empty (no notification). An example of this command using the Linux mail command is as follows:

`JOBNOTIFYCMD = {mail -s $subject $user << $body &}`

where *cmd* is enclosed in braces ({}), and both \$subject and \$body are set by the calcmd.

- **JOBNOTIFYEMAIL = *email***

Specifies an additional e-mail address that receives the job notification status.

- **KEEPINITIAL = *flag***

Specifies the flag that sets “JOB FLAG 1” to keep the initial remotes launched in CalCM from being revoked if certain job mode and RDS usage conditions are met. The following values specify the conditions to set the flag:

1 — RDS is used.

2 — RDS is allocated automatically in Calibre.

4 — HYPER mode is enabled.

8 — HYPER REMOTE mode is enabled (default).

You can choose the conditions when the initial remote hosts should be kept for all Calibre jobs, by summing the values and specifying this sum to the KEEPINITIAL keyword.

For example, if you want CalCM to keep the initial remotes allocated automatically for all Calibre jobs using RDS technology and all Calibre jobs launched with -hyper remote mode, specify “KEEPINITIAL 11”, because the sum of the values is  $1+2+8 = 11$  (includes 8 by default).

- **LAUNCHDELAY = *delay\_time***

Specifies the delay time between remote launches. Units are in milliseconds. It is a value or a Tcl expression respectively, and can be specified as:

LAUNCHDELAY = 500

LAUNCHDELAY = [expr round(rand() \* 500)]

If the **LAUNCH DELAY** statement is specified in the *job.conf* file, it overrides LAUNCHDELAY.

- **NODENOTIFY = *flag***

Specifies a flag to notify when there is a node status change as in the case when a node is unregistered. The following flag values control the notification behavior:

0 — The default value. This value notifies of the node status change only after all the processes on the node are removed.

1 — This value notifies the removal of a node after it is unregistered.

2 — This value notifies the removal of all nodes that are unregistered.

- **NODENOTIFYCMD = *cmd***

Specifies an additional command that is run when sending a notification on the node status change. The variable \$flag is set by the value specified in NODENOTIFY and the \$nodeName for the removed node names. An example of the command usage is as follows (the default is empty):

```
NODENOTIFYCMD = {echo "$flag: $nodeName" >> removed.txt &}
```

- **PRUNEDBINTERVAL = *interval***

Specifies the interval for the job database pruning in seconds. Pruning begins at the next update, and subsequent attempts to prune are done at each interval time. The default interval is 0, which means no pruning. Pruning reduces the size of the database by removing historical information.

- **PRUNEDBSAVETIME = *value***

Specifies the maximum value in seconds that data is saved in a table when the job database pruning is attempted. For example, if this value is set to 3600 and pruning is attempted, any data older than 3600 seconds in the table is pruned. The default value is 0, which means no pruning is attempted. Pruning reduces the size of the database by removing historical information.

You can specify the values separately for each table. The default value applies to any tables without explicit values. Syntax for wildcard (\*) and multiple specifications are supported as shown in these examples.

Specify that all tables in the application save up to 3600 seconds of data.

```
PRUNEDBSAVETIME = {*} 3600}
```

Specify that the “finished jobs” table save up to 2400 seconds of data, and the remaining tables save up to 3600 seconds of data.

```
PRUNEDBSAVETIME = {* 3600} \
{finished jobs 2400}
```

- **PRUNEDBSTARTTIME = *time***

Specifies when the job database pruning occurs. Pruning is attempted at the specified time. If specified with PRUNEDBINTERVAL, the PRUNEDBINTERVAL setting is ignored, and a warning is issued to the transcript. Pruning reduces the size of the database by removing historical information.

The following formats are supported:

- For daily pruning, specify time as *hour:minute*. For example, to prune the database at 1:00 a.m. each day, specify “01:00”.
- For weekly pruning, specify time as *day-of-week-hour:minute*. For example, to prune the database on Monday at 8:00 a.m. each week, specify “Monday-08:00”.
- For monthly pruning, specify time as *week-of-the-month-day-of-week-hour:minute*. For example, to prune the database at 1:00 a.m. on the first Sunday of each month, specify “1st-Sunday-01:00”. To prune on the last Sunday of each month at 1:00 a.m., specify “4th-Sunday-01:00”.

- **QUEUEDIR = *queue\_dir***

Specifies the path for the job queue directory. The default is the current working directory.

- **QUOTALEVELLIMIT = *num***

Specifies the level to apply the quota. By default, the lower level quota is ignored and the jobs are launched by FIFO. Refer to “[CalCM Resource Management](#)” on page 35 for more information.

- **QUOTAOVERDRAFT = *num***

Specifies the percentage value for the quota overdraft. The actual number is the value times the quota value of the group. Jobs belonging to the same quota group can grow up to the quota value plus the overdraft. This argument can be used to limit the number of jobs launched. Refer to “[CalCM Resource Management](#)” on page 35 for more information.

- **QUOTAOVERDRAFTLEVEL = *range***

Specifies the range where the quota overdraft level can be applied. By default, there is no restriction on the quota overdraft between quota groups. The range is specified as *[start-]end*. The quota overdraft is applied from the start level to the end level. The quota level below the start value is limited to the quota, but no overdraft happens. If the start value is not specified, it defaults to 1. Refer to “[CalCM Resource Management](#)” on page 35 for more information.

- **QUOTAPOLICY = *quota\_policy\_list***

Specifies the policy for each quota. Accepts a list in the form of “*quota:policy*” separated by ‘,’. If quota “\*” is used, all quotas are affected. Either FIFO or FAIRSHARE can be passed to the policy. When FAIRSHARE is specified, the sub quota launches jobs based on the quota and the summation of the maximum number of attributes for all jobs launched by the user. The default is “\*:FIFO”.

**QUOTAPOLICY = DEFAULT:FAIRSHARE**

In this example, quota group D1/P1 and all of the D2 group use FAIRSHARE, and the rest of the quota uses FIFO policy:

**QUOTAPOLICY = D1/P1:FAIRSHARE,D2:FAIRSHARE**

- **SCANJOBS = *flag***

Specifies the job scan mode. If specified, job scanning is performed while attaching the application. Valid modes include:

- 0 — No scanning is performed (default).
- 1 — Pings the jobs recorded in the job database.
- 2 — Scans the ports on the primary hosts specified in the cluster configuration file.
- 3 — Performs modes 1 and 2.

- **SCANJOBINFO = *flag***

Specifies whether to scan the job transcript to gather job information and record it in the info\_jobs table.

- 0 — No scanning of the job transcript is performed (default).
- 1 — Scanning and recording of the job transcript is performed.

- **STARTGUID = *num***

Specifies the start job ID. In LSF, the job ID starts from 100000 to maintain the number of digits for jobs. The default is 1.

- **VERSIONFILTERS = *filters***

Specifies a Calibre version that is not supported in CalCM. The job with the version and mode that matches the filter is modified to not use the specified job mode. To avoid the job mode from being changed, you can specify “JOB FLAG 1024” in the *job.conf* file. If no mode is specified, the job is not launched. You can specify “JOB NOTIFICATION” in the *job.conf* file to include the filter with the job status notification. The syntax is the same as that used for calcめ\_notification\_app.tcl VERSIONFILTERS. After the filter is matched, no further version checking is done.

## Description

The calcめ\_jobqueue\_app.tcl application creates a job dispatcher for the jobs located in a given directory. It also creates the necessary configuration files to run each job, sets environment parameters, receives job announcements from CalCM, and reads the network configuration.

When the calcm\_jobqueue\_app.tcl application launches a job, the potential resource usage of the active jobs is taken into account to determine whether the job to be launched exceeds the cluster or quota remote limits. The potential resource usage of the active jobs is also used in determining the priority order of different quota groups. Any suspended jobs are removed from the calculation of cluster and quota remote maximum values.

### Messaging Commands

You can use the following messaging commands with [calcm\\_send\\_message](#) to change the priority for a job, or adjust or update the cluster:

- [adjust\\_cluster](#)
- [check\\_cluster\\_limit](#)
- [prioritize\\_job](#)
- [close\\_node](#)
- [open\\_node](#)
- [read\\_conf](#)
- [register\\_node](#)

For a full list of valid messaging commands, see “[Messaging Commands](#).”

### Examples

```
APPLICATION TCL calcm_jobqueue_app.tcl FILE [
    QUEUEDIR = calcm/jobqueue
    CLUSTERCONF = cluster.conf
    CLUSTERCOUNT = 1
    CLUSTERMIN = 1
    CLUSTERMAX = 100
]
```

## **calcm\_notification\_app.tcl**

Monitors a calcmd log file and cluster status to send notifications to the job administrator if necessary.

### **Usage**

```
APPLICATION TCL path/calcm_notification_app.tcl [ FILE '['  
    [NOTIFYCMD = cmd]  
    [ATTRTRACKERS = tracker_list]  
    [CHECKINTERVAL = sec]  
    [DBPATH = file_path]  
    [DAEMONFILTERS = proc]...  
    [EVENTFILTERS = regexp]...  
    [EVENTTRACKERS = tracker_list]  
    [JOBEVENTFILTERS = regexp]...  
    [JOBEVENTTRACKERS = tracker_list]  
    [JOBFILTERS = proc ...]  
    [JOBTIMEOUT = sec]  
    [KILLONTIMEOUT = flag]  
    [MASTERFILTERS = proc]...  
    [NOTIFYCONF = file_path]  
    [NOTIFYEMAIL = email_list]  
    [NOTIFYINTERVAL = sec]  
    [NOTIFYSHUTDOWN = flag]  
    [PRUNEDBINTERVAL = interval]  
    [PRUNEDBSAVETIME = value]  
    [PRUNEDBSTARTTIME = time]  
    [RECORDMESSAGE = value]  
    [REMOTEFILTERS = proc]...  
    [TIMEOUT = sec]  
    [VERSIONFILTERS = filters]  
'' ]]
```

## Arguments

- **NOTIFYCMD = *cmd***

Specifies the command that is used to send notifications based on event, job, and cluster status. The NOTIFYCMD is used with calcm\_notification\_app.tcl arguments to report notifications to specified users. The default notification is to send an e-mail. An example of this command using the Linux mail command is as follows:

```
NOTIFYCMD = {mail -s $subject $users << $body &}
```

where *cmd* is enclosed in braces ({ }), both \$subject and \$body are set by the calcmd, and \$users includes \$user in NOTIFYEMAIL. Multiple commands can be split with the backslash (\) as shown:

```
NOTIFYCMD = {mail -s $subject $users << $body &} \
{echo $body >> notification.txt &}
```

- **ATTRTRACKERS = *tracker\_list***

Specifies a list of attribute trackers used to gather specific information for notifications about the job and cluster status. The list is enclosed in braces ({ }) as shown:

```
{“name” limit value exp}
```

where the attribute tracker arguments are defined as follows:

*name* — The attribute tracker name enclosed in quotes (“ ”). To add the attribute tracker for the host where calcmd is launched, you must include the keyword “daemon” in “*name*”.

*limit* — The number of results to keep.

*value* — The actual stored value that can be specified by a Nemo API or a predefined value. Refer to the “[CalCM Nemo API Reference Dictionary](#)” for more information.

*exp* — The evaluating expression used to obtain the result. Valid predefined variables include:

- *value* — The retrieved value from *value*.
- *dvalue* — The integer value from *value*.
- *pvalue* — The value from the previous cycle *value*.
- *pdvalue* — The integer value from the previous cycle *value*.

- **CHECKINTERVAL = *sec***

Specifies a value in seconds for how frequently the calcmd log is monitored. It is recommended to keep this value the same as the **SERVER INTERVAL** value in the CalCM configuration file (*calcmd.conf*). The default value is 60 seconds.

- **DBPATH = *file\_path***

Specifies the path to the notification database. You can define the file permissions for the database file by appending the permissions to the filename (for example, *calcnotification.db*,775). The default is *calcnotification.db.name*.

- **DAEMONFILTERS = *proc ...***

Specifies Tcl procedures to filter CalCM daemon host status information. If there is a match, the daemon status or attribute tracker information is reported to the user(s) specified in NOTIFYCMD. Note that the status or attribute tracker information is available only if you use an application to monitor and collect the hardware information for the daemon host. The variables and Tcl procedures used for DAEMONFILTERS are the same as those used for MASTERFILTERS argument and **REMOTE RESOURCE** command. The default is no specification.

- **EVENTFILTERS = *regexp ...***

Specifies Tcl regular expressions for evalEvent, evalFilter, and evalFilterExpr Tcl procedures for the events to match against each line in the CalCM daemon log file (*calcmd.log*). If there is a match, then the event is reported to the user(s) specified in NOTIFYCMD. Use braces ({ }) to enclose the filter for proper handling of the backslash (\) character and other special characters.

Tcl procedures starting with “eval1\*”, specified as “eval1Event”, “eval1Filter”, and “eval1FilterExpr”, can be used to report a notification only once. After the first report, the notification is removed from the filter and no longer monitored.

The evalEvent Tcl procedure can be used with EVENTFILTERS to check the frequency or number of occurrences of the event tracker. If specified, an action is evaluated. The format of the evalEvent argument is as follows:

*evalEvent “name” limit message [action]*

where the event tracker arguments are defined as follows:

*name* — Represents the event tracker name enclosed in quotes (“ ”).

*limit* — Specifies that if the number of occurrences of the event tracker is more than the limit, the message is reported, and the counter is reset. The value of the limit is specified as a positive integer.

*message* — Specifies the message for the event that is reported.

*action* — Specifies the action for the event that is reported.

The evalFilter Tcl procedure can be used with EVENTFILTERS to filter the event and to notify of the occurrence of the event. If specified, an action is evaluated. The format of the evalFilter argument is as follows:

*evalFilter filter message [action]*

where the event tracker arguments are defined as follows:

*filter* — Specifies the filter match for the event that is reported.

*message* — Specifies the message for the event that is reported.

*action* — Specifies the action for the event that is reported.

The evalFilterExpr Tcl procedure can be used with EVENTFILTERS to act based on the filter match and the condition.

**evalFilterExpr** *filter condition message [action]*

where the event tracker arguments are defined as follows:

*filter* — Specifies the filter match for the event that is reported.

*condition* — Specifies the condition to be met for the event that is reported.

*message* — Specifies the message for the event that is reported.

*action* — Specifies the action for the event that is reported.

An example of EVENTFILTERS with the evalFilter and evalEvent Tcl procedures is shown:

```
EVENTFILTERS = {evalFilter {^/*\s*Your timebomb license will
expire}\} \
{^\s*ERROR:\} \
{^WARNING: \(\LBD\)} \
{evalEvent "license_recover_failure" 10 {license recovery failed
more than $count times.}}
```

- **EVENTTRACKERS** = *tracker\_list*

Specifies the list of event trackers. The list is enclosed in braces ({} ) as shown:

{“*name*” *limit filter [condition]*}

where the event tracker arguments are defined as follows:

*name* — Represents the event tracker name enclosed in quotes (“ ”).

*limit* — Specifies how many evaluated results to keep. A value of -1 can be specified to keep the number of matched results indefinitely.

*filter* — Specifies the regular expression applied to the CalCM daemon log file (*calcmd.log*).

[*condition*] — Specifies a condition that determines whether the event tracker is triggered. If no condition is specified, the event tracker triggers when the filter argument is matched. The arguments in the condition are extracted as “*argn*” based on the filter and the number of arguments “*argc*”, the elapsed time “*elapsed*”, and the user name “*user*”.

An example of EVENTTRACKERS is shown:

```
EVENTTRACKERS = {"license_recover_failure" -1 {Unable to recover all
licenses}}
```

- **JOBEVENTFILTERS = *regexp***

Specifies Tcl regular expressions for evalEvent, evalFilter, or evalFilterExpr Tcl procedures for job events. If specified, the job's transcript is matched with the specified filters. If there is a match, the notification is recorded in the notification database and reported to the user(s) specified in NOTIFYCMD. The jobAttr can be used in the condition argument to check the job attributes specified in the *job.conf* file. Use braces ({} ) to enclose the filter for proper handling of the backslash (\) character and other special characters.

An example of JOBEVENTFILTERS is shown:

```
JOBEVENTFILTERS = {^/*\s*Your timebomb license will expire}\|
{^\\s*ERROR:} \
{evalFilterExpr {^CPU TIME = (\d+) [+ ]*(\d*) .* REAL TIME = (\d+)
.* OPS COMPLETE = (\d+)} {$argc > 3 && ($arg3 > ($arg1 + $arg2)) }
{OP $arg4 with REAL TIME > CPU TIME.}}
```

- **JOBEVENTTRACKERS = *tracker\_list***

Specifies the list of job event trackers as specified in EVENTTRACKERS. The job event tracker is applied to the job's transcript. The jobAttr can be used in the condition argument to check the job attributes specified in the *job.conf* file.

An example of JOBEVENTTRACKERS is shown:

```
JOBEVENTTRACKERS = {"no_remote_cpu" -1 {^CPU TIME = (\d+) [+ ]*(\d*)
.* REAL TIME = (\d+).* OPS COMPLETE} {$argc == 3 && $arg2 == 0}}
```

- **JOBFILTERS = *proc ...***

Specifies Tcl procedures to filter job information. The Tcl procedures evalExpr and evalAttr can be used in conjunction with the results gathered from a JOBFILTERS attribute tracker list. If there is a match, the notification is reported to the user(s) specified in NOTIFYCMD. If specified, an action is evaluated. You can use the [nemo:job](#) API with \$jobObject and the [nemo:master](#) API with \$masterObj to get specific information about a Calibre job and primary process. Refer to the “[CalCM Nemo API Reference Dictionary](#)” for more information. The default is no specification.

Tcl procedures starting with “eval1\*”, specified as “eval1Expr” and “eval1Attr”, can be used to report a notification only once. After the first report, the notification is removed from the filter and no longer monitored.

The JOBFILTERS valid variables include:

- jobObj — Job object.
- jobID — The job ID of the job.
- masterObj — The primary host object of the job.
- hdb — The HDB number for each primary.
- nodeObj — The primary node object of the job.

- nodeName — The primary node name.
- user — The user who launched the job.
- updated — The boolean flag indicating whether the job is updated properly.
- demand — The job's total CPU demand.
- adjusted\_demand — The job's adjusted CPU demand.
- cpu\_adjustable — The boolean flag indicating whether the job CPU supply is adjustable.
- supply — The number of remotes supplied to the job.
- revoke — The number of remotes revoked from the job.

The evalExpr Tcl procedure argument format is as follows:

**evalExpr *expr message [action]***

If the evalExpr *expr* is evaluated true, the message is reported and if specified, the action is evaluated. The arguments for the evalExpr are as follows:

- *expr* — Specifies the expression to evaluate.
- *message* — Specifies the notification message if the expression is true.
- *action* — Specifies the action that is evaluated if the expression is true. The JOBFILTERS variables can be used in the *action* argument.

The evalAttr Tcl procedure argument format is as follows:

**evalAttr “*name*” *condition message [action]***

If the evalAttr “*name*” is evaluated true for a specified condition, the message is reported. After the notification, the attribute tracker information is reset. The arguments for the evalAttr are as follows:

- *name* — The value or result to be evaluated enclosed in quotes (“ ”).
- *condition* — The condition for evaluation.

The following variables can be used in the condition argument in addition to the JOBFILTERS variables:

- count — The number of positive results.
- total — The total number of positive results that occur and not reset.
- ncount — The number of non-positive results.
- consecutive — The number of consecutive positive results.
- nconsecutive — The number of consecutive non-positive results.
- limit — The number of results tracked.

- values — The list of values gathered from the attribute tracker list.
- value — The last attribute tracker list value.
- dvalues — The list of integer values gathered from the attribute tracker list.
- dvalue — The last integer value in the attribute tracker list.
- results — The list of results evaluated.
- result — The last result evaluated.
- *message* — The notification message if the expression is true. The variables can be used from the *condition* argument.
- *action* — Specifies the action that is evaluated if the expression is true. The JOBFILTERS variables can be used in the *action* argument.
- **JOBTIMEOUT = *sec***  
Specifies the timeout interval in seconds until the job transcript is updated. If this timeout interval is exceeded, it is reported. The default value is 4800 seconds. This parameter is active only when JOBEVENTFILTERS is specified.
- **KILLONTIMEOUT = *flag***  
Specifies whether to kill the CalCM daemon when the timeout interval occurs. The default is false; do not kill. Valid values include:
  - 0 — CalCM daemon is not killed when timeout occurs (default).
  - 1 — CalCM daemon is killed when timeout occurs.
- **MASTERFILTERS = *proc ...***  
Specifies Tcl procedures to filter Calibre primary process information. The Tcl procedures evalExpr and evalAttr can be used in conjunction with the results gathered from a MASTERFILTERS attribute tracker list. If there is a match, the notification is reported to the user(s) specified in NOTIFYCMD. You can use the Nemo API with \$nodeObj to get specific information about a primary process running on a cluster node. Refer to the “[CalCM Nemo API Reference Dictionary](#)” for more information. The variables used for MASTERFILTERS are the same as the variables used for **REMOTE RESOURCE**. The default is no specification. The example is as follows:

```
MASTERFILTERS = {$down} \
    {evalExpr {$swap >= 100} {The swap space is more than 100 mb.}}
```
- **NOTIFYCONF = *file\_path***  
Specifies the path of a file that contains the notification attribute trackers and filter configurations. The following configuration variables are used:
  - ATTRTRACKERS
  - DAEMONFILTERS

- EVENTFILTERS
- EVENTTRACKERS
- JOBEVENTFILTERS
- JOBEVENTTRACKERS
- JOBFILTERS
- MASTERFILTERS
- REMOTEFILTERS
- VERSIONFILTERS

---

**Note**

---

 By specifying notifications outside of the main CalCM configuration file (*calcmd.conf*), as with NOTIFYCONF and [reset\\_notification](#), you can update the notification settings without restarting the CalCM daemon.

---

- NOTIFYEMAIL = *email\_list*  
Specifies a list of comma-separated e-mail addresses that receive the notification. The \$user variable is substituted with the username that launched the Calibre job, and the \$defaultUserName variable is substituted with the username that launched the calcmd. The default value is \$user.
- NOTIFYINTERVAL = *sec*  
Specifies the interval for the notification in seconds. It can be used to reduce the number of notification e-mails by accumulating the notifications for the interval. The default value is 1800 seconds.
- NOTIFYSHUTDOWN = *flag*  
Specifies whether to notify if a shutdown event of the CalCM daemon occurs. If the flag is set to true, the event is triggered when detaching an application from the CalCM daemon. The default is false; do not notify. Valid values include:
  - 0 — Do not notify when a CalCM daemon shutdown event occurs (default).
  - 1 — Notify when a CalCM daemon shutdown event occurs.
- PRUNEDBINTERVAL = *interval*  
Specifies the interval for the notification database pruning in seconds. Pruning begins at the next update, and subsequent attempts to prune are done at each interval time. The default interval is 0, which means no pruning. Pruning reduces the size of the database by removing historical information.
- PRUNEDBSAVETIME = *value*  
Specifies the maximum value in seconds that data is saved in a table when the notification database pruning is attempted. For example, if this value is set to 3600 and pruning is

attempted, any data older than 3600 seconds in the table is pruned. The default value is 0, which means no pruning is attempted. Pruning reduces the size of the database by removing historical information.

You can specify the values separately for each table. The default value applies to any tables without explicit values. Syntax for wildcard (\*) and multiple specifications are supported as shown in these examples.

Specify that all tables in the application save up to 3600 seconds of data.

```
PRUNEDBSAVETIME = {* 3600}
```

Specify that the “notifications” table save up to 2400 seconds of data, and the remaining tables save up to 3600 seconds of data.

```
PRUNEDBSAVETIME = {* 3600} \
{notifications 2400}
```

- **PRUNEDBSTARTTIME = *time***

Specifies when the notification database pruning occurs. Pruning is attempted at the specified time. If specified with PRUNEDBINTERVAL, the PRUNEDBINTERVAL setting is ignored, and a warning is issued to the transcript. Pruning reduces the size of the database by removing historical information.

The following formats are supported:

- For daily pruning, specify time as *hour:minute*. For example, to prune the database at 1:00 a.m. each day, specify “01:00”.
- For weekly pruning, specify time as *day-of-week-hour:minute*. For example, to prune the database on Monday at 8:00 a.m. each week, specify “Monday-08:00”.
- For monthly pruning, specify time as *week-of-the-month-day-of-week-hour:minute*. For example, to prune the database at 1:00 a.m. on the first Sunday of each month, specify “1st-Sunday-01:00”. To prune on the last Sunday of each month at 1:00 a.m., specify “4th-Sunday-01:00”.

- **RECORDMESSAGE = *value***

Specifies whether or not to record messages in the notification database. The default value is 1, which specifies to record messages in the notifications database. Message recording is disabled by specifying a value of 0.

- **REMOTEFILTERS = *proc ...***

Specifies Tcl procedures to filter Calibre remote process information. If there is a match, the remote status or attribute tracker information is reported to the user(s) specified in NOTIFYCMD. The variables and Tcl procedures used for REMOTEFILTERS are the same as those used for MASTERFILTERS and **REMOTE RESOURCE**. The default is no specification.

- **TIMEOUT = *sec***

Specifies the timeout interval in seconds until all the CalCM daemon update cycle is finished. If this timeout interval is exceeded, it is reported. The default value is 2400 seconds.

**Note**

 The CalCM daemon update cycle can slow down as a function of its required workload. Update cycle times exceeding 10 minutes are considered excessive and may indicate an issue that should be reported to the CalCM administrator.

---

- **VERSIONFILTERS = *filters***

Specifies a Calibre version that is not supported in CalCM. A notification is reported for the job with the version that matches the filter. If the optional job mode parameter is specified, it is reported in the notification only if the version and the mode(s) match the filter. After the filter is matched, no further version checking is done.

The allowed modes are as follows:

```
DCA: Dynamic CPU allocation (supply/revoke of remotes)
HT: Hyperthread
HS: Hyperscale
```

Filters are specified one per line. For multiple filters, the backslash (\) is used. The example is as follows:

```
VERSIONFILTERS = {<2010.2} \
{2016.2_1.<100 HT HS DCA} \
{CALIBREWB <2016.2}
```

where version filters are specified as follows:

- A filter is composed of a sub-version that is separated with “.” or “\_” and the mode.
- A comparison operator can be specified before the sub-version. The default comparison value is “==” (match).
- An optional executable name can be specified before the version filter to apply the filter for a specific binary such as “calibrewb”. The strings “CALIBRE” and “CALIBREWB” can be specified in the executable name. Notification occurs only if the version and the job mode(s) are matched.

See “[Example 5](#)” on page 193 for expanded details.

## Description

The calcnotification\_app.tcl application checks the calcmd log file in the background and reports warnings or errors on every update cycle. Filters for the job, primary, and remote processes are also evaluated during the application update. If the attribute tracker and filter conditions are matched, the notification is reported at the end of the update cycle. The option

KILLONTIMEOUT can be used to kill the CalCM daemon if necessary. The default filter for the calcnotification\_app.tcl application is as follows:

```
{^/*\s*Your timebomb license will expire}  
{^\s*ERROR:  
{^WARNING: \ (LBD\)} }
```

The calcnotification\_app.tcl application also supports a tag syntax for grouping notifications. The tags can be used to show filtered messages on the jobs detail web page of the CalCM dashboard. If a notification filter has no tag, the default tag is “notifications”. See “[Example 7](#)” on page 194 for expanded details.

## Examples

### Example 1

This is an example of calcnotification\_app.tcl usage that sends an e-mail whenever a log file line is detected that starts with “ERROR:”, has the phrase “Your timebomb will expire”, or if the JOB UPDATE to next JOB UPDATE interval exceeds 600 seconds. The CalCM daemon is not killed upon timeout.

```
APPLICATION TCL $CALCM_APP/calcnotification_app.tcl FILE [  
LOG = $CALCM_HOME/notification.log  
NOTIFYCMD = {mail -s $subject $user << $body &}  
NOTIFYEMAIL = yourEmail@yourDomain  
EVENTFILTERS = {(^*\$ERROR:) | (Your timebomb will expire)}  
TIMEOUT = 600  
KILLONTIMEOUT = 0  
]
```

### Example 2

This is an example of the calcnotification\_app.tcl application using ATTRTRACKERS and JOBFILTERS to report low CPU demand using the Tcl evalAttr procedure. If the CPU demand is less than or equal to zero for nine consecutive update cycles, it is reported.

```
ATTRTRACKERS = {"demand,$jobID" 10 $demand {$dvalue <= 0}}  
JOBFILTERS = {evalAttr "demand,$jobID"  
{$consecutive >= $limit-1}  
{The demand was less than 0 for $limit update cycles.}}
```

### Example 3

This is an example of the calcnotification\_app.tcl application using JOBFILTERS with the nemo::job API to retrieve the number of Calibre primary processes and report low CPU demand using the Tcl evalAttr procedure.

```
JOBFILTERS = {evalExpr {[nemo::job number_of_masters $jobObj] == 0} {The  
number of HDB is 0.}} \  
{evalAttr "demand,$jobID" {$consecutive >= $limit-1} {The demand was less  
than 0 for $limit update cycles ($attrValues($name)).}}
```

#### **Example 4**

This is an example of the calcnotification\_app.tcl application using ATTRTRACKERS and REMOTEFILTERS to report memory swap space with the Tcl evalAttr procedure. If the swap space is greater than or equal to 100 MB for ten consecutive cycles, it is reported. This attribute tracker and filter can also be specified for MASTERFILTERS.

```
ATTRTRACKERS = { "swap,$nodeName" 10 $swap {$dvalue <= 100} }
REMOTEFILTERS = {evalAttr {$swap >= 100}
{$consecutive >= $limit-1}
{The swap space is more than 100 mb for $limit update cycles.}}
```

#### **Example 5**

This is an example of the calcnotification\_app.tcl application using VERSIONFILTERS to filter and report notifications for Calibre versions and modes that are not supported in CalCM. This syntax can also be used for calcnotification\_jobqueue\_app.tcl VERSIONFILTERS.

```
VERSIONFILTERS = {<2010.2} \
{2016.2_1.<100 HT HS DCA} \
{CALIBREW <2016.2}
```

Filters are applied as follows:

- The version “2010.1\_23” is matched to the first filter “<2010.2” because “2010.1\_23” is less than “2010.2”.
- The version “2016.2\_1.99” is matched in the second filter because “2016.2\_1” is equal and “99 < 100”. The job is launched in HYPER MTFLEX mode and is matched to HS and DCA.
- The version of “CALIBREW” that is less than “2016.2” is matched to the third filter.

#### **Example 6**

This is an example of the calcnotification\_app.tcl application using REMOTEFILTERS to launch a custom script when a remote host is detected to have low memory and high swap usage. Specify the expression on a single line.

```
REMOTEFILTERS = {evalExpr {$mem_free < 20000 && $swap > 10000}
{Remote free memory < 20 GB, Swap > 10 GB,
 launching analyze_remote_memory_usage}
{exec /usr/bin/timeout 60s ssh -oStrictHostKeyChecking=no
 -oBatchMode=yes $hostname <FullPath>/analyze_remote_memory_usage >>
 <FullPath>/analyze_remote_memory_usage.log &}}
```

This is the expected result in the CalCM daemon log:

```
2019-01-02 09:45:48] NOTE: update_application (Wed Jan 2 09:45:48 2019),  
Notification Service v1.1_1.61.2.3  
[2019-01-02 09:45:48] Notification action: exec /usr/bin/timeout 60s  
ssh -oStrictHostKeyChecking=no  
-oBatchMode=yes $hostname <FullPath>/analyze_remote_memory_usage >>  
<FullPath>/analyze_remote_memory_usage.log &  
[2019-01-02 09:45:48] Notification user(mylogin), remote(remotehost):  
Remote free memory < 20 GB, Swap > 10 GB,  
launching analyze_remote_memory_usage
```

Considerations when launching a custom script for this application are as follows:

- The custom script is launched inside the CalCM daemon cycle loop and can severely slow down the CalCM cycle when swapping notification is raised on multiple remote hosts at the same time, for example, and can even cause a timeout.

It is therefore strongly recommended to launch custom scripts that are not long running by design. If the script needs to connect by SSH protocol to a swapping host, for example, you should protect against long executions by specifying the following:

```
{exec /usr/bin/timeout 60s ssh $hostname ... &}
```

The ampersand (&) background launch ensures that a slow execution of the custom script does not slow down the CalCM daemon itself.

The Linux timeout command ensures that background processes cannot accumulate indefinitely on the CalCM host if they are unresponsive.

- The custom script can raise an alert to call for immediate action or be used as a post-run analysis tool to identify a specific process that resulted in a degraded runtime of CalCM jobs using this remote host.

### Example 7

This is an example of the calcm\_notification\_app.tcl application using tag syntax for grouping notifications. You can insert a specific tag into the *message* argument. The tag syntax is “tag(*tag\_name*)” with only one tag specification per filter. The *tag\_name* can consist of only alphabetic characters (a-z, A-Z), numbers (0-9), and the underscore (\_). This syntax is supported in the evalAttr, evalExpr, evalEvent, and evalFilter Tcl procedures. The example is as follows:

```
JOBFILTERS = {evalAttr  
"long_ops,$jobID,$hdb" {$consecutive >= $limit-1}  
{tag(long_ops): $JobID : The present operation is ongoing  
for 7 hours with under 10 cores.} } \  
{evalAttr "not_updated,$jobID" {$consecutive >= $limit-1}{tag(update):  
The job has not responded for $limit update cycles.}}
```

## calcm\_rmanager\_app.tcl

Shares cluster resources between jobs.

### Usage

```
APPLICATION TCL path/calcm_rmanager_app.tcl [ FILE '['  
    [ACCOUNTMISMATCHLIMIT = num]  
    [ADJUSTMINMAXOP = flag]  
    [ADJUSTRAMPUPDOWN = flag]  
    [DBPATH = file_path]  
    [MAXCHANGE = num]  
    [MINMAXOP = op_minmax_list]  
    [QUOTADB = path]  
    [QUOTAOVERDRAFT = num]  
    [QUOTAOVERDRAFTLEVEL = num]  
    [QUOTAOVERDRAFTPRIORITY = num]  
    [RAMPUPDOWN = list]  
    [RAPIDRAMPUP = values]  
    [RAPIDRAMPUPRANGE = values]  
    [REVOKEDEMANDLIMIT = num]  
    [REVOKEUNREGISTERED = flag]  
    [SUSPENDMIN = num]  
' ] ]
```

### Arguments

- ACCOUNTMISMATCHLIMIT = *num*  
Specifies to check for a mismatch between the allocated and connected remotes on a remote node. If the number of connected remotes is reported as 0, for the specified number of cycles, the remotes are not considered in calculating the CPU demand. The default is three consecutive daemon update cycles. The pending remotes on the node are already not included in the CPU demand calculation.
- ADJUSTMINMAXOP = *flag*  
Specifies whether to enable the adjustment of the number of minimum and maximum remote hosts based on the job operation. If enabled, this argument can be used in

conjunction with the MINMAXOP argument to specify a list of job operations with minimum and maximum remote host numbers. Valid values include:

- 0 — Disables the adjustment of the minimum and maximum number of remote hosts based on the job operation (default).
- 1 — Enables the adjustment of the minimum and maximum number of remote hosts based on the job operation and applies a default value of DRC=\*/256 for all jobs. Specifying the (\*) character keeps the existing number of remote hosts. The format is as follows:

*job\_operation\_class=min/max*

where

*min/max* specifies the minimum and maximum number of remote hosts, respectively.

- 2 — Overrides the *min* value settings if the minimum MINMAXOP value is less than the actual number of cores in the primary host. For example, with the following settings specified:

```
ADJUSTMINMAXOP = 2
MINMAXOP = DRC:*=24/256, LITHO:*=24/1024
```

If the DRC:/\* MINMAXOP minimum value is 24, and the actual number of cores in the primary host is 32, the settings for the minimum MINMAXOP value are updated to 32 as follows:

```
ADJUSTMINMAXOP = 2
MINMAXOP = DRC:*=32/256, LITHO:*=32/1024
```

- **ADJUSTRAMPUPDOWN = *flag***

Specifies whether to enable the feature to adjust the ramp up/down ratio that is multiplied in calculating the job CPU demand. If the flag is set to 1, then the ramp up/down feature is enabled and can be used in conjunction with the RAMPUPDOWN argument to specify a list of job operation class and ramp up/down ratio values. The default is 0 (disabled).

- **DBPATH = *file\_path***

Specifies the path to the resource manager database. The supply, revoke, and quota information is stored in this database. You can define the file permissions for the database file by appending the permissions to the filename (for example, *calcm\_rmanager.db,666*). The default is *calcm\_rmanager.db*.

- **MAXCHANGE = *num***

Specifies the maximum number of resources that can be added to a job. The limit is implemented by capping the demand for a job. The default is 2147483647.

- **MINMAXOP = *op\_minmax\_list***

Specifies a list of job operations with the minimum and maximum number of remote hosts to allocate for a specified job operation. The MINMAXOP values override the existing minimum and maximum number of remote hosts allocated for a job operation. For example:

```
MINMAXOP = DRC:*=256/256, LITHO:*=128/1024
```

where

- Job operation class supports type matching with wildcard (\*) and (?) expressions.
- Multiple list items are comma-separated.
- Numbers of minimum and maximum remote hosts are separated by a forward slash (/).

The default is no specification. Arguments for MINMAXOP are overridden by the [REMOTE MINMAX](#) statement in the job configuration file.

- **QUOTADB = *path***

Specifies the path to a quota configuration file. The hierarchical quota management capability is activated when this argument is specified. Refer to “[Quota Configuration File \(calcm\\_quota.conf\)](#)” on page 29 for more information.

- **QUOTAOVERDRAFT = *num***

Specifies the percentage value for the quota overdraft. The actual number is the value times the quota value of the group. Jobs belonging to the same quota group can grow up to the quota value plus the overdraft. This argument can be used to limit the number of jobs launched. Refer to “[CalCM Resource Management](#)” on page 35 for more information.

- **QUOTAOVERDRAFTLEVEL = *num***

Specifies the level where the quota overdraft is forced. By default, there is no restriction on the quota overdraft between quota groups. Refer to “[CalCM Resource Management](#)” on page 35 for more information.

- **QUOTAOVERDRAFTPRIORITY = *num***

Specifies the maximum job priority that can be affected by the quota overdraft. The job with higher priority can get more remotes beyond the quota. The default value is 0. This argument can be used to control whether the TAT job with dynamic priority can exceed the quota overdraft. Refer to “[CalCM Resource Management](#)” on page 35 for more information.

- **RAMPUPDOWN = *list***

Specifies a list containing the operation class and its ramp up/down ratio. The default value is null (no specification). The job’s CPU demand is adjusted if the job is in the specified operation class. The list values can be overridden with REMOTE RAMPUPDOWN rules in the *job.conf* file.

- **RAPIDRAMPUP = *values***

Specifies multiplier values to adjust the ramp up ratio. This argument is used along with the RAPIDRAMPUPRANGE argument to apply different multipliers to the CPU demand based on the utilization. The values are listed in corresponding order for these arguments.

RAPIDRAMPUP values are multiplier values greater than 1 resulting in faster ramp up of CPU demanding jobs to fill the cluster. The default values are 4 (quadruple ramp-up speed) and 2 (double ramp-up speed). You can specify 0 to disable this feature.

- **RAPIDRAMPUPRANGE = *values***

Specifies different percentage limits to adjust the ramp up ratio. If the utilization percentage of remotes in the CalCM cluster is less than the specified limit, then the corresponding multiplier value in the RAPIDRAMPUP argument is applied. RAPIDRAMPUPRANGE values are positive percentages less than 100. The default values are 70 and 20. For example, with the following settings specified:

```
RAPIDRAMPUP = 4 2 1.5
RAPIDRAMPUPRANGE = 20 70 90
```

If the total number of remotes is 1000, and the number of allocated remotes is 800, then the utilization percentage is 80. This is more than the percentage limit of 70 and less than the percentage limit of 90, so the corresponding multiplier value of 1.5 is used in calculating the CPU demand. When more than 90% of the remotes managed by CalCM are used, the normal CPU demand value is used to compute the CPU additions to each job.

- **REVOKEDEMANDLIMIT = *num***

Specifies a negative limit value for CPU demand. A running job with a CPU demand less than a negative limit value of “256” (default) can be revoked regardless of the job priority.

- **REVOKEUNREGISTERED = *flag***

Specifies whether to revoke an unregistered remote. Valid values include:

0 — Unregistered remotes are not revoked (default).

1 — Unregistered remotes are revoked. This occurs at every update cycle.

- **SUSPENDMIN = *num***

Specifies a minimum number of remotes allocated in the cluster string for the case of a running (active) job that has been suspended. This argument overrides the default *min* remotes number allocated for a job that is suspended with the [suspend\\_job](#) command.

## Description

The resource manager application, *calcm\_rmanager\_app.tcl*, monitors the CPU status of the current jobs and, where necessary, dynamically adjusts the CPU demand by jobs. The resource manager allocates cluster nodes based on the CPU demand of running a job, the job priorities, and the available hardware resources.

The hierarchical quota specified with the [JOB QUOTA](#) statement in the job configuration file is passed to *calcm\_rmanager\_app.tcl* and the proportional share is calculated based on that quota.

The hierarchical quota management capability is activated by defining the QUOTADB argument to point to a quota configuration file. For example:

```
QUOTADB = $CALCM_HOME/conf/calcm_quota.conf
```

## Messaging Commands

You can use the following messaging commands with [calcm\\_send\\_message](#) to adjust the quota or cluster resources:

- [adjust\\_cluster](#)
- [adjust\\_minmax](#)
- [adjust\\_quota](#)
- [reset\\_quota](#)

## Examples

```
APPLICATION TCL calcm_rmanager_app.tcl FILE [
    QUOTADB = $CALCM_HOME/conf/calcm_quota.conf
]
```

## **calcm\_rmonitor\_app.tcl**

Collects CPU and license usage for each job.

### **Usage**

```
APPLICATION TCL path/calcm_rmonitor_app.tcl [ FILE '['  
    [DBPATH = file_path]  
    [MAXSAMPLINGINTERVAL = seconds]  
    [PRUNEDBINTERVAL = interval]  
    [PRUNEDBSAVETIME = value]  
    [PRUNEDBSTARTTIME = time]  
'']
```

### **Arguments**

- **DBPATH = *file\_path***  
Specifies the path to the resource monitor database. You can define the file permissions for the database file by appending the permissions to the filename (for example, *calcm\_rmonitor.db,666*). The default is *calcm\_rmonitor.db*.
- **MAXSAMPLINGINTERVAL = *seconds***  
Specifies the maximum time between sampling data. The default is 1200 seconds.
- **PRUNEDBINTERVAL = *interval***  
Specifies the interval for the resource monitor database pruning in seconds. Pruning begins at the next update, and subsequent attempts to prune are done at each interval time. The default interval is 0, which means no pruning. Pruning reduces the size of the database by removing historical information.
- **PRUNEDBSAVETIME = *value***  
Specifies the maximum value in seconds that data is saved in a table when the resource monitor database pruning is attempted. For example, if this value is set to 3600 and pruning is attempted, any data older than 3600 seconds in the table is pruned. The default value is 0, which means no pruning is attempted. Pruning reduces the size of the database by removing historical information.

You can specify the values separately for each table. The default value applies to any tables without explicit values. Syntax for wildcard (\*) and multiple specifications are supported as shown in these examples.

Specify that all tables in the application save up to 3600 seconds of data.

```
PRUNEDBSAVETIME = { * 3600 }
```

Specify that the “rmonitor license” table saves up to 2400 seconds of data, and the remaining tables save up to 3600 seconds of data. The data in the compression tables is pruned according to the configuration value of the parent table.

```
PRUNEDBSAVETIME = {* 3600} \
    {rmonitor license 2400}
```

- PRUNEDBSTARTTIME = *time*

Specifies when the resource monitor database pruning occurs. Pruning is attempted at the specified time. If specified with PRUNEDBINTERVAL, the PRUNEDBINTERVAL setting is ignored, and a warning is issued to the transcript. Pruning reduces the size of the database by removing historical information.

The following formats are supported:

- For daily pruning, specify time as *hour:minute*. For example, to prune the database at 1:00 a.m. each day, specify “01:00”.
- For weekly pruning, specify time as *day-of-week-hour:minute*. For example, to prune the database on Monday at 8:00 a.m. each week, specify “Monday-08:00”.
- For monthly pruning, specify time as *week-of-the-month-day-of-week-hour:minute*. For example, to prune the database at 1:00 a.m. on the first Sunday of each month, specify “1st-Sunday-01:00”. To prune on the last Sunday of each month at 1:00 a.m., specify “4th-Sunday-01:00”.

## Description

The *calcm\_rmonitor\_app.tcl* application monitors the following information and reports this information using the CalCM dashboard web application:

- Resource utilization by department
- Resource utilization by cost center
- License utilization by department
- License utilization by cost center
- License utilization per job level
- Job duration

To enable the CalCM dashboard web application, see the [calcm\\_http\\_server\\_app.tcl](#) application. The server name on which the CalCM daemon is running and the dashboard server port specified in the CalCM daemon configuration file (*calcmd.conf*) are entered in a browser window.

```
http://server_name:server_port
```

The CalCM dashboard web application displays usage plots and data tables and provides options for filtering and interacting with the jobs and data.

## Examples

```
APPLICATION TCL calcm_rmonitor_app.tcl FILE [
    DBPATH = calcm_rmonitor.db
]
```

## **calcm\_systemanalysis\_app.tcl**

Creates a database of job and host information for statistical and graphical analysis.

### **Usage**

```
APPLICATION TCL path/calcm_systemanalysis_app.tcl [ FILE '['  
    [DBPATH = file_path]  
    [PRUNEDBINTERVAL = interval]  
    [PRUNEDBSAVETIME = value]  
    [PRUNEDBSTARTTIME = time]  
'' ]
```

### **Arguments**

- DBPATH = *file\_path*

Specifies the path to the database. The default is *calcm\_analysis.db*.

For backward compatibility, you can migrate the data from the job database to another version of the database by setting the CALCM\_MIGRATE\_DB environment variable to 1. This allows you to move to a different version of the database and not lose any data when the database schema changed. The supported changes in the schema include:

- Add column
- Remove column
- Change the data type of a column to “TEXT”

The database file with the old schema is renamed to “databaseFileName.db.backup”.

- PRUNEDBINTERVAL = *interval*

Specifies the interval for the database pruning in seconds. Pruning begins at the next update, and subsequent attempts to prune are done at each interval time. The default interval is 0, which means no pruning. Pruning reduces the size of the database by removing historical information.

- PRUNEDBSAVETIME = *value*

Specifies the maximum value in seconds that data is saved in a table when the database pruning is attempted. For example, if this value is set to 3600 and pruning is attempted, any data older than 3600 seconds in the table is pruned. The default value is 0, which means no pruning is attempted. Pruning reduces the size of the database by removing historical information.

You can specify the values separately for each table. The default value applies to any tables without explicit values. Syntax for wildcard (\*) and multiple specifications are supported as shown in these examples.

Specify that all tables in the application save up to 3600 seconds of data.

```
PRUNEDBSAVETIME = {* 3600}
```

Specify that the “nodes” table saves up to 2400 seconds of data, the “jobs remotes” table saves up to 1800 seconds of data, and the remaining tables save up to 3600 seconds of data.

```
PRUNEDBSAVETIME = {* 3600} \
{nodes 2400} \
{jobs remotes 1800}
```

- **PRUNEDBSTARTTIME** = *time*

Specifies when the database pruning occurs. Pruning is attempted at the specified time. If specified with PRUNEDBINTERVAL, the PRUNEDBINTERVAL setting is ignored, and a warning is issued to the transcript. Pruning reduces the size of the database by removing historical information.

The following formats are supported:

- For daily pruning, specify time as *hour:minute*. For example, to prune the database at 1:00 a.m. each day, specify “01:00”.
- For weekly pruning, specify time as *day-of-week-hour:minute*. For example, to prune the database on Monday at 8:00 a.m. each week, specify “Monday-08:00”.
- For monthly pruning, specify time as *week-of-the-month-day-of-week-hour:minute*. For example, to prune the database at 1:00 a.m. on the first Sunday of each month, specify “1st-Sunday-01:00”. To prune on the last Sunday of each month at 1:00 a.m., specify “4th-Sunday-01:00”.

## Description

The *calcsm\_systemanalysis\_app.tcl* application builds a database of job and host information gathered through regular snapshots. The data, which can be used for statistical and graphical analysis, is output to a database consisting of three main tables:

- Network — Contains the total number of available CPU cores and the total number of CPU cores utilized by job.
- Nodes — Contains the load average at 1, 5, and 15 minute intervals per host.
- Jobs — Contains the total CPU cores allocated, total CPU cores utilized, and CPU demand per job.

## Examples

```
APPLICATION TCL calcsm_systemanalysis_app.tcl FILE [
    DBPATH = calcsm_analysis.db
]
```

## calcm\_tat\_app.tcl

Controls the job turn-around time (or TAT) by sending a remote count request to the resource monitor application. Job budget time is specified in the job configuration file. The CalCM TAT application supports increasing the priority when the duration of a job is expected to exceed the budget.

### Usage

```
APPLICATION TCL path/calcm_tat_app.tcl [ FILE '['  
    [ADJUST_BUDGET = [true | false]]  
    [ALLOW_AOT = [true | false]]  
    [DB_QUERY_KEYS = [OPE | KEY | MODE | RULE | VERSION]]  
    [DBPATH = file_path]  
    [EXTRA_RATIO = ratio]  
    [MAX_REMOTE = value]  
    [MIN_REMOTE = value]  
    [NOTIFICATION_ESTATUS = [ 0 | 1 | 2 ]]  
    [NOTIFICATION_EXCEED_RATIO = percent]  
    [NOTIFICATION_FILE_PATH = file_path]  
    [NOTIFICATION_INTERVAL = minutes]  
    [PRIORITY_BUMP = value]  
    [PRIORITY_CHANGE_ECP = value]  
    [PRIORITY_CHANGE_ERT_RATIO = value]  
    [PRUNEDBINTERVAL = interval]  
    [PRUNEDBSAVETIME = value]  
    [PRUNEDBSTARTTIME = time]  
    [SCALE_OPS = operation]  
'']
```

### Arguments

- **ADJUST\_BUDGET = [true | false]**

Specifies a case-insensitive value that enables or disables the budget adjustment feature. When this argument is enabled and the job duration exceeds the accumulated budget, then the budget is adjusted. It is recommended that you run your jobs with this feature enabled (true) for optimal performance. Refer to “[Budget Management](#)” for more information on the budget adjustment feature. The default is true.

- **ALLOW\_AOT = [true | false]**  
Specifies whether the TAT application revokes remotes from a budgeted job as long as the CPU demand is positive. When set to “true”, this argument enables an “allow ahead of time” mode which increases the chance that a budgeted job will finish earlier than the budgeted time. The default is false (disabled).
- **DB\_QUERY\_KEYS = [OPE | KEY | MODE] RULE | VERSION]**  
Specifies which database fields to query in order to find a job in the database. Valid values, which must be separated by the “|” character, include:
  - OPE — Operation name(s).
  - KEY — User-specified job key. See [JOB KEY](#) for more information.
  - MODE — Execution mode.
  - RULE — Rule file name.
  - VERSION — Calibre version.The default value is “OPE|KEY|MODE”, which specifies that the TAT application should query the database to find a job based on the operation name(s), user-specified job key, and execution mode. You typically use RULE and VERSION only in situations where these values do not change.
- **DBPATH = *file\_path***  
Specifies a path or filename for the TAT database file. An invalid path generates an open database file error. The default location is `$CALCM_HOME/calcm_tat.db`.
- **EXTRA\_RATIO = *ratio***  
Specifies a runtime ratio for DRC SVRF operations. This ratio is calculated as part of the extra budget ratio used in auto budgeting mode. The range is 0.0-1.0. The default value is 0.1 (10%).
- **MAX\_REMOTE = *value***  
Specifies the maximum number of remotes that the TAT application can request per job. The specified value must be greater than the MIN\_REMOTE value and less than  $2^{32}$ . In most situations it is recommended that you use the default value, which enables the job to grow its resources to the needed potential in order to meet the targeted turn around time. The default is 10000.
- **MIN\_REMOTE = *value***  
Specifies the minimum number of remotes that the TAT application can request for a job. The default is 10.

- **NOTIFICATION\_ESTATUS = [ 0 | 1 | 2 ]**  
Specifies an “exit status” notification event control value that is used when a CalCM job finishes. A notification script, if defined by the NOTIFICATION\_FILE\_PATH argument, is executed when a job finishes with a non-zero exit status. Valid values include:
  - 0 — Enables an “exit status” event if the exit status is not 0. This is the default.
  - 1 — Enables an “exit status” event regardless of the exit status.
  - 2 — Disables an “exit status” event.
- **NOTIFICATION\_EXCEED\_RATIO = *percent***  
Specifies a percentage for detecting the conditions under which to issue a notification event. If the estimated job duration exceeds the total budget by more than the specified percentage, a notification script is executed. The default is 0, which disables the “estimation exceed” and “runtime exceed” events. Note that specifying a small value may generate false warnings since there is a margin of error built into the estimation. A value of 30 or greater is recommended. Refer to “[Notification Events](#)” for more information on the different runtime (or notification) events.
- **NOTIFICATION\_FILE\_PATH = *file\_path***  
Specifies a file which is executed when a runtime event occurs. The supported runtime events include “check point,” “estimation exceed,” and “runtime exceed.” An error is issued if an invalid path is specified. The default is no value, which disables all notification features.
- **NOTIFICATION\_INTERVAL = *minutes***  
Specifies a minimum interval (in minutes) to issue a notification event. The TAT application checks the notification event with each update cycle in CalCM. The default is 0 which causes a notification event to be issued only once.
- **PRIORITY\_BUMP = *value***  
Specifies a global priority bump value for all CalCM TAT jobs. This value can be overridden for a job by specifying the [JOB TAT\\_PRIORITY\\_BUMP](#) in the job configuration file (*job.conf*) or by using the [set\\_priority\\_bump](#) messaging command. Refer to the “[Priority Control](#)” section for more information. The default is 10.
- **PRIORITY\_CHANGE\_ECP = *value***  
Specifies that the priority of a job is increased if the Estimated Completion Percentage (ECP) exceeds the stated budget by *value*.
- **PRIORITY\_CHANGE\_ERT\_RATIO = *value***  
Specifies that the priority of a job is increased if the Estimated Remaining Time (ERT) exceeds the stated budget by *value*. For example, if the budgeted run time is 10 hours, a value of 30 indicates that the priority is increased if the ERT exceeds 13 hours. The job priority is increased by the specified PRIORITY\_BUMP value.

- **PRUNEDBINTERVAL = *interval***

Specifies the interval for the TAT database pruning in seconds. Pruning begins at the next update, and subsequent attempts to prune are done at each interval time. The default interval is 0, which means no pruning. Pruning reduces the size of the database by removing historical information.

- **PRUNEDBSAVETIME = *value***

Specifies the maximum value in seconds that data is saved in a table when the TAT database pruning is attempted. For example, if this value is set to 3600 and pruning is attempted, any data older than 3600 seconds in the table is pruned. The default value is 0, which means no pruning is attempted. Pruning reduces the size of the database by removing historical information.

You can specify the values separately for each table. The default value applies to any tables without explicit values. Syntax for wildcard (\*) and multiple specifications are supported as shown in these examples.

Specify that all tables in the application save up to 3600 seconds of data.

```
PRUNEDBSAVETIME = {* 3600}
```

Specify that the “jobinfo” table saves up to 2400 seconds of data, and the remaining tables save up to 3600 seconds of data.

```
PRUNEDBSAVETIME = {* 3600} \
{jobinfo 2400}
```

- **PRUNEDBSTARTTIME = *time***

Specifies when TAT database pruning occurs. Pruning is attempted at the specified time. If specified with PRUNEDBINTERVAL, the PRUNEDBINTERVAL setting is ignored, and a warning is issued to the transcript. Pruning reduces the size of the database by removing historical information.

The following formats are supported:

- For daily pruning, specify time as *hour:minute*. For example, to prune the database at 1:00 a.m. each day, specify “01:00”.
- For weekly pruning, specify time as *day-of-week-hour:minute*. For example, to prune the database on Monday at 8:00 a.m. each week, specify “Monday-08:00”.
- For monthly pruning, specify time as *week-of-the-month-day-of-week-hour:minute*. For example, to prune the database at 1:00 a.m. on the first Sunday of each month, specify “1st-Sunday-01:00”. To prune on the last Sunday of each month at 1:00 a.m., specify “4th-Sunday-01:00”.

- **SCALE\_OPS = *operation***

Specifies the scalable operations for automatic budget mode. Current valid values include LITHO, OPCBIAS, and OPCSBAR. You can specify multiple operation types in the string (for example “LITHO OPCSBAR”). This argument is designed for TAT controllable

operations and is not recommend for use on other operations, such as DRC. The default is “LITHO”.

## Description

The CalCM TAT application is a Tcl application used to control TAT aspects, such as the budget and priority, of a Calibre job. The default application resides with the CalCM Tcl applications in the MGC\_HOME tree, and defines default values for all arguments. The defaults for all arguments can be overridden by specifying the argument and new value in the CalCM configuration file (*calcmd.conf*) as shown in the example.

## Messaging Commands

You can use the following messaging commands with [calcm\\_send\\_message](#) to change the budget or priority for a specific job:

- [set\\_budget](#)
- [set\\_priority\\_bump](#)

## Examples

```
// CalCM configuration file - calcmd.conf
...
VARIABLE TAT_DB $CALCM_HOME/dbfiles/calcm_tat.db
...

APPLICATION TCL $CALCM_APP/calcm_tat_app.tcl FILE [
    DBPATH = $TAT_DB
    MAX_REMOTE = 1000
    ADJUST_BUDGET = true
    PRIORITY_BUMP = 20
    NOTIFICATION_FILE_PATH = $CALCM_HOME/notify.sh
    NOTIFICATION_EXCEED_RATIO = 30
    NOTIFICATION_INTERVAL = 5
]
...
```

## **calcm\_tcl\_app.tcl**

Creates a cluster snapshot database that can be accessed by other applications or directly by users.

### **Usage**

```
APPLICATION TCL path/calcm_tcl_app.tcl [ FILE '['  
    [PRUNEDBINTERVAL = interval]  
    [SNAPSHOTDIR = snapshot_directory[,permission]]  
'' ]
```

### **Arguments**

- PRUNEDBINTERVAL = *interval*  
Specifies the interval for the snapshot database pruning in seconds. Pruning begins at the next update, and subsequent attempts to prune are done at each interval time. The default interval is 0, which means no pruning. Pruning reduces the size of the database by removing historical information.
- SNAPSHOTDIR = *snapshot\_directory[,permission]*  
Specifies the path to the SQL database. The default is the current directory path. Multiple database files are created in this directory: a database file for primary snapshot databases that manages access to individual snapshot databases, and additional database files for individual snapshots. You can optionally specify the access permission of the snapshot directory and the database files.

Note that the snapshot database is cleaned up during the CalCM daemon (calcmd) startup and at each update cycle (if the reference count is zero, or the PRUNEDBINTERVAL is exceeded).

### **Description**

The calcm\_tcl\_app.tcl application creates a database that manages cluster snapshot data. This data is used by the CalCM web page application through a web server.

### **Examples**

```
APPLICATION TCL calcm_tcl_app.tcl FILE [  
    SNAPSHOTDIR = $CALCM_HTML/snapshot,666  
]
```

# Chapter 5

## CalCM Configuration File Reference

### Dictionary

---

This section provides reference information for the CalCM configuration files, cluster configuration files, and job configuration files.

Refer to “[Syntax Conventions](#)” for information on the conventions used in this chapter.

<b>CalCM Configuration File Reference .....</b>	<a href="#"><b>212</b></a>
<b>CalCM Cluster Configuration File Reference .....</b>	<a href="#"><b>223</b></a>
<b>CalCM Job Configuration File Reference.....</b>	<a href="#"><b>228</b></a>

# CalCM Configuration File Reference

This section describes the statements and associated arguments which can be used in a CalCM configuration file.

A list of these statements and brief description is provided in [Table 5-1](#).

**Table 5-1. Configuration File Commands**

Command Name	Description
<a href="#">APPLICATION</a>	Loads a Tcl application.
<a href="#">DATABASE FLAG</a>	Specifies a flag to change the database access behavior.
<a href="#">LICENSE FEATURES</a>	Specifies the licenses to be managed by CalCM for dynamic licensing.
<a href="#">LICENSE PORT</a>	Specifies the port used by the CalCM daemon to provide licenses for Calibre jobs.
<a href="#">LICENSE SERVER</a>	Specifies a list of CalCM daemons that provide licenses for Calibre jobs.
<a href="#">SERVER INTERVAL</a>	Specifies the update interval.
<a href="#">SERVER PORT</a>	Specifies the port used by the CalCM daemon.
<a href="#">SERVER TIMEOUT</a>	Specifies a timeout value for a job.
<a href="#">VARIABLE</a>	Defines a local environment variable to be used in the configuration.

Refer to “[CalCM Configuration File \(calcmd.conf\)](#)” for more information.

# APPLICATION

Loads a Tcl application.

## Usage

**APPLICATION** **TCL** *path* [*configuration*]

## Arguments

- **TCL** *path*

Specifies the path to the application source.

- *configuration*

An optional clause that can be specified with any application type and has the following syntax:

```
FILE '['  
';'  
']';
```

It allows application-specific sections to be embedded into the main configuration file. The brackets ([ ]) are required. The line breaks (newlines) are required in order to parse the arguments.

## Description

This statement specifies an application to be loaded by CalCM at startup. More than one application statement can be specified in a configuration file. The order in which applications are loaded is undefined and does not necessarily correspond to the order in which they are defined in the configuration file.

## Examples

```
APPLICATION TCL $CALCM_APP/calcm_jobqueue_app.tcl FILE [  
    QUEUEDIR = $CALCM_HOME/jobqueue  
]  
  
APPLICATION TCL $CALCM_APP/calcm_rmonitor_app.tcl FILE [  
    DBPATH = $RMONITOR_DB  
]
```

## **DATABASE FLAG**

Specifies a flag to change the database access behavior.

### **Usage**

**DATABASE FLAG** *flag*

### **Arguments**

- *flag*

Allows control of the database access behavior. The bit set for the flag includes the following values:

- **1** — Specifies to enable MySQL partition support during table creation and pruning. This setting is effective only if MySQL server version is 5.6 or higher.
- **2** — Specifies to enable the database optimization of MySQL database tables in CalCM.
- **3** — Specifies to use the default database access behavior during table creation and pruning.
- **4** — Specifies to enable background database migration for each application simultaneously.

### **Description**

This statement specifies a flag to change the database access behavior. With the MySQL partition support, the behavior of database table pruning is changed. The partition, based on the range, is created at the CalCM daemon startup. Instead of deleting the rows based on the pruning save time, partitions older than the save time are dropped, and new partitions are created. The database optimization does not occur on the partitioned tables.

### **Examples**

```
DATABASE FLAG 2
```

# LICENSE FEATURES

Specifies the licenses to be managed by CalCM for dynamic licensing.

## Usage

**LICENSE FEATURES FILE '['**  
    *feature\_name* [*exact\_access\_date*] = *number* [MIN *minimum\_number*]  
    ...  
    **']'**

## Arguments

- ***feature\_name***  
A required string that specifies the name of the Calibre license to be managed by CalCM.
- ***exact\_access\_date***  
An optional string specifying the minimum Exact Access date using the format YYYY.MM0 (for example, 2012.030 represents March 2012). This value represents the minimum version of the license feature the cluster must support. If a string is not specified for this argument, the latest installed version of the CalCM daemon is used.
- ***number***  
A required integer that specifies the number of licenses to be managed by CalCM.
- **MIN *minimum\_number***  
An optional keyword and integer that specify the minimum number of licenses required for CalCM to start.

## Description

This statement specifies the Calibre license names and the number of licenses to be managed by CalCM for dynamic licensing. Only the specified licenses are managed by CalCM.

You typically use the *exact\_access\_date* argument in one of the following situations:

- You are running an old version of the CalCM daemon and want to ensure support for newer releases by requiring a newer version of licenses to be checked out.
- You want to allow CalCM to check out older licenses to provide support for older Calibre jobs.

## Examples

In the following example, 10 licenses are acquired to run Calibre nmDRC and Calibre nmDRC-H using the release on which CalCM is currently running. The cluster supports Calibre nmOPC for versions released on or before June 2012 (Calibre 2012.2). CalCM starts when a minimum number of ten licenses is acquired.

```
LICENSE FEATURES FILE [
    calibredrc = 10
    calibrehdrc = 10
    calnmopc 2012.060 = 20 MIN 10
]
```

# LICENSE PORT

Specifies the port used by the CalCM daemon to provide licenses for Calibre jobs.

## Usage

**LICENSE PORT** *number*

## Arguments

- *number*

A required integer that specifies a TCP port number. The default value is 9901.

## Description

This statement specifies the port used by the CalCM daemon to provide licenses for Calibre jobs. This statement is only required if you want to specify a port other than the default (9901).

## Examples

```
LICENSE PORT 9876
```

## Related Topics

[LICENSE SERVER](#)

# LICENSE SERVER

Specifies a list of CalCM daemons that provide licenses for Calibre jobs.

## Usage

**LICENSE SERVER** *host[:port][;host[:port]...]*

## Arguments

- *host*

A required value that specifies the hostname of the CalCM daemon. The short hostname should be used.

- *port*

Specifies the port number, which must be the same value used for the LICENSE PORT statement. The default is 9901.

## Description

This statement specifies a list of CalCM daemons that provide licenses from the License Broker Daemon (LBD) for Calibre jobs. This list is used by Calibre jobs in the event of a licensing failure. Calibre jobs attempt to reconnect to these servers in the order listed until the licenses can be recovered.

This statement is required when backup CalCM daemons are used and must list the primary and all backup CalCM daemons and port numbers. This statement is not required for configurations that are not running a backup CalCM daemon.

## Examples

```
LICENSE PORT 9876
LICENSE SERVER calcml:9876;calcm2:9876
```

## Related Topics

[LICENSE PORT](#)

# SERVER INTERVAL

Specifies the update interval.

## Usage

**SERVER INTERVAL *number***

## Arguments

- ***number***

Specifies an interval time in seconds. The default value is 120 seconds.

---

### **Caution**

---

 You should use the default of 120 seconds unless you are performing debugging operations.

---

## Description

This statement specifies the update interval of the snapshot in seconds. It is recommended that the default value be used.

## Examples

```
SERVER INTERVAL 120
```

## **SERVER PORT**

Specifies the port used by the CalCM daemon.

### **Usage**

**SERVER PORT** *number*

### **Arguments**

- *number*

Specifies the port number to be used by the CalCM daemon. The default port number is 9900. It is recommended that you do not use well known TCP ports less than 1024.

### **Description**

This statement specifies the port number to be used by the CalCM daemon.

### **Examples**

```
SERVER PORT 9900
```

# **SERVER TIMEOUT**

Specifies a timeout value for a job.

## **Usage**

**SERVER TIMEOUT** *number*

## **Arguments**

- *number*

Specifies the timeout value of a job update in seconds. The default value is 45 seconds. The job update will abort if there is no response from the job within the specified time.

## **Description**

This statement specifies a timeout value for a job update in seconds.

## **Examples**

```
SERVER TIMEOUT 30
```

# **VARIABLE**

Defines a local environment variable to be used in the configuration.

## **Usage**

**VARIABLE *var value***

## **Arguments**

- ***var***

Specifies the variable name. You can define a variable name with or without a “\$” prefixed to the variable name (see example).

- ***value***

Specifies the value assigned to the variable. If the value contains a space character, it must be enclosed in quotation marks (“ ”) or braces ({ }).

## **Description**

This statement defines a local environment variable to be used in the configuration. It is typically used to define variables used in the CalCM configuration file (*calcmd.conf*). You can then use the “\$” prefix to call a variable in *calcmd.conf* and CalCM Tcl applications. Using this statement is similar to using the “set var” or “setenv” command to define a variable for a shell environment.

## **Examples**

```
VARIABLE CALCM_HOME /home/calcm
VARIABLE $CALCM_PLATFORM SSH
```

# CalCM Cluster Configuration File Reference

This section provides reference information for the CalCM cluster configuration statements available for use in the cluster configuration file.

For more information on the cluster configuration file, refer to “[Cluster Configuration File \(cluster.conf\)](#)” and “[Configuring the CalCM Environment](#)”. A list of commands and a brief description is provided in [Table 5-2](#).

**Table 5-2. Cluster Configuration Commands**

Command Name	Description
<a href="#">CLUSTER NAME</a>	Specifies a cluster that represents hostnames.
<a href="#">MASTER HOST</a>	Specifies a primary host.
<a href="#">RDS HOST</a>	Specifies an RDS (Remote Data Server) host.
<a href="#">REMOTE HOST</a>	Specifies a remote host.

## CLUSTER NAME

Specifies a cluster that represents hostnames.

### Usage

**CLUSTER NAME** *name* HOSTS '['*hostname* ...']'

### Arguments

- *name*  
Specifies a cluster name. Only alphanumeric or underscore (\_) characters are allowed.
- *hostname*  
Specifies one or more hostnames that belong to the cluster.

### Description

This statement declares a cluster that represents the specified hostnames. The declared cluster name preceded by the at (@) symbol can be specified in place of a hostname. It can be used in cluster configuration (*cluster.conf* file) and resource-related job configuration (*job.conf* file).

### Examples

This example declares five cluster names with specified hostname groups. The cluster names are then used in **MASTER HOST**, **REMOTE HOST**, and **RDS HOST** statements.

```
CLUSTER NAME primaries HOSTS [primary01 primary02]
CLUSTER NAME remotes1 HOSTS [node20001 node30002 node30001]
CLUSTER NAME remotes2 HOSTS [node10001 node10003 node10002]
CLUSTER NAME remotes HOSTS [
@remotes1 @remotes2
]

CLUSTER NAME rds HOSTS [
node10002 node10003
node40001 //node40001
]

MASTER HOST @primaries SLOT 2
REMOTE HOST @remotes SLOT 8
RDS HOST @rds SLOT 4
```

# MASTER HOST

Specifies a primary host.

## Usage

**MASTER HOST *name* SLOT *njobs***

## Arguments

- ***name***

Specifies a primary hostname or cluster name. Cluster names must be preceded by an at (@) symbol).

- **SLOT *njobs***

Specifies the number of job slots that can be run in parallel on the primary host.

## Description

This statement specifies a primary host for running Calibre jobs along with the number of job slots. You can specify multiple MASTER HOST statements in the cluster configuration file.

## Examples

```
MASTER HOST host123 SLOT 4
MASTER HOST @cluster SLOT 4
```

## RDS HOST

Specifies an RDS (Remote Data Server) host.

### Usage

**RDS HOST *name* SLOT *nslots***

### Arguments

- ***name***  
Specifies the hostname of the RDS host or cluster name with a preceding at (@) symbol.
- **SLOT *nslots***  
Specifies the maximum number of RDS that can be launched on the host.

### Description

This statement specifies an RDS host. It can be used by specifying **REMOTE DATA** in the job configuration file. You can specify more than one RDS host in the cluster configuration file. The RDS backup server is not counted toward the RDS host slot number.

### Examples

```
RDS HOST remote SLOT 2
RDS HOST @cluster SLOT 2
```

# REMOTE HOST

Specifies a remote host.

## Usage

**REMOTE HOST** *name* [SLOT *ncpuslot*] [RDS *nslots*]

## Arguments

- ***name***  
Specifies the name of the remote host or a cluster name (with a preceding at (@) symbol).
- **SLOT *ncpuslot***  
Specifies the number of slots on the remote host to use for the job.
- **RDS *nslots***  
Specifies the maximum number of RDS that can be launched on the remote host. The detailed information can be found in [RDS HOST](#).

## Description

This statement specifies a remote host that is managed by CalCM and allocated for Calibre jobs. You can specify multiple remote host statements in the cluster configuration file.

## Examples

```
REMOTE HOST hostname SLOT 4
REMOTE HOST @cluster SLOT 4
```

# CalCM Job Configuration File Reference

This section provides reference information for the CalCM job configuration statements available for use in the job configuration file.

Refer to “[Job Configuration File \(job.conf\)](#)” and “[Configuring Calibre Jobs to Run Under CalCM](#)” for more information. A list of commands and brief description is provided in [Table 5-3](#).

**Table 5-3. CalCM Job Configuration Commands**

Command Name	Description
<a href="#">JOB BUDGET</a>	Specifies a budget time for operations.
<a href="#">JOB DIRECTORY</a>	Specifies the directory containing the job you want to run.
<a href="#">JOB ENV</a>	Specifies the path to the job environment file.
<a href="#">JOB INFO</a>	Specifies to save detailed job information.
<a href="#">JOB KEY</a>	Specifies a job key.
<a href="#">JOB LOG</a>	Specifies a log file name for an SVRF rule file.
<a href="#">JOB MODE</a>	Specifies the run mode to be used for an SVRF rule file.
<a href="#">JOB MTFLEX</a>	Specifies the job specific Calibre MTflex configuration.
<a href="#">JOB MTFLEX_OVERRIDE</a>	Specifies to override or append to the existing job specific Calibre MTflex configuration.
<a href="#">JOB NOTIFICATION</a>	Specifies a type of notification to be sent during a job.
<a href="#">JOB NOTIFICATION_FILTERS</a>	Specifies a notification filter that is applied to the job’s transcript.
<a href="#">JOB NOTIFICATION_TRACKERS</a>	Specifies a notification tracker that is applied to the job’s transcript.
<a href="#">JOB POST_EXEC/JOB_POST_EXEC_CHECK</a>	Specifies a file to be executed when the Calibre job completes.
<a href="#">JOB PRE_EXEC/JOB_PRE_EXEC_CHECK</a>	Specifies a file to be executed before running Calibre.
<a href="#">JOB PRIORITY</a>	Specifies a priority for a Calibre job.
<a href="#">JOB QUOTA</a>	Specifies a quota string for a Calibre job.
<a href="#">JOB RULE</a>	Specifies an SVRF rule file name.
<a href="#">JOB SLOT</a>	Specifies the number of slots required for running a job.
<a href="#">JOB_TAT_PRIORITY_BUMP</a>	Specifies a new priority bump for job.

**Table 5-3. CalCM Job Configuration Commands (cont.)**

<b>Command Name</b>	<b>Description</b>
<b>JOB TOTAL</b>	Specifies the total number of rule files to be executed for the job.
<b>JOB USER</b>	Specifies a user name for the job.
<b>LAUNCH DELAY</b>	Specifies the amount of delay time before launching RCS or RDS hosts.
<b>LAUNCH MASTER_NAME</b>	Specifies the primary host to be used for remote connections.
<b>LAUNCH MAXCOUNT</b>	Specifies a maximum number of remotes that can be launched for a job.
<b>LAUNCH SMTFACTOR</b>	Specifies whether to allow hyperthreading in Calibre jobs running on remotes.
<b>LAUNCH WAIT</b>	Specifies the amount of time that Calibre waits for a connection to be made.
<b>MASTER PREFERRED</b>	Specifies a list of preferred nodes for a primary host.
<b>MASTER RESOURCE</b>	Specifies a resource requirement for a primary host.
<b>REMOTE COUNT</b>	Specifies an initial number of remotes.
<b>REMOTE COUNT_MAX</b>	Specifies the maximum number of remotes in a Calibre MTflex configuration.
<b>REMOTE COUNT_MIN</b>	Specifies the minimum number of remotes required to start a job in a Calibre MTflex configuration.
<b>REMOTE DATA</b>	Specifies the number of remote data servers.
<b>REMOTE DATA_MIN</b>	Specifies the minimum number of remote data servers (RDS) to start a job.
<b>REMOTE DATA_RECOVEROFF</b>	Specifies whether to launch backup remote data servers.
<b>REMOTE DATA_RESOURCE</b>	Specifies a resource requirement for remote data server (RDS) nodes.
<b>REMOTE ENV</b>	Specifies an environment variable file name for Calibre remotes.
<b>REMOTE INIT_RESOURCE</b>	Specifies a resource requirement for initial remote nodes.
<b>REMOTE MAX</b>	Specifies a maximum number of remotes.
<b>REMOTE MAXCHANGE</b>	Specifies the maximum number of remote resources that can be added to a job in a single CalCM cycle.
<b>REMOTE MIN</b>	Specifies a minimum number of remotes.

**Table 5-3. CalCM Job Configuration Commands (cont.)**

Command Name	Description
REMOTE MINMAX	Specifies a minimum and maximum number of remote hosts for each job operation.
REMOTE PREFERRED	Specifies a list of preferred nodes for a remote host.
REMOTE RAMPUPDOWN	Specifies a ramp up/down ratio for each operation to adjust the job's CPU demand.
REMOTE RESOURCE	Specifies a resource requirement for remote hosts.

# JOB BUDGET

Specifies a budget time for operations.

## Usage

```
JOB BUDGET {[CHECK:]operation_name1=budget};  
{[CHECK:]{operation_nameN=budget} ...} [_EXTRA_=budget] {_EONLY_}  
JOB BUDGET [_AUTO_]_AUTO2_];[TOT:budget];[EXT:budget] [_EONLY_]
```

## Arguments

- **CHECK:**  
Specifies that a check point notification is sent when the associated *operation\_name* argument exceeds the specified *budget*. The CHECK argument can be specified multiple times in a budget string.
- ***operation\_name1=budget***  
Specifies the name of an operation and an associated budget (in minutes). The operation name is the name announced by Calibre when the operation is executed. This argument may be used multiple times to specify a budget for each operation. Each definition must be separated by a “;” (semi-colon).
- **\_EXTRA\_=*budget***  
This optional keyword is used to assign a budget value (in minutes) for all non-LITHO operations. This argument should only be specified once.
- **\_AUTO\_**  
Specifies that TAT should use the automatic budgeting mode to define the budget for job operations. TAT uses data collected from previous runs to determine the budget.
- **\_AUTO2\_**  
Specifies that TAT should use the automatic budgeting mode to define the budget for job operations. In this mode, TAT evenly allocates the budget to scalable operations having an operations class that matches pre-defined operation types specified by the SCALE\_OPS argument in [calcm\\_tat\\_app.tcl](#).  
  
For example, if the operations specified for the SCALE\_OPS argument are “LITHO OPCSBAR,” then TAT treats LITHO and OPCSBAR operations as scalable and the operations are budgeted.
- **TOT:*budget***  
Specifies the total budget for the job.
- **EXT:*budget***  
An optional argument that specifies an extra budget in the total budget.

- EONLY

Enables an “estimation only” mode in which TAT calculates the max value. In this mode, the TAT application does not change the job’s CPU demand or priority but changes the job’s max value that is used by [calcm\\_jobqueue\\_app.tcl](#) to limit submitting a job when the CLUSTERLIMIT argument is specified. The budgeted job overwrites the max value by computing the optimal resource count demand on the budget.

## Description

The JOB BUDGET statement is used in the job configuration file to specify the budget time for the following supported Calibre operations:

- nmOPC
- OPCverify
- OPCbias
- OPCsbar
- MDP Embedded SVRF

The TAT application overrides the specified JOB BUDGET when the following conditions are met:

- The ADJUST\_BUDGET argument in the [calcm\\_tat\\_app.tcl](#) application is set to “TRUE.”
- The duration of the job is exceeding an accumulated budget value that is calculated by the TAT application.

Based on data collected from the job and operations, the TAT application calculates and applies an adjusted budget for the remaining operations in the job.

## Examples

### Example 1

This example assigns a budget of 120 minutes to the operation named nmOPC1 and 300 minutes to the operation named verify.

```
JOB BUDGET nmOPC1=120;verify=300
```

### Example 2

This example assigns a budget of 200 minutes to the operation named nmOPC1 and all other non-LITHO operations are assigned a budget of 30 minutes.

```
JOB BUDGET nmOPC1=200;_EXTRA_=30
```

### Example 3

This example assigns a budget of 120 minutes to all non-LITHO operations.

---

```
JOB BUDGET _EXTRA_=120
```

#### Example 4

This example issues a check point notification event when a drc\_out operation does not finish within 60 minutes.

```
JOB BUDGET CHECK:drc_out=60
```

#### Example 5

This example issues a check point notification event when a drc\_out operation does not finish within 60 minutes or the opc\_in operation does not finish within 120 minutes.

```
JOB BUDGET CHECK:drc_out=60;CHECK:opc_in=120
```

#### Example 6

This example issues a check point notification event when an opc\_out operation does not finish within 400 minutes or a drc\_out operation does not finish within 60 minutes.

```
JOB BUDGET opc_out=400;CHECK:drc_out=60
```

## **JOB DIRECTORY**

Specifies the directory containing the job you want to run.

### **Usage**

**JOB DIRECTORY** *dir\_path*

### **Arguments**

- *dir\_path*

Specifies a directory path that contains the job you want to run. The default path is where the job configuration file (*job.conf*) is located. See “[Job Configuration File \(job.conf\)](#)” on page 28. It is not recommended to use relative path (./) or tilde path (~*user*/) specifications, because these locations can change at the time of the job launch.

### **Description**

This statement specifies the directory path containing the job you want to run. You can use this statement to separate the directory that contains the design files for the job from the directory that contains the job configuration files. If the directory path specified by JOB DIRECTORY in the *job.conf* file is inaccessible, CalCM issues a warning in the daemon transcript, and the job is not submitted.

### **Examples**

```
JOB DIRECTORY /home/user/opc
```

# **JOB ENV**

Specifies the path to the job environment file.

## Usage

**JOB ENV** *fname*

## Arguments

- *fname*

Specifies the name of the job environment file. The file is fetched from the directory specified in **JOB DIRECTORY**, or by default, from the path where the job configuration file (*job.conf*) is located.

## Description

This statement specifies the name of the job environment file, which is used to define job-specific environment variables. The job environment file is used for the Calibre run associated with the most recent **JOB RULE** statement in the job configuration file. Refer to “[Job Environment File \(job.env\)](#)” for more information on the job environment file.

## Examples

```
JOB ENV test.env
```

## JOB INFO

Specifies to save detailed job information.

### Usage

**JOB INFO {*path* | *pairs*}**

### Arguments

- ***path***

Specifies a path to a file that contains job information keyword and value pairs. The space-separated pairs are specified one per line. Specifications for values should not contain spaces or special characters. See [Table 5-4](#) for valid keywords.

*job\_information\_keyword* *job\_information\_value*

- ***pairs***

Specifies a list of comma-separated pairs. The pairs contain the job information keyword and value separated by an equal sign (=). See [Table 5-4](#) for valid keywords.

*job\_information\_keyword*=*job\_information\_value*

**Table 5-4. Job Information Keywords**

Keyword	Data Type	Example Value	Comments
node	numeric	8, 10, 14, 28	Foundry process name.
tech_node	alphanumeric	Foundry node including variation name (FinFet, FDSOI)	Precise technology node.
device_name	alphanumeric	SSUVW01, S7LMN02	Multiple-project wafer (MPW) name, or similar to prime_die for a single-chip mask order.
layer_group	alphanumeric	Implant, FrontEnd, BackEnd	Name for a group of layers that share similarities.
layer	alphanumeric	M1, M2, V0	Precise layer name.
layer_revision	alphanumeric	M1XX, V0XX	Foundry-specific mask name.
module	alphanumeric	DRC, OPC, ORC, MPC, MDP ...	Type of job.
db_area	numeric	1.2, 12.6, 50, 200	Silicon size in mm <sup>2</sup> .
keyword	alphanumeric	<i>rules.svrf</i>	Name of the SVRF keyword file.

**Table 5-4. Job Information Keywords (cont.)**

<b>Keyword</b>	<b>Data Type</b>	<b>Example Value</b>	<b>Comments</b>
keyword_version	alphanumeric	1.12, 1.14, 1.12.2.1	Version of the SVRF keyword file. Can contain SVRF version and branches.
mask_set	alphanumeric	SNWXYZ1, MLW1610	Reference for the set of masks used together to implement a chip.
order_num	alphanumeric	Specifications for front-end mask order, back-end mask order, ...	Foundry convention of mask order.
prime_die	alphanumeric	Chip10ABC, P10	Chip name, unlike device_name that can be a MPW name.
time_to_mask	numeric	84	Time (hours) allocated to deliver mask data to mask shop.
runtime_alert	numeric	12	Time (hours) alert threshold for the type of job.

## Description

The JOB INFO statement enables you to store specified job information for further analysis in the job information (info\_jobs) database table during the job submission.

## Examples

### Example 1

This example uses the JOB INFO statement with a file path specification:

```
JOB INFO job.info
```

The keywords and values are space-separated.

node	<i>8</i>
tech_node	<i>LN10ABC</i>
device_name	<i>SFUVWXYZ01</i>
layer_group	<i>BackEnd</i>
layer	<i>M1</i>
layer_revision	<i>V0BZ</i>
module	<i>OPC</i>
db_area	<i>12.6</i>
keyword	<i>SVRFRules</i>
keyword_version	<i>1.16</i>
mask_set	<i>SNPQRS1</i>
order_num	<i>1.4</i>
prime_die	<i>P10</i>
time_to_mask	<i>84</i>
runtime_alert	<i>12</i>

### Example 2

This example uses the JOB INFO statement with keyword and value pair specifications separated by the equal sign (=) in a comma-separated list. The example is formatted to fit the page (specify the list of pairs as a single line).

```
JOB INFO node=8,tech_node=LN10ABC,device_name=SFUVWXYZ01,  
layer_group=BackEnd,layer=M1,layer_revision=V0BZ,module=OPC,  
db_area=12.6,keyword=SVRFRules,keyword_version=1.16,mask_set=SNPQRS1,  
order_num=1.4,prime_die=P10,time_to_mask=84,runtime_alert=12
```

## **JOB KEY**

Specifies a job key.

### **Usage**

**JOB KEY** *job\_key*

### **Arguments**

- *job\_key*

Specifies an arbitrary string which is assigned to the job.

### **Description**

The JOB KEY statement allows you to assign a name to a Calibre job. The JOB KEY name is stored in the TAT database and can be used to access a specific job record in the database.

### **Examples**

```
JOB KEY job1
```

## **JOB LOG**

Specifies a log file name for an SVRF rule file.

### **Usage**

**JOB LOG** *fname*

### **Arguments**

- *fname*

Specifies the name of a log file.

### **Description**

This statement specifies a log file name for the Calibre job. The log file is associated with the most recent **JOB RULE** statement in the job configuration file. You can use “%J”, “%ID”, or “\$jobID” in the JOB LOG statement *fname* parameter to include the CalCM ID value of the job as part of the output file name.

### **Examples**

```
JOB RULE test.drc
JOB MODE MTFLEX
JOB LOG test.%J.log
```

# **JOB MODE**

Specifies the run mode to be used for an SVRF rule file.

## **Usage**

**JOB MODE** *mode*

## **Arguments**

- *mode*

Specifies the mode for running an SVRF rule file. Available options include:

FLAT\_MT — Flat and multi-threaded mode.  
FLAT\_MTFLEX — Flat and Calibre MTflex mode.  
MT — Multi-Threaded mode without remotes.  
MTFLEX — Calibre MTflex mode with remotes.  
HYPER\_MT — Hyperscaling and multi-threaded mode.  
HYPER\_MTFLEX — Hyperscaling and Calibre MTflex mode.  
HYPER\_REMOTE\_MTFLEX — Hyperscaling and Calibre MTflex mode with remotes.

Refer to the [Calibre Administrator's Guide](#) for the command options for these modes.

## **Description**

This statement specifies the Calibre run mode to be used for an SVRF rule file. The specified mode is associated with the most recent **JOB RULE** statement in the job configuration file.

A different mode can be specified for each rule file.

## **Examples**

```
JOB RULE test.drc  
JOB MODE MTFLEX
```

```
JOB RULE verify.drc  
JOB MODE MT
```

## **JOB MTFLEX**

Specifies the job specific Calibre MTflex configuration.

### **Usage**

**JOB MTFLEX cmdlines**

### **Arguments**

- **cmdlines**

Specifies the Calibre MTflex configuration commands. The command lines must be separated by a new line character (\n) and contained within one line.

### **Description**

This statement is used to specify job specific configuration commands for running with Calibre MTflex functionality. Using this statement replaces the existing default Calibre MTflex configuration commands used by CalCM. Refer to the “[Configuration File Reference Dictionary](#)” in the *Calibre Administrator’s Guide* for information on the configuration statements.

### **Examples**

This example Calibre MTflex command line specifies a system monitoring interval “4” to standard output, sets the cluster parameters for the connection of the local host to the remote hosts, and defines the storage location for temporary files.

```
JOB MTFLEX MONITOR SYSTEM HEARTBEAT 4 PRINT\nLAUNCH CLUSTER COUNT
$clusterCount\nLOCAL HOST DIR $tempDir
```

## **JOB MTFLEX\_OVERRIDE**

Specifies to override or append to the existing job specific Calibre MTflex configuration.

### **Usage**

**JOB MTFLEX\_OVERRIDE cmdlines**

### **Arguments**

- *cmdlines*

Specifies the Calibre MTflex configuration commands. The command lines must be separated by a new line character (\n) and contained within one line.

### **Description**

This statement is used to specify job specific configuration commands for running with Calibre MTflex functionality. If the first two specified keywords match an existing Calibre MTflex configuration command, the existing command is overridden; otherwise, the JOB MTFLEX\_OVERRIDE configuration command is appended. Refer to the “[Configuration File Reference Dictionary](#)” in the *Calibre Administrator’s Guide* for information on the configuration statements.

### **Examples**

This example Calibre MTflex command line specifies a system monitoring interval “2” to standard output, and prints the current memory usage for each remote host into a log file every 10 minutes “600”. The command line overrides (if matching condition applies) or appends to existing Calibre MTflex configuration commands.

```
JOB MTFLEX_OVERRIDE MONITOR SYSTEM HEARTBEAT 2 PRINT\nREMOTE INTERVAL 600
MEMORY
```

## JOB NOTIFICATION

Specifies a type of notification to be sent during a job.

### Usage

**JOB NOTIFICATION** *types*

### Arguments

- *types*

Specifies the job notification types. The allowed types are as follows:

RETRY — The job failed to launch and will be retried in the next update cycle.  
START — The job started.  
ANNOUNCE — The job is announced by Calibre.  
SUSPEND — The job is suspended with [suspend\\_job](#) message.  
RESUME — The job is resumed with [resume\\_job](#) message.  
END — The job finished.  
FAIL — The job failed.

### Description

This statement specifies the notification types that will be sent to the job administrator. By default, *calcm\_notification\_app.tcl* is used to send the notification. Alternatively, the **JOBNOTIFYFYCMD** command in *calcm\_jobqueue\_app.tcl* can be used to send the notification. The job notification types must be separated by the vertical bar ( | ) character.

### Examples

```
JOB NOTIFICATION START | RESUME | SUSPEND | END | FAIL
```

# **JOB\_NOTIFICATION\_FILTERS**

Specifies a notification filter that is applied to the job's transcript.

## **Usage**

**JOB\_NOTIFICATION\_FILTERS** *filters*

## **Arguments**

- *filters*

Specifies regular expression filters that are applied to the job's transcript.

## **Description**

The filters specify regular expressions that are applied to each line of the job's transcript. If a match is found, it is recorded in the notification database. The filters can be separated with a backslash (\) character and a new line. The syntax “{@INHERIT}” can be specified to incorporate the JOBEVENTFILTERS option specified in [calcm\\_notification\\_app.tcl](#).

## **Examples**

```
JOB_NOTIFICATION_FILTERS {@INHERIT} \
{^WARNING: .*license} \
{^MTFLEX RUNTIME WARNING}
```

## **JOB\_NOTIFICATION\_TRACKERS**

Specifies a notification tracker that is applied to the job's transcript.

### **Usage**

**JOB\_NOTIFICATION\_TRACKERS** *trackers*

### **Arguments**

- *trackers*

Specifies the notification trackers that are applied to the job's transcript.

### **Description**

The filters specify the notification trackers that are applied to each line of the job's transcript. The syntax “{@INHERIT}” can be specified to incorporate the JOBEVENTTRACKERS option specified in [calcm\\_notification\\_app.tcl](#).

### **Examples**

```
JOB_NOTIFICATION_TRACKERS {@INHERIT} \
 {"starting_time" -1 {\//\// Starting time:}}
```

## JOB POST\_EXEC/JOB POST\_EXEC\_CHECK

Specifies a file to be executed when the Calibre job completes.

### Usage

**JOB POST\_EXEC** *filename parameters*

**JOB POST\_EXEC\_CHECK** *filename parameters*

### Arguments

- *filename*

Specifies the name of the file to execute.

- *parameters*

Specifies any parameters that should be passed to *filename* when it is executed. The following variables are replaced with the appropriate values when submitting a job to run under CalCM:

\$jobID — The ID of the job.

\$jobDirPath — The path from which the job is launched.

\$jobPriority — The priority specified by **JOB PRIORITY**.

\$jobMode — The mode of the job specified by **JOB MODE**.

\$launchName — The name of the primary host.

\$jobRule — The name of the rule file specified by **JOB RULE**.

\$jobLog — The name of the log file specified by **JOB LOG**.

\$jobEnv — The name of the environment file specified by **JOB ENV**.

\$jobQuota — The quota value specified by **JOB QUOTA**.

\$jobUser — The username specified by **JOB USER**.

### Description

This statement specifies the name of a file to be executed after the Calibre job completes. The parameters are passed to the specified file when it is executed. The exit status of the Calibre job is passed through the environment variable CALIBRE\_EXIT\_STATUS. A status of '0' means no error.

The JOB POST\_EXEC\_CHECK is the same as the JOB POST\_EXEC except that it is executed only if the Calibre job ran without any problems.

### Examples

```
JOB POST_EXEC echo $jobUser >>& $jobLog
```

## **JOB PRE\_EXEC/JOB PRE\_EXEC\_CHECK**

Specifies a file to be executed before running Calibre.

### **Usage**

**JOB PRE\_EXEC** *filename parameters*

**JOB PRE\_EXEC\_CHECK** *filename parameters*

### **Arguments**

- ***filename***

Specifies the name of a file to execute.

- ***parameters***

Specifies any parameters that should be passed to ***filename*** when it is executed. The following variables are replaced with the appropriate values when submitting a job to run under CalCM:

\$jobID — The ID of the job.

\$jobDirPath — The path from which the job is launched.

\$jobPriority — The priority specified by **JOB PRIORITY**.

\$jobMode — The mode of the job specified by **JOB MODE**.

\$launchName — The name of the primary host.

\$jobRule — The name of the rule file specified by **JOB RULE**.

\$jobLog — The name of the log file specified by **JOB LOG**.

\$jobEnv — The name of the environment file specified by **JOB ENV**.

\$jobQuota — The quota value specified by **JOB QUOTA**.

\$jobUser — The username specified by **JOB USER**.

### **Description**

This statement specifies the name of a file to be executed prior to running Calibre. The parameters are passed to the specified file when it is executed.

The **JOB PRE\_EXEC\_CHECK** is the same as the **JOB PRE\_EXEC** except that it exits the *calcm\_run* script if the execution file fails.

### **Examples**

```
JOB PRE_EXEC echo $jobID >& $jobLog
```

# **JOB PRIORITY**

Specifies a priority for a Calibre job.

## **Usage**

**JOB PRIORITY *number***

## **Arguments**

- ***number***

Specifies a priority. The default minimum value is 0 and the maximum value is 16382. A larger value implies a higher priority.

## **Description**

This statement specifies a priority for a Calibre job. If the priority for a job is higher than other jobs, it is dispatched first and the job is given priority over the allocation of resources.

Whenever there is a demand by a job with a higher priority, the resource manager will attempt to allocate remotes by revoking remotes from the lowest priority job. The job with the lowest priority can still run with a minimum number of remotes. Refer to “[CalCM Resource Management](#)” for more information.

## **Examples**

```
JOB PRIORITY 2
```

## **JOB QUOTA**

Specifies a quota string for a Calibre job.

### **Usage**

**JOB QUOTA** *string*

### **Arguments**

- *string*

Specifies a quota string using the following syntax:

**GROUP[/GROUP]**

### **Description**

This statement specifies a quota string for a Calibre job. The first value in the quota string identifies the department that is running the job.

This information is passed to the resource manager application which accesses a quota configuration table and builds a quota hierarchy based on the specified quota *string*. Refer to “[Quota Configuration File \(calcm\\_quota.conf\)](#)” for details on building the quota hierarchy for a job.

### **Examples**

`JOB QUOTA D2/P3`

# **JOB RULE**

Specifies an SVRF rule file name.

## **Usage**

**JOB RULE** *fname*

## **Arguments**

- *fname*

Specifies the name of an SVRF rule file.

## **Description**

This statement specifies the name of an SVRF rule file name that is used for part or all of a Calibre job. You can specify multiple JOB RULE statements in the job configuration file. The **JOB LOG** and **JOB MODE** statements are automatically associated with the most recent JOB RULE statement in the job configuration file.

## **Examples**

```
JOB TOTAL 2  
  
JOB RULE test.drc  
JOB MODE MTFLEX  
JOB LOG test.%J.log  
  
JOB RULE verify.drc  
JOB MODE MT  
JOB LOG verify.%J.log
```

## **JOB SLOT**

Specifies the number of slots required for running a job.

### **Usage**

**JOB SLOT *number***

### **Arguments**

- ***number***

Specifies the number of slots needed to run a job. The default is 1.

### **Description**

This statement specifies the number of slots on the primary host that are needed to run a job. Only the primary host with enough empty slots can run a job. If a primary host is available with enough slots, the job is left in the job queue and the next job is launched.

### **Examples**

```
JOB SLOT 2
```

# **JOB TAT\_PRIORITY\_BUMP**

Specifies a new priority bump for job.

## Usage

**JOB TAT\_PRIORITY\_BUMP *number***

## Arguments

- *number*

Specifies a priority bump value which overrides the global priority bump value.

## Description

The JOB TAT\_PRIORITY\_BUMP statement specifies a new priority bump value for a job. This value overrides the global priority bump value defined by the [calcm\\_tat\\_app.tcl](#) application.

## Examples

```
JOB TAT_PRIORITY_BUMP 20
```

## **JOB TOTAL**

Specifies the total number of rule files to be executed for the job.

### **Usage**

**JOB TOTAL *number***

### **Arguments**

- ***number***

Specifies the total number of rule files in the job configuration file to be executed for the job.

### **Description**

This statement specifies the total number of rule files to be executed for the job. If this statement is not specified, then all rules specified by **JOB RULE** statements in the configuration file are executed for the job.

### **Examples**

```
JOB TOTAL 2

JOB RULE test.drc
JOB MODE MTFLEX

JOB RULE verify.drc
JOB MODE MT
```

# **JOB USER**

Specifies a user name for the job.

## **Usage**

**JOB USER** *name*

## **Arguments**

- *name*

Specifies a user name.

## **Description**

This statement specifies a user name that is stored in the job database and displayed in an HTTP job page. The user name can be used for sending an e-mail if the user name is matched to an e-mail address. You can also use the following format to specify the cost center for the job:

**JOB USER** *cost\_center1/johnd*

## **Examples**

**JOB USER** johnd

## LAUNCH DELAY

Specifies the amount of delay time before launching RCS or RDS hosts.

### Usage

**LAUNCH DELAY** *expr*

### Arguments

- *expr*

Specifies a delay time in milliseconds that can be specified in a Tcl expression as follows:

[expr round(rand() \* 500)]

In LSF and Grid, the LSB\_JOBINDEX and SGE\_TASK\_ID environment variables can be used to throttle array jobs.

### Description

This statement specifies a delay time before launching each RCS or RDS host which allows the Calibre primary host to handle the bottleneck of accepting incoming connection requests. When LAUNCH DELAY is specified in the *job.conf* file, it overrides LAUNCHDELAY in the *calcmd.conf* file under the [calcm\\_jobqueue\\_app.tcl](#) application.

You can specify a value of 0 for LAUNCH DELAY for no delay in the launching of RCS or RDS hosts (this is the default value).

### Examples

```
LAUNCH DELAY [expr round(rand() * 500)]
```

## LAUNCH MASTER\_NAME

Specifies the primary host to be used for remote connections.

### Usage

**LAUNCH MASTER\_NAME** *host*

### Arguments

- *host*

Specifies the IP address or hostname of the primary host.

### Description

This statement specifies the hostname (or IP address) of the primary host. The connection name used by the remote hosts defaults to the local hostname() returned by the operating system. The specified name is used by the NAME argument in the [LAUNCH CLUSTER](#) statement. This statement should not be used unless it is required for some reason, such as forcing the use of a specific network interface other than the default.

### Examples

```
LAUNCH MASTER_NAME 192.168.1.100
```

## LAUNCH MAXCOUNT

Specifies a maximum number of remotes that can be launched for a job.

### Usage

**LAUNCH MAXCOUNT *number***

### Arguments

- *number*

Specifies a maximum number of remotes.

### Description

This statement specifies the maximum number of remotes that can be launched for a job. For jobs launched with hyperthreading support, the LAUNCH MAXCOUNT number should be specified with 2x the number of remotes specified by [REMOTE COUNT\\_MAX](#) to allow for the extra number of remotes.

### Examples

```
LAUNCH MAXCOUNT 120
```

# LAUNCH SMTFACTOR

Specifies whether to allow hyperthreading in Calibre jobs running on remotes.

## Usage

**LAUNCH SMTFACTOR** *flag*

## Arguments

- *flag*

Specifies whether or not to enable hyperthreading. Valid values include:

- 1 — Enables hyperthreading. This is the default.
- 0 — Disables hyperthreading.

## Description

This statement is used to enable or disable support for hyperthreading when running Calibre jobs on remotes. For example, specifying “LAUNCH CLUSTER ... SMTFACTOR 0” in the calcm\_mtflex.conf file, sets the flag to 0, and hyperthreading is disabled. Refer to [LAUNCH CLUSTER](#) in the *Calibre Administrator's Guide* for detailed information.

## Examples

```
LAUNCH SMTFACTOR 0
```

## LAUNCH WAIT

Specifies the amount of time that Calibre waits for a connection to be made.

### Usage

**LAUNCH WAIT *number***

### Arguments

- ***number***

Specifies the amount of time in seconds that the primary host waits for a connection to be made to a remote host. The default timeout to the first remote host connecting is 600.

### Description

This statement specifies the number of seconds that the local host will wait for a remote host to connect. If a remote host does not connect within the specified time, the local host attempts to connect to the remote host specified by the next **REMOTE HOST** statement. The timeout is reset to the full LAUNCH WAIT value (default is 600 seconds if unspecified) for the second remote host to connect, and so forth, for subsequent remote host connections.

### Examples

```
LAUNCH WAIT 120
```

# MASTER PREFERRED

Specifies a list of preferred nodes for a primary host.

## Usage

**MASTER PREFERRED** *list*

## Arguments

- *list*

Specifies an ordered list of space separated host names as the preferred nodes. The wildcard (\*) and (?) expressions are supported for matching multiple host names.

## Description

When this statement is used and a job launches, the preferred primary host is allocated if available. Host allocation starts from the left with the first available primary host in the list. If the specified primary hosts are not available, any available primary host is allocated.

## Examples

```
MASTER PREFERRED primary1* primary2* primary3000?
```

# MASTER RESOURCE

Specifies a resource requirement for a primary host.

## Usage

**MASTER RESOURCE** *expr*

## Arguments

- *expr*

Specifies a Tcl expression that represents a resource requirement. The following predefined variables can be used in this expression:

```
hostname
mem_free
mem_total
mem = mem_total - mem_free
swap_free
swap_total
swap = swap_total - swap_free
job_slot_total
job_slot_free
job_slot = job_slot_total - job_slot_free
```

The units used for memory and swap is MB.

## Description

This statement specifies the a resource requirement for a primary host. If the requirement is not met, then the job is not launched on the primary host. When specifying a hostname in the expression, it should be nested in parenthesis.

## Examples

```
MASTER RESOURCE ($job_slot > 1) && ($hostname != dummy01)
```

# REMOTE COUNT

Specifies an initial number of remotes.

## Usage

**REMOTE COUNT** *number*

## Arguments

- *number*

Specifies an initial number of remotes.

## Description

This statement specifies an initial number of remotes. The specified *number* is used by the COUNT argument in the [LAUNCH CLUSTER](#) statement. The value can be verified in the job’s “cluster” attribute, which is displayed in announce\_job and in the HTTP server job page. The value specified in this statement overrides the value specified in a [calcm\\_jobqueue\\_app.tcl](#) configuration.

## Examples

```
REMOTE COUNT 2
```

## REMOTE COUNT\_MAX

Specifies the maximum number of remotes in a Calibre MTflex configuration.

### Usage

**REMOTE COUNT\_MAX *number***

### Arguments

- *number*

Specifies the maximum number of remotes.

### Description

This statement specifies the maximum number of remotes that can be allocated for a job. The value can be verified in the job’s “cluster” attribute, which is displayed in announce\_job and in the HTTP server job page. The value specified in this statement overrides the value specified in a [calcm\\_jobqueue\\_app.tcl](#) configuration. The REMOTE COUNT\_MAX value can be overridden by [calcm\\_rmanager\\_app.tcl](#). If ADJUSTMINMAXOP is enabled in calcm\_rmanager\_app.tcl, the maximum number of remotes specified by the [REMOTE MINMAX](#) statement overrides the REMOTE COUNT\_MAX value during a specified job operation.

### Examples

```
REMOTE COUNT_MAX 200
```

# **REMOTE COUNT\_MIN**

Specifies the minimum number of remotes required to start a job in a Calibre MTflex configuration.

## Usage

**REMOTE COUNT\_MIN** *number*

## Arguments

- *number*

Specifies a minimum number of remotes.

## Description

This statement specifies the minimum number of remotes that are required to start a job. The specified number is used by the MINCOUNT argument in the [LAUNCH CLUSTER](#) statement.

When this statement is used, the minimum number of remote connections must be established before the wait time expires in order for the Calibre run to start. Therefore, it is recommended that you also use the [LAUNCH WAIT](#) statement to adjust the wait time as necessary.

## Examples

```
REMOTE COUNT_MIN 20
```

## REMOTE DATA

Specifies the number of remote data servers (RDS).

### Usage

**REMOTE DATA {*number* | AUTO}**

### Arguments

- ***number***

Specifies the number of remote data servers.

- **AUTO**

Specifies to launch remote data servers automatically in Calibre.

### Description

This statement specifies the number of RDS to launch or it specifies to launch RDS automatically in Calibre. By default, each RDS consumes up to 2 GB of memory including a backup server. The RDS should be matched with a backup in a different host. This ensures that the number of slots that can be used in one host cannot exceed half of the number of remote data servers. The limit can be removed by setting **REMOTE DATA\_RECOVEROFF** to “1” which disables the launch of the backup remote data server.

If REMOTE DATA AUTO is specified, the number of RDS launched in Calibre is based on the initial number of remote compute servers (RCS). The initial number of RCS are allocated in groups of four if possible to avoid allocating more RDS.

The REMOTE DATA AUTO option should not be specified with the **REMOTE DATA\_RESOURCE** statement.

Refer to “Remote Data Server” in the *Calibre Administrator’s Guide* for more information about RDS.

### Examples

```
REMOTE DATA 4
```

## REMOTE DATA\_MIN

Specifies the minimum number of remote data servers (RDS) to start a job.

### Usage

**REMOTE DATA\_MIN** *number*

### Arguments

- *number*

Specifies the minimum number of remote data servers.

### Description

This statement specifies the minimum number of remote data servers that are required to start a job. The job exits if the number of remote data servers attached to a job is less than the specified number.

### Examples

```
REMOTE DATA_MIN 4
```

## **REMOTE DATA\_RECOVEROFF**

Specifies whether to launch backup remote data servers.

### **Usage**

**REMOTE DATA\_RECOVEROFF** *flag*

### **Arguments**

- *flag*

Specifies the launching of backup remote data servers. Valid values include:

- 0 — Enables the launch of the backup remote data server. This is the default.
- 1 — Disables the launch of the backup remote data server.

### **Description**

This statement specifies whether or not to enable the launch of backup remote data servers. If disabled, a memory savings can result; however, data stored in the RDS cannot be restored if a problem is encountered in the remote host where the RDS is launched.

### **Examples**

```
REMOTE DATA_RECOVEROFF 1
```

# REMOTE DATA\_RESOURCE

Specifies a resource requirement for remote data server (RDS) nodes.

## Usage

**REMOTE DATA\_RESOURCE** *expr*

## Arguments

- *expr*

Specifies a Tcl expression that represents a resource requirement. The following predefined variables can be used in the expression. The unit of memory and swap is MB.

- down: host is down
- up: host is up
- closed: host is closed
- hostname
- mem free
- mem total
- mem = mem total - mem free
- swap free
- swap total
- swap = total - swap free

## Description

This statement and expression specify a resource requirement for RDS nodes. The node that matches the resource requirement is allocated as RDS remotes. When specifying a host name in the expression, you must nest it with parentheses and enclose its value in braces ({}).

The REMOTE DATA\_RESOURCE statement should not be specified with the [REMOTE DATA AUTO](#) statement option.

## Examples

This resource requirement for initial remote nodes specifies a swap space value of 0 MB and excludes hostname dummy01.

```
REMOTE DATA_RESOURCE ($swap == 0) && ($hostname != {dummy01})
```

## REMOTE ENV

Specifies an environment variable file name for Calibre remotes.

### Usage

**REMOTE ENV** *fname*

### Arguments

- *fname*

Specifies an environment variable file name.

### Description

This statement specifies the name of a file which contains the environment variables to be passed to the Calibre remotes. There are four syntax form possibilities that can be used to specify an environment variable and its value. The syntax allows a value to be specified with an environment variable if the dollar sign (\$) is escaped with a backslash (\) character.

```
setenv var value
export var=value
var:value
var:\$val
```

Refer to “[Job Environment File \(job.env\)](#)” for definitions of job-specific environment variables.

### Examples

REMOTE ENV test.env

# REMOTE INIT\_RESOURCE

Specifies a resource requirement for the initial remote nodes.

## Usage

**REMOTE INIT\_RESOURCE** *expr*

## Arguments

- *expr*

Specifies a Tcl expression that represents a resource requirement. The following predefined variables can be used in the expression. The unit of memory and swap is MB.

- down: host is down
- up: host is up
- closed: host is closed
- hostname
- mem free
- mem total
- mem = mem total - mem free
- swap free
- swap total
- swap = total - swap free

## Description

This statement and expression specify a resource requirement for initial remote nodes. The node that matches the resource requirement is allocated as initial remotes. When specifying a host name in the expression, you must nest it with parentheses and enclose its value in braces ({}).

## Examples

This resource requirement for initial remote nodes specifies a swap space value of 0 MB and excludes hostname dummy01.

```
REMOTE INIT_RESOURCE ($swap == 0) && ($hostname != {dummy01})
```

## REMOTE MAX

Specifies a maximum number of remotes.

### Usage

**REMOTE MAX *number***

### Arguments

- *number*

Specifies a maximum number of remotes.

### Description

This statement specifies a maximum number of remotes. The specified number is used by [calcm\\_rmanager\\_app.tcl](#) to limit the number of remotes. This is not a hard limit so the actual number of remotes can go up to the value specified by [REMOTE COUNT\\_MAX](#).

### Examples

```
REMOTE MAX 200
```

# REMOTE MAXCHANGE

Specifies the maximum number of remote resources that can be added to a job in a single CalCM cycle.

## Usage

**REMOTE MAXCHANGE num[,min[-max]]**

## Arguments

- **num**

Specifies how the maximum number of remote resources is determined for a job. Valid values include:

0 — The default value, which means no limit on the number of remotes supplied.

---

### Note

 In order to rapidly adapt resources to a job's CPU demand, it is *not* recommended to use the default value.

---

Any positive integer value — This forces CalCM to throttle the number of remotes supplied by the CalCM cycle to this value.

---

### Note

 In order to rapidly adapt resources to a job's CPU demand, it is *not* recommended to use this hard-coded value, which might be different from job to job.

---

-1 — The maximum number of remotes supplied is automatically determined. The CalCM daemon transcript shows the automatically computed value after the job start. For example:

```
[2019-06-03 21:11:41] MAXCHANGE of job "726204" is set to 300.
```

-2 — For a hyperthreaded job, specify this value instead of -1 to automatically determine the maximum number of remotes that can be supplied in a single CalCM cycle.

- **min**

The min option can be used to specify the minimum number of remotes to supply for a job when the MAXCHANGE value is automatically computed by CalCM (REMOTE MAXCHANGE using -1 or -2 values). The default minimum value is 1.

- **max**

The max option can be used to specify the maximum number of remotes to supply for a job when the MAXCHANGE value is automatically computed by CalCM (REMOTE MAXCHANGE using -1 or -2 values). The default maximum value is 600.

## Description

This statement specifies a maximum number of remote resources to supply for a job during a single CalCM cycle. You can use this statement to reduce the timeout caused by large numbers of remotes. Specific resource cases can be handled at the job-level by using REMOTE MAXCHANGE in the job configuration file (*job.conf*).

### Note

 Depending on the Calibre version (older than 2017.2), job type, hardware environment, and OS settings, a large number of remotes added to a Calibre job at once might not all connect successfully. The purpose of the REMOTE MAXCHANGE command is to limit the maximum number of remotes being added to a Calibre job per CalCM cycle thus helping to avoid such an issue.

---

## Examples

### Example 1

This example specifies that the maximum number of remotes to supply for a job is automatically determined within the default range of one remote minimum and 600 remotes maximum.

```
REMOTE MAXCHANGE -1
```

### Example 2

This example specifies for a hyperthreaded job that the maximum number of remotes to supply for a job is automatically determined within the range of 20 remotes minimum and 600 remotes maximum, the default.

```
REMOTE MAXCHANGE -2,20
```

### Example 3

This example specifies that the maximum number of remotes to supply for a job is automatically determined within the range of ten remotes minimum and 300 remotes maximum.

```
REMOTE MAXCHANGE -1,10-300
```

## REMOTE MIN

Specifies a minimum number of remotes.

### Usage

**REMOTE MIN** *number*

### Arguments

- *number*

Specifies a minimum number of remotes.

### Description

This statement specifies a minimum number of remotes. The specified number is used in [calcm\\_rmanager\\_app.tcl](#) to keep the minimum number of remotes. This value can be overridden by [calcm\\_rmanager\\_app.tcl](#). If ADJUSTMINMAXOP is enabled in [calcm\\_rmanager\\_app.tcl](#), the minimum number of remotes specified by the [\*\*REMOTE MINMAX\*\*](#) statement overrides the **REMOTE MIN** value during a specified job operation.

### Examples

```
REMOTE MIN 2
```

## REMOTE MINMAX

Specifies a minimum and maximum number of remote hosts for each operation.

### Usage

#### **REMOTE MINMAX *rules***

### Arguments

- ***rules***

Specifies a comma-separated list for allocating the number of minimum and maximum remote hosts within each job operation class.

If no keywords are specified, the default filter is defined by the job operation and class. For example,

```
job operation=min/max
```

where you can specify the following parameters:

- The wildcard (\*) and (?) characters to match the job operation.
- The wildcard (\*) to keep the existing *min* or *max* value(s) for the remote hosts.
- The minus sign (-) for *min* value to freeze the remote host allocation and revoke pending remote hosts during the specific job operation. The *max* value must be specified in this case, but it is not used.

---

#### **Note**

 In some cases, specifying “-” for *min* value may not freeze the remote host allocation and revoke the pending remote hosts due to event timing and non-real time information from the running job.

---

The following keywords can be used instead of filtering by the job operation and class:

**index *svrf\_index*** — The index number of the SVRF operation as reported in Calibre transcript. For example,

```
CPU TIME = 16 + 943  REAL TIME = 319 ... OPS COMPLETE = 42 OF 5057
```

The specification of index “0” can be used to define a REMOTE MINMAX setting for all run the stages occurring *before* the first SVRF operation. For example,

“STANDARD VERIFICATION RULE FILE COMPILATION MODULE”,  
“LICENSING MODULE”, “LAYOUT DATA INPUT MODULE”, “RESULTS DATABASE INITIALIZATION MODULE”.

Once the first SVRF operation starts, it runs inside the “EXECUTIVE MODULE” stage.

**index *start-end*** — The exact index range of the operations being executed. The keyword “last” can be used in *end* to specify the last index.

*name operation name* — The name of the operation being executed.

*class operation class name* — The class name of the operation that is being executed.

*stage module\_name* — The Calibre stage module name. If the stage is before the executive module, the operation index is 0, denoting the input module stage.

The following special command can be used to inherit the global configuration setting:

**@INHERIT** — Inherits the global setting specified in MINMAXOP in [calcm\\_rmanager\\_app.tcl](#).

## Description

This statement specifies a list of job operations or keywords along with the numbers of minimum and maximum remote hosts to allocate for each job operation. Matching for the job operation specification is sequential. The values for REMOTE MINMAX override the minimum and maximum values in **REMOTE MIN** and **REMOTE COUNT\_MAX** during the job operation. REMOTE MINMAX values are applied to the job operation only if **ADJUSTMINMAXOP** is enabled in [calcm\\_rmanager\\_app.tcl](#). For a specified job operation, the default values in **ADJUSTMINMAXOP** and the values in MINMAXOP are overridden by REMOTE MINMAX.

### Note

 When Hyperscaling mode is used and the HDB 0 job operation is null, the next non-null job operation (HDB 1-4) in the specification determines the adjustment of the remote host allocation.

## Examples

### Example 1

This example specifies the numbers of minimum and maximum remote hosts allocated for DRC and LITHO job operations respectively. The global settings are inherited from MINMAXOP in the calcm\_rmanager\_app.tcl application.

```
REMOTE MINMAX DRC:=256/256,LITHO:=256/1024,@INHERIT
```

### Example 2

This example specifies the numbers of minimum and maximum remote hosts allocated for job operation index range 1-10 and job operation class name OPCSBAR.

```
REMOTE MINMAX index(1-10)=256/256,class(OPCSBAR)=256/256
```

### Example 3

This example specifies the minimum and maximum remote hosts allocated for job operations with null and non-null job operations on HDBs under Hyperscaling.

```
REMOTE MINMAX DRC:=256/256,DFM:=512/512,LITHO*=1024/2048
```

Case 1: In this case, HDB 0 is null, and the next non-null job operation on HDB 1 determines the allocation.

```
HDB0  HDB1  HDB2  HDB3  HDB4
NULL  DRC   DRC   DRC   DFM
-> 256/256
```

Case 2: In this case, HDB 0 and HDB 1 are null, and the next non-null job operation on HDB 2 determines the allocation.

```
HDB 0  HDB 1  HDB 2  HDB 3  HDB 4
NULL    NULL   DRC     DRC     DFM
-> 256/256
```

Case 3: In this case, HDB 0 is null, and the next non-null job operation on HDB 1 determines the allocation.

```
HDB 0  HDB 1  HDB 2  HDB 3  HDB 4
NULL  DFM   DRC   DRC   DFM
-> 512/512
```

Case 4: In this case, there are no null job operations, and the first job operation on HDB 0 determines the allocation.

```
HDB 0  HDB 1  HDB 2  HDB 3  HDB 4
DRC   LITHO  DRC   DRC   DFM
-> 256/256
```

# REMOTE PREFERRED

Specifies a list of preferred nodes for a remote host.

## Usage

**REMOTE PREFERRED** *list*

## Arguments

- *list*

Specifies an ordered list of space separated host names as the preferred nodes. The wildcard (\*) and (?) expressions are supported for matching multiple host names.

## Description

When this statement is used and a remote is allocated, the preferred remote host is selected if available. Host allocation starts from the left with the first available remote host in the list. If the specified remote hosts are not available, any available remote host is allocated.

## Examples

```
REMOTE PREFERRED remote1* remote2* remote3000?
```

## REMOTE RAMPUPDOWN

Specifies a ramp up/down ratio for each operation to adjust the job's CPU demand.

### Usage

**REMOTE RAMPUPDOWN *rules***

### Arguments

- ***rules***

Specifies a comma-separated list of operations and their ramp up/down ratios used for adjusting job CPU demand within each job operation class. The job resource calculation is multiplied by the up/down ratios depending on the demand. In the case of positive demand, it is multiplied by the up ratio, and for negative demand, it is multiplied by the down ratio. These ratios are used to control (speed up or slow down) the addition or removal of resources, which can be useful when considering the operation scalability.

The format of the rules is shown,

*operation=up/down*

where you can specify the following parameters:

- The wildcard (\*) and (?) characters to match the job operation and class.
- The wildcard (\*) to keep the existing demand value.
- The minus sign (-) in *up* to freeze the remote host allocation during the specific job operation class.

---

#### Note

 In some cases, specifying “-” in *up* may not freeze the remote host allocation due to event timing and non-real time information from the running job.

---

The filters that can be specified in [REMOTE MINMAX](#) can be used in *operation* in REMOTE RAMPUPDOWN.

### Description

This statement specifies a list of job operation class and ramp up/down ratios. The specified ratio is used to adjust the CPU demand during the operation. The rule match is done sequentially and applied if a match exists. REMOTE RAMPUPDOWN values are applied to the operation only if ADJUSTRAMPUPDOWN is enabled in [calcm\\_rmanager\\_app.tcl](#). For a given job operation class, values specified in RAMPUPDOWN are overridden by REMOTE RAMPUPDOWN.

## Examples

### Example 1

This example specifies the up/down ratios used for calculating CPU demand for DRC and LITHO job operations respectively. The “DRC:” specification matches all DRC job operation classes, and the “LITHO:” specification matches all LITHO job operations. The global settings are inherited from RAMPUPDOWN in the calcm\_rmanager\_app.tcl application.

```
REMOTE RAMPUPDOWN DRC:*=*/0.5,LITHO*=1.5/*,@INHERIT
```

### Example 2

This example specifies the up/down ratios used for calculating CPU demand for job operation index range 1-10 and job operation class name OPCSBAR.

```
REMOTE RAMPUPDOWN index(1-10)=0.5/1.0,class(OPCSBAR)=2.0/*
```

# REMOTE RESOURCE

Specifies a resource requirement for remote hosts.

## Usage

**REMOTE RESOURCE** *expr*

## Arguments

- *expr*

Specifies a Tcl expression that represents a resource requirement. The following predefined variables can be used in this expression:

down: host is down  
up: host is up  
closed: host is closed  
hostname  
cluster: list of cluster names  
mem\_free  
mem\_total  
mem = mem\_total - mem\_free  
swap\_free  
swap\_total  
swap = swap\_total - swap\_free

The units used for memory and swap is MB.

## Description

This statement specifies a resource requirement for remote hosts that must be met in order to run the job. If the requirement is not met, the expression is evaluated as false and the remote hosts are revoked. The REMOTE RESOURCE *expr* can be used to pass a job specific resource requirement in launching remotes under platform tools. The *expr* is passed as \$jobAttr(resource) in the Calibre “REMOTE COMMAND.” An expression that includes a hostname must be nested in parentheses. The cluster can be specified in the cluster configuration (*cluster.conf* file). The is\_cluster proc can be used to match a cluster name.

## Examples

### Example 1

This example specifies a resource requirement that has a defined value for “\$swap” and requires remotes from a hostname that does not match “host01”.

```
REMOTE RESOURCE ($swap == 0) && ($hostname != host01)
```

### Example 2

This example specifies a resource requirement that requires remote hosts from hostnames starting with “host01” only.

```
REMOTE RESOURCE [regexp {^host01} $hostname]
```

### Example 3

This example specifies a resource requirement that requires remote hosts from the matched cluster “cluster1”.

```
REMOTE RESOURCE [is_cluster cluster1]
```



# Appendix A

## CalCM Patch Requirements

---

A list of the patch requirements is provided for running CalCM.

**Calibre Release and Patch Requirements for CalCM .....** [285](#)

## Calibre Release and Patch Requirements for CalCM

CalCM requires certain Calibre patch levels to support different modes and applications.

The bold highlighted versions are the recommended versions of the Calibre release.

---

### Note

---

 Calibre versions 2017.1 and above are fully compatible with CalCM usage. Starting with version 2017.2, the CPU ramp-up has improved performance. Starting with version 2018.1, Calibre jobs have improved hardware monitoring.

---

**Table A-1. Patch Requirements for CalCM**

Calibre Release	Dynamic CPU and Licensing	Hyperscaling	Hyperthreading
2008.4	<b>2008.4_61 (P21)</b>	2008.4_61 (P21)	Not Supported
2009.1	<b>2009.1_63 (P18)</b>	2009.1_63 (P18)	Not Supported
2009.2	<b>2009.2_61 (P27)</b>	2009.2_61 (P27)	Not Supported
2009.3	<b>2009.3_65 (P35)</b>	2009.3_65 (P35)	2009.3_65 (P35)
2009.4	<b>2009.4_72 (P37)</b>	2009.4_72 (P37)	2009.4_72 (P37)
2010.1	<b>2010.1_47 (P18)</b>	2010.1_47 (P18)	2010.1_47 (P18)
2010.2	<b>2010.2_59 (P21)</b>	2010.2_59 (P21)	2010.2_59 (P21)

**Table A-1. Patch Requirements for CalCM (cont.)**

Calibre Release	Dynamic CPU and Licensing	Hyperscaling	Hyperthreading
2010.3	<b>2010.3_63 (P28)</b>	2010.3_63 (P28)	2010.3_63 (P28)
2010.4	<b>2010.4_70 (P38)</b>	2010.4_70 (P38)	2010.4_70 (P38)
2011.1	<b>2011.1_59 (P23)</b>	2011.1_59 (P23)	2011.1_59 (P23)
2011.2	<b>2011.2_61 (P27)</b>	2011.2_61 (P27)	2011.2_61 (P27)
2011.3	<b>2011.3_67 (P32)</b>	2011.3_67 (P32)	2011.3_67 (P32)
2011.4	<b>2011.4_61 (P30)</b>	2011.4_61 (P30)	2011.4_61 (P30)
2012.1	<b>2012.1_57 (P23)</b>	2012.1_57 (P23)	2012.1_57 (P23)
2012.2	<b>2012.2_80 (P39)</b>	2012.2_80 (P39)	2012.2_80 (P39)
2012.3	<b>2012.3_55 (P27)</b>	2012.3_55 (P27)	2012.3_55 (P27)
2012.4	<b>2012.4_87 (P58)</b>	2012.4_87 (P58)	2012.4_87 (P58)
2013.1	<b>2013.1_58 (P26)</b>	2013.1_58 (P26)	2013.1_58 (P26)
2013.2	<b>2013.2_62 (P29)</b>	2013.2_62 (P29)	2013.2_62 (P29)
2013.3	<b>2013.3_75 (P36)</b>	2013.3_75 (P36)	2013.3_75 (P36)
2013.4	<b>2013.4_68 (P35)</b>	2013.4_68 (P35)	2013.4_68 (P35)
2014.1	<b>2014.1_78 (P41)</b>	2014.1_78 (P41)	2014.1_78 (P41)
2014.2	<b>2014.2_58 (P25)</b>	2014.2_58 (P25)	2014.2_58 (P25)
2014.3	<b>2014.3_67 (P33)</b>	2014.3_67 (P33)	2014.3_67 (P33)

**Table A-1. Patch Requirements for CalCM (cont.)**

Calibre Release	Dynamic CPU and Licensing	Hyperscaling	Hyperthreading
2014.4	<b>2014.4_77 (P41)</b>	2014.4_77 (P41)	2014.4_77 (P41)
2015.1	<b>2015.1_70 (P35)</b>	2015.1_70 (P35)	2015.1_70 (P35)
2015.2	<b>2015.2_51 (P20)</b>	2015.2_51 (P20)	2015.2_51 (P20)
2015.3	<b>2015.3_63 (P27)</b>	2015.3_63 (P27)	2015.3_63 (P27)
2015.4	<b>2015.4_55 (P25)</b>	2015.4_55 (P25)	2015.4_55 (P25)
2016.1	<b>2016.1_49 (P19)</b>	2016.1_49 (P19)	2016.1_49 (P19)
2016.2	<b>2016.2_55 (P19)</b>	2016.2_55 (P19)	2016.2_55 (P19)
2016.3	<b>2016.3_51 (P17)</b>	2016.3_51 (P17)	2016.3_51 (P17)
2016.4	<b>2016.4_43 (P5)</b>	2016.4_43 (P5)	2016.4_43 (P5)



# **Appendix B**

## **Example Configuration Files and Transcript**

---

This section contains example configuration files for CalCM and an example of the transcript output from the tool.

<b>CalCM Configuration File Example .....</b>	<b>289</b>
<b>Job Configuration File Example .....</b>	<b>291</b>
<b>CalCM Daemon Transcript Example .....</b>	<b>291</b>

## **CalCM Configuration File Example**

The CalCM configuration file is an ASCII file consisting of a series of statements that define options for the CalCM daemon. These options are applied to any Calibre job that is run using the associated instance of the CalCM daemon.

## Example Configuration Files and Transcript

### CalCM Configuration File Example

---

```
VARIABLE CALCM_HOME /home/jsmith/calcm
VARIABLE CALCM_APP $MGC_HOME/pkgs/icv_calcm/tcl

SERVER PORT 9905
SERVER INTERVAL 120
LICENSE PORT 9906
LICENSE FEATURES FILE [
    calmdpmerge = 300
    calfracturem = 300
    calfracturej = 300
    calfracturet = 300
    calfractureh = 300
    calopcsbar = 300
    calibredrc = 300
    calibrehdrc = 300
    calmdpesvrf = 300
    calfracturem = 300
    calmpcpro = 300
    calopcpro = 300
    calnmmmpc = 300
    calnmopc = 300
    caltdopc = 300
    calopcverify = 300
    calibrelfd = 300
]

APPLICATION TCL $CALCM_APP/calcm_jobqueue_app.tcl FILE [
    CLUSTERCONF = $CALCM_HOME/config/cluster.conf
    QUEUEDIR = $CALCM_HOME/jobqueue // Job queue directory
    SCANJOBS = 1
    CLUSTERMIN = 2
]

APPLICATION TCL $CALCM_APP/calcm_tat_app.tcl FILE [
    DBPATH = $CALCM_HOME/db/calcm_tat.db
    MAX_REMOTE = 1000
    ADJUST_BUDGET = true
    PRIORITY_BUMP = 20
]

APPLICATION TCL $CALCM_APP/calcm_rmanager_app.tcl FILE [
    QUOTADB = $CALCM_HOME/config/calcm_quota.conf
]

APPLICATION TCL $CALCM_APP/calcm_systemanalysis_app.tcl FILE [
    DBPATH = $CALCM_HOME/db/calcm_analysis.db
]

APPLICATION TCL $CALCM_APP/calcm_rmonitor_app.tcl FILE [
    DBPATH $CALCM_HOME/db/calcm_rmonitor.db
    PRUNEDBINTERVAL = 86400
]

APPLICATION TCL $CALCM_APP/calcm_http_server_app.tcl FILE [
    ROOT = $CALCM_HOME/public_html
    ENABLE_DASHBOARD = 1
    DASHBOARDPORT = 9902
]
```

## Job Configuration File Example

The job configuration file defines job-specific attributes, such as the job priority, the rule file for the job, and the budget for job operations. The job configuration file is also used to define job account information.

```
JOB TYPE MTflex
JOB PRIORITY 0
JOB RULE sim.drc
JOB LOG rules.%ID.log
JOB ENV job.env

JOB TAT_PRIORITY_BUMP 20

REMOTE MAX 500

JOB KEY v20xx.x

JOB USER DIV_TEX/username
JOB QUOTA DEPT2/FLOW1

// budget
JOB BUDGET dens_out=80,opcv_out=25,_EXTRA_=5
```

## CalCM Daemon Transcript Example

The CalCM daemon generates a transcript which is captured in a log file located at *\$CALCM\_HOME/calcmd.log*.

## Example Configuration Files and Transcript

### CalCM Daemon Transcript Example

---

```
// Calibre Cluster Manager v20xx.x_x.xxxx Thu Jan 17 15:18:20 PST 20xx
//
// Copyright Siemens 1996-2020
// All Rights Reserved.
// THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
// WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION
// OR ITS LICENSORS AND IS SUBJECT TO LICENSE TERMS.
//
// Mentor Graphics software executing under x86-64 Linux
//
// This software is subject to the terms of section 3 of the
// End User License Agreement or your signed agreement with
// Mentor Graphics Corporation, whichever applies.
//
// Your timebomb license will expire: Thu Apr 11 20xx
//
// CPU Info: Cores = 2
// Max file descriptors: 1024
// 64 bit virtual addressing enabled
// Running Mgc_home/pkgs/icv_calcm/pvt/calcmd -log calcmd_20xx_x.log
// Process ID: 27724
//
// Starting time: Thu Jan 17 15:59:59 20xx
//
// Cluster Manager running on 1 core
Locking "path/CALCM_HOME/config/calcmd.conf" on mpcpro in the process,
27724.
//
// Locking time: Thu Jan 17 16:00:01 20xx
//
Reading configuration from path/CALCM_HOME/config/calcmd.conf
VARIABLE CALCM_HOME path/CALCM_HOME
VARIABLE CALCM_APP $MGC_HOME/pkgs/icv_calcm/tcl
VARIABLE $CALCM_PLATFORM_INCLUDE $CALCM_HOME/config/calcm_tat.inc
SERVER PORT 9905
SERVER INTERVAL 120
LICENSE PORT 9906
LICENSE FEATURES FILE [
    calmdpmerge = 300
    calfracturem = 300
    calfracturej = 300
    calfracturet = 300
    calfractureh = 300
    calopcsbar = 300
    calibredrc = 300
    calibrehdrc = 300
    calmdpesvrf = 300
    calfracturem = 300
    calmpcpro = 300
    calopcpro = 300
    calnmmmpc = 300
    calnmopc = 300
    caltdopc = 300
    calopcverify = 300
    calibreelfd = 300
]
APPLICATION TCL $CALCM_APP/calcm_jobqueue_app.tcl FILE [
    CLUSTERCONF = $CALCM_HOME/config/cluster.conf
```

```

QUEUEDIR = $CALCM_HOME/jobqueue // Job queue directory
JOBDB = $CALCM_HOME/db/calcm_job.db
SCANJOBS = 1
CLUSTERMIN = 2
]
APPLICATION TCL $CALCM_APP/calcm_tat_app.tcl FILE [
    DBPATH = $CALCM_HOME/db/calcm_tat.db
    MAX_REMOTE = 1000
    ADJUST_BUDGET = true
    PRIORITY_BUMP = 20
]
APPLICATION TCL $CALCM_APP/calcm_rmanager_app.tcl FILE [
    QUOTADB = $CALCM_HOME/config/calcm_quota.conf
]
APPLICATION TCL $CALCM_APP/calcm_systemanalysis_app.tcl FILE [
    DBPATH = $CALCM_HOME/db/calcm_analysis.db
]
APPLICATION TCL $CALCM_APP/calcm_rmonitor_app.tcl FILE [
    DBPATH $CALCM_HOME/db/calcm_rmonitor.db
    PRUNEDBINTERVAL = 86400
]
APPLICATION TCL $CALCM_APP/calcm_http_server_app.tcl FILE [
    ROOT = $CALCM_HOME/public_html
    ENABLE_DASHBOARD = 1
    DASHBOARDPORT = 9902
]

RMonitor: checking database time range ...
Rmonitor: database time range 0 - 0
// Calibre License Broker v20xx.x_x.xxxx Thu Jan 17 15:19:47 PST 20xx
//
// Copyright Siemens 1996-2020
// All Rights Reserved.
// THIS WORK CONTAINS TRADE SECRET AND PROPRIETARY INFORMATION
// WHICH IS THE PROPERTY OF MENTOR GRAPHICS CORPORATION
// OR ITS LICENSORS AND IS SUBJECT TO LICENSE TERMS.
//
// Mentor Graphics software executing under x86-64 Linux
//
// This software is subject to the terms of section 3 of the
// End User License Agreement or your signed agreement with
// Mentor Graphics Corporation, whichever applies.
//

(LBD) [L] LBD listening on port 51694...
// DEBUG: Connecting to LBD mpcpro:51694
DEBUG: (LBD) [L] Connection accepted from 134.86.184.130 [1]
DEBUG: (LBD) [T2] Connection servicer thread starting.
(LBD) [T2] Connection established from 134.86.184.130
(LBD) [T2.C27724] Cluster 27724 registered. Now acquiring license pool...
DEBUG: (LBD) Acquiring 300 calmdpmerge license(s) (queueing OFF)...
(LBD) 300 calmdpmerge (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 600 calfracturem license(s) (queueing OFF)...
(LBD) 600 calfracturem (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calfracturej license(s) (queueing OFF)...
(LBD) 300 calfracturej (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calfracturet license(s) (queueing OFF)...
(LBD) 300 calfracturet (2016.020) licenses acquired.

```

## Example Configuration Files and Transcript

### CalCM Daemon Transcript Example

---

```
DEBUG: (LBD) Acquiring 300 calfractureh license(s) (queueing OFF)...
(LBD) 300 calfractureh (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calopcsbar license(s) (queueing OFF)...
(LBD) 300 calopcsbar (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calibredrc license(s) (queueing OFF)...
(LBD) 300 calibredrc (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calibrehdrc license(s) (queueing OFF)...
(LBD) 300 calibrehdrc (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calmdpesvrf license(s) (queueing OFF)...
(LBD) 300 calmdpesvrf (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calmpcpro license(s) (queueing OFF)...
(LBD) 300 calmpcpro (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calopcpro license(s) (queueing OFF)...
(LBD) 300 calopcpro (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calnmmpc license(s) (queueing OFF)...
(LBD) 300 calnmmpc (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calnmopc license(s) (queueing OFF)...
(LBD) 300 calnmopc (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 caltdopc license(s) (queueing OFF)...
(LBD) 300 caltdopc (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calopcverify license(s) (queueing OFF)...
(LBD) 300 calopcverify (2016.020) licenses acquired.
DEBUG: (LBD) Acquiring 300 calibrelfd license(s) (queueing OFF)...
(LBD) 300 calibrelfd (2016.020) licenses acquired.
// LBD configuration server listening on port 9906...
Server is listening at mpcpro:9905
Reading cluster configuration from path/CALCM_HOME/config/cluster.conf
MASTER HOST host00 JOB 5
REMOTE HOST host01 CPU 8
REMOTE HOST host02 CPU 8
REMOTE HOST host03 CPU 8
REMOTE HOST host04 CPU 8
REMOTE HOST host05 CPU 8
REMOTE HOST host06 CPU 8
REMOTE HOST host07 CPU 8
REMOTE HOST host08 CPU 8
REMOTE HOST host09 CPU 8
REMOTE HOST host10 CPU 8
REMOTE HOST host11 CPU 8
REMOTE HOST host12 CPU 8

Registering node host00
Registering node host01 with 8 CPUs
Registering node host02 with 8 CPUs
Registering node host03 with 8 CPUs
Registering node host04 with 8 CPUs
Registering node host05 with 8 CPUs
Registering node host06 with 8 CPUs
Registering node host07 with 8 CPUs
Registering node host08 with 8 CPUs
Registering node host09 with 8 CPUs
Registering node host10 with 8 CPUs
Registering node host11 with 8 CPUs
Registering node host12 with 8 CPUs
TAT INFO: DATABASE QUERY KEYS: OPE KEY MODE
TAT INFO: SCALABLE OPERATIONS: LITHO (1)
TAT INFO: database file: path/CALCM_HOME/db/calcm_tat.db
TAT INFO: Registered rules: 0
```

```
Reading quota configuration from path/CALCM_HOME/config/calcm_quota.conf
[TOP]
D1      8 1
D2      2 1

[D1]
P1      3 1
P2      1 1

[D2]
P3      1 1
P4      4 1
Reading Quota(0): TOP
Reading Quota(1): TOP/D1 TOP/D2
Running 6 applications:
  Job Queue v2.0_1.189.6.2
    Application Library v1.204.4.2
    Common Library v1.147
  TAT Application v1.0_1.88
    Application Library v1.26
    Common Library v1.147
  Resource Manager v1.2_1.82
    Application Library v1.116.4.1
    Quota Manager Library v1.3
    Common Library v1.147
  System Analysis Database v1.1_1.44
  Resource Monitor v1.0_1.46
  Web Server v1.3_1.238.4.1
// DEBUG: Applying licensing policy...
// DEBUG: Cluster Manager desired for 98 cores
// DEBUG: Licensing first thread...
// DEBUG: Licensing additional threads...
// DEBUG: Rule r_calcm matches.  Running...
// DEBUG: Acquiring 25 calresourceman license(s) (queueing OFF)...
// 25 calresourceman licenses acquired.
// DEBUG: License(s) consumed successfully (Qty 25).  Satisfying
features.
// DEBUG: Additional license for Cluster Manager acquired. (Cores
desired=98; authorized=100)
// DEBUG: Cluster Manager is over licensed.  It's authorized for 100
core(s), but only 98 are needed.

// Cluster Manager running on 98 cores
--- JOB UPDATE (Thu Jan 17 16:00:13 20xx) ---
Number of Jobs: 0
--- APP UPDATE (Thu Jan 17 16:00:13 20xx) ---
TAT TOTAL CPU DEMAND: 0 (0 30:0 60:0 180:0)
Jobs and remotes: 0/5/5 0/96 x4
Quota(cpu):
--- END UPDATE -----
--- JOB UPDATE (Thu Jan 17 16:02:13 20xx) ---
Number of Jobs: 0
--- APP UPDATE (Thu Jan 17 16:02:13 20xx) ---
TAT TOTAL CPU DEMAND: 0 (0 30:0 60:0 180:0)
Jobs and remotes: 0/5/5 0/96 x4
--- END UPDATE -----
Unlocking...
Exiting.
```



# Appendix C

## CalCM Migration Best Practices

---

This section provides a best practice for migrating, or porting, the CalCM daemon from one Calibre version to another.

<b>Introduction</b> .....	<b>297</b>
<b>CalCM Users</b> .....	<b>297</b>
<b>Overview of Migration Best Practices</b> .....	<b>298</b>
<b>Migration Tasks</b> .....	<b>300</b>
Preparing for CalCM Daemon Migration.....	300
Starting Up and Qualifying the New Version of CalCM .....	302
Migrating CalCM .....	304

## Introduction

When running CalCM in a production environment, system maintenance and upgrade events are commonplace. In some instances (for example, when you want to acquire newer capabilities of CalCM associated with a different version of Calibre), the CalCM daemon must be shut down and restarted. Keep in mind that in a production environment, this process must be a completely transparent operation. In other words, any system restart should have zero negative impact on any of the active or queued Calibre jobs, nor should it impede any production CalCM functionality - data, current or historical, should not be lost. The act of porting the CalCM daemon from one Calibre version to another is referred to as the migration process.

The purpose of this section is to walk CalCM administrators through a step-by-step CalCM migration procedure and provide a common set of best practices. In doing so, it aims to mitigate the risk factor associated with CalCM updates and maintenance within an active, production environment.

## CalCM Users

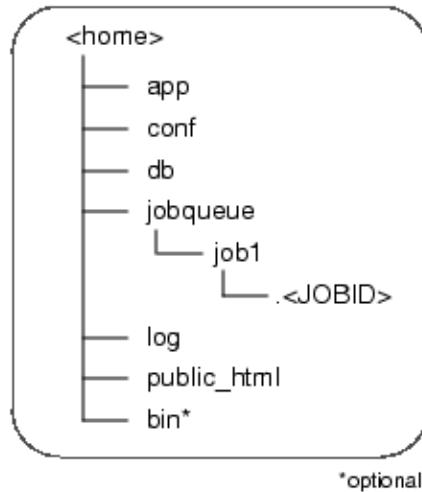
Typical users of CalCM fall into one of three primary categories: Information Technology Users, Production Flow Administrators, or Production Flow Users.

This best practice is written for Production Flow Administrators or the individuals who are responsible for governing and maintaining the CalCM system and the Calibre jobs within the production environment. It is desirable that these users are intermediate or advanced users of CalCM and Calibre.

Additional Comments:

- This migration process was validated against Calibre 2011.2 and 2011.3 production released versions on Linux RHEL5.4.
- Under the conditions, procedures, and examples stated within this document, forwards (from an older to a newer version) and backwards (from a newer to an older version) migration event procedures are claimed to be identical in practice.
- The code samples and command line entries in this section are real, working samples. Due to differences in user environments, the entries may not work verbatim; however, the concepts and practices should be applicable for all CalCM customers.
- This document assumes the recommended CalCM directory structure as described in the “[Configuring the CalCM Environment](#)”. This directory houses the files of interest for the migration.

**Figure C-1. Recommended Directory Structure for CalCM**



## Overview of Migration Best Practices

When the Calibre Cluster Manager (CalCM) daemon is running in a production environment, it typically has one or more Calibre jobs that it is constantly in communication with. At the same time, the jobs are also in constant communication with the License Broker Daemon (LBD) in order to properly serve up and retract licenses from jobs with positive or negative demand. On top of these two items, CalCM is also constantly collecting data and storing it within the application-specific databases in conjunction with keeping a transcription of all of the CalCM daemon-visible activities.

Upon migrating the CalCM daemon, the user will halt all of the interactions mentioned previously for a brief duration of time - depending on the cluster size, this process takes five seconds in addition to the time it takes to check out *calresourceman* licenses for the entire cluster - with the exception of the Calibre jobs which can continue processing. During this

process, both the new daemon and the new backup daemon will undergo their internal startup procedures. Shortly afterwards, the primary CalCM daemon reconnects back to the currently running jobs while the backup daemon listens quietly in the background.

When you follow these practices, you should be able to perform the migration process successfully while mitigating the risk factor associated with migrating the CalCM daemon within a production environment.

# Migration Tasks

---

The section includes the tasks that are required to successfully migrate the CalCM Daemon to a new release of Calibre. These tasks should be performed in the order listed.

<b>Preparing for CalCM Daemon Migration.....</b>	<b>300</b>
<b>Starting Up and Qualifying the New Version of CalCM .....</b>	<b>302</b>
<b>Migrating CalCM .....</b>	<b>304</b>

## Preparing for CalCM Daemon Migration

There exists a handful of actions that you must perform prior to migrating the CalCM daemon to a different Calibre version.

### Procedure

1. Begin by making an official request for a non-production cluster for CalCM migration testing.
  - a. Request an isolated, exclusively reserved cluster ( $\geq 2$  primary,  $\geq 200$  remotes).
    - i. Hardware and network specifications should be comparable to the production CalCM environment.

---

#### Note

 Be sure that no other users are allowed to use any of the remotes or primaries in parallel while the migration process is underway.

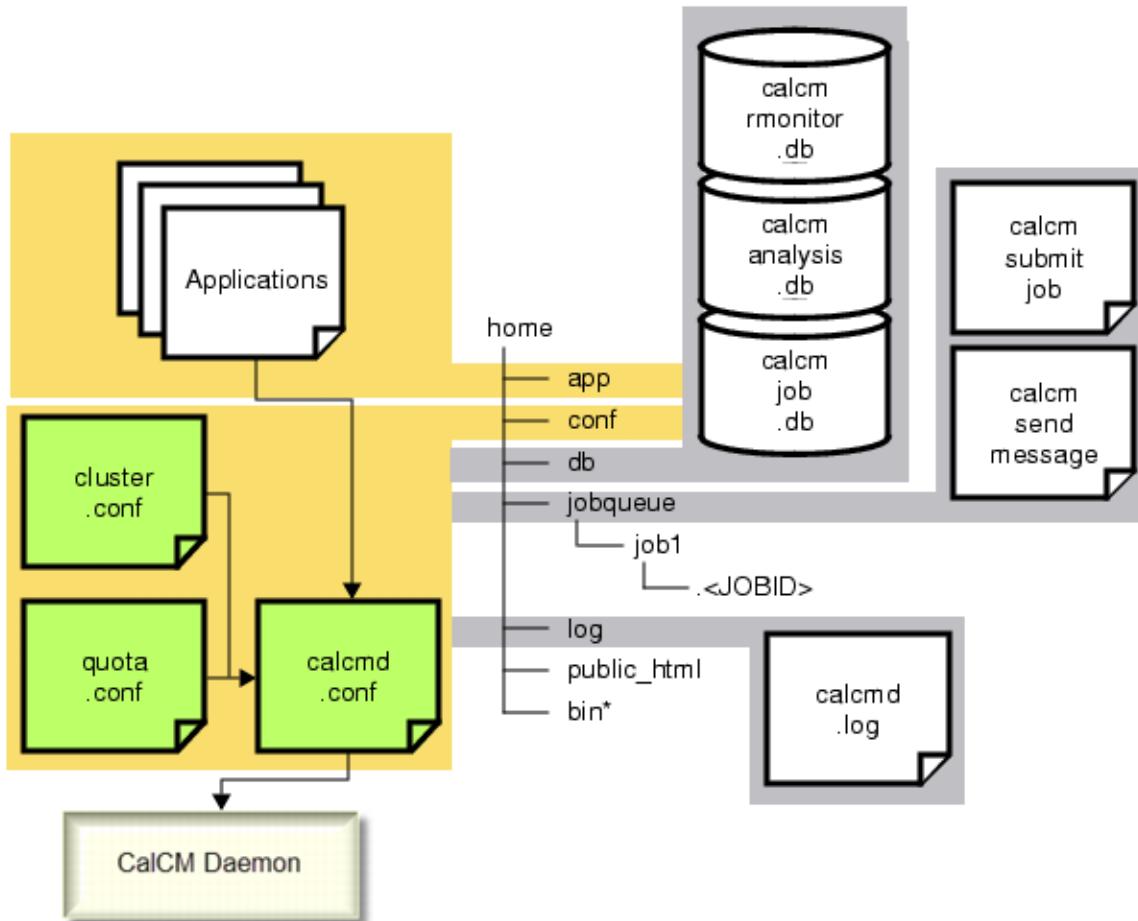
---

- b. Recommend cluster reservation (2 weeks).
  - i. Time frame can vary and may need to be negotiated for more or less time.
  - ii. Time dependencies for the migration are typically a direct function of the acceptance criteria and testing time.
  - iii. With an experienced CalCM user and no testing, the migration process could be scripted and performed in under an hour.
2. Make a copy of the production CALCM\_HOME directory to a temporary location for testing. For example:

```
% setenv CALCM_HOME ./home  
% mkdir -p ./tmp  
% cp -r $CALCM_HOME ./tmp
```

- a. Verify the temporary location contains all of the necessary files (for example, databases, applications, configuration files, and log files) and the directory structure that CalCM requires during startup as shown in [Figure C-2](#).

**Figure C-2. CalCM Files Needed for the Migration**



- b. CALCM\_HOME should originally point to the directory from which the production CalCM daemon was initially launched. This directory contains subdirectories like *app*, *jobqueue*, and *public\_html*.

**Caution**

 Be sure to copy and not move the directory as this will cause the production CalCM daemon to stop functioning correctly.

3. Unpack and install the new Calibre version for which the CalCM daemon is slated to be started with.
4. Once Calibre is available, copy the release applications from the new MGC\_HOME tree into the *app* directory.

---

**Note**

-  If non-released applications exist, remove them from the *app* directory first, before performing this step. After the release applications have been copied into the *app* directory, then move back the non-release applications.
- 

```
% rm $CALCM_HOME/app/* // removes the old copies of applications  
% cp $MGC_HOME/pkgs/icv_calcm/tcl/* $CALCM_HOME/app/.
```

---

**Note**

-  This directive assumes that the *calcmd.conf* file contains the absolute paths to the local application files in the application directory (*./app*) which are passed as the primary argument to the APPLICATION TCL statements.
- 

5. Make any changes to the configuration files as required by the new CalCM version.
  - a. For some releases, new release applications may become mandatory to use. In doing so, the *./conf/calcmd.conf* file should be updated to reflect the necessary changes.
  - b. For some releases, applications may be updated such that they have a new option or may require a new argument to be provided. In doing so, the *./conf/calcmd.conf* file should be updated to reflect the necessary changes.
6. Collect three or more test cases that can be used for qualification of CalCM overtime (not just for testing a single CalCM release). Criteria for the tests:
  - Each Calibre job must be run in Calibre MTflex mode (estimated time to completion of  $\geq 1$  hour).
  - Preferred Calibre products to test are Calibre nmOPC, Calibre OPCVerify, and Calibre MDP.
  - Always run on the same client Calibre version.
  - Note: Except for the first time running the test, do not use the CalCM daemon version of Calibre as overtime, as it will always be newer than the test cases.
  - Optional. Run one test with Hyperscaling.

## Starting Up and Qualifying the New Version of CalCM

This procedure makes sure that CalCM is behaving as expected when running within the non-production environment using the new CalCM version.

### Procedure

1. Start the CalCM daemon on a system (that is not designated as either a primary or a remote) using the new Calibre version with the new applications and the existing databases.

```
$MGC_HOME/bin/calcmd -config conf/calcmd.conf -log log/  
calcmd.log &
```

2. Monitor the CalCM daemon log file (*./log/calcmd.log*) and verify that the daemon has started up properly and encountered no issues.

```
% sleep 20s  
% egrep -i '(err|warn)' log/calcmd.log  
% grep "Cluster Manager running"
```

- a. If both the egrep statement returns nothing and the grep statement returns a line match, then the CalCM daemon has started up correctly and will be in listening mode until the first job is submitted.

This also implies that the applications, their setups, and the databases are compatible with the new CalCM version.

- b. If either fails to meet the requirements stated in the previous step (2a), then a problem has occurred and an assessment needs to be performed to determine the root cause and fix the issue.
  - c. Once the issues have been resolved, then repeat steps 1 and 2 in this section before proceeding.
3. Run qualification tests to make sure that CalCM can queue, dispatch, and monitor Calibre jobs as expected.

```
% ./jobqueue/calcm_submit_job `pwd`/jobqueue/job1/job.conf
```

- a. Check the CalCM log file (*./log/calcmd.log*) for the following messages in the prescribed order to know that the job was queued, announced, launched, monitored, and had completed.
  - i. '... successfully'
  - ii. 'Job "<jobID>" (<primary>:<port>) is announced.'
  - iii. 'Number of Jobs: 1'

This is assuming that only 1 job was started and that CalCM is not aware of any other job in progress.

- iv. '<primary>:<pid>: forget\_job <jobID>' 'Job "<jobID>" terminated'

Another option is to wait a few minutes and then look in the testing directory at the log file for the Calibre job.

- b. Once each job has completed, look at the testing results.
  - i. Compare the run times in the log file for the new job with the run time in the log file for the baseline job (wall clock time).
    - a. If they are within an acceptable small percentage, then the test can pass.

- b. If the results fall outside of an acceptable range, it can be directly attributed to CalCM, the network, or the hardware.
- ii. Compare the output results in the log file for the new job with the output results in the log file for the baseline job.  
If run on the same Calibre version, the output results should always be XOR clean when compared to the baseline results.
- iii. Compare the CPU profiles.

Use the http server application ([calcm\\_http\\_server\\_app.tcl](#)) to look at the CPU profile for the job in order to assess whether or not the newer version of CalCM is working comparably, better, or worse than before.

This process is a little tricky in the sense that the HTTP SERVER no longer has the job profile on display when it is done, yet can be very helpful for a user who is comfortable with CalCM behavior on the baseline jobs. In order to get this information, you can run the following command to extract the profile information out of the transcript from the CalCM daemon.

```
% egrep '^Job "<jobID>"' log/calcmd.log > <jobID>.prf
```

- c. If all qualification jobs run to completion within the expected time-to-complete, the CalCM daemon is not reporting any issues, all jobs have been monitored with no flags raised, and all apps are behaving as expected, then the compatibility tests pass and the act of migrating the daemon can now commence.

## Migrating CalCM

Once CalCM has been qualified with the new version of Calibre, you can now perform the actual process of migrating CalCM.

### Procedure

1. Kill the CalCM daemons.
  - a. Start by killing the “backup” CalCM daemon first.

```
% kill -HUP <pid(calcm_daemon)>
```

---

#### Note

 If not killed first, it can lead to issues such as a corrupted job database, jobs unable to reconnect to CalCM, and licenses hanging.

---

Alternative command to kill the CalCM daemon:

```
% killall -HUP calcmd
```

**Note**

 Do not use the ‘-9’ switch when killing any process with CalCM. This can cause processes such as the LBD to not shut down properly.

---

- b. Next, kill the production CalCM daemon.

```
% kill -HUP <pid(calcm_backup_daemon)>
```

**Note**

 If production runs are in process, shutting down the daemon should have little or no impact on the jobs. However, the dynamic CPU allocation ceases to occur and jobs continue to completion with their current number of cores. The startup procedure for CalCM attempts to reconnect to these jobs automatically.

---

2. Copy over the applications to the production directory and make a backup of the production databases.

```
% cp ./tmp/app/* $CALCM_HOME/app  
% mkdir ./db/bak.<timestamp>  
% cp $CALCM_HOME/db/* ./db/bak.<timestamp>
```

3. Verify that MGC\_HOME is pointing to the correct version of Calibre.

4. Start a production and backup CalCM daemon.

- a. Start the new production-ready, tested version of the CalCM daemon.

```
% cd $CALCM_HOME  
% $MGC_HOME/bin/calcmd -config conf/calcmd.conf  
-log log/calcmd.log &
```

- b. Start a backup daemon.

**Note**

 It is recommended to start the backup daemon on a different system than the one the primary daemon is running on. This will avoid a system crash resulting in no CalCM daemon running at all.

---

```
% $MGC_HOME/bin/calcmd -config conf/calcmd.conf  
-log log/calcmd.log &
```

5. If it is necessary to shut down the production and backup daemons, you should shut down the backup daemon first. The backup daemon is the one with the latest timestamp.

- a. Follow the monitoring procedures in “[Starting Up and Qualifying the New Version of CalCM](#)” to make sure the daemon is launched successfully.
- b. Monitor the *calcmd.log* file to look for the announcements from production running jobs coming back to CalCM.

This assumes that SCANJOBS = 1 and PRUNEJOBS were set within the CalCM configuration file.

- c. Once all jobs have reported back in, then the migration task has been completed.

# Appendix D

## Troubleshooting CalCM

---

Information on troubleshooting problems with CalCM is provided.

**CalCM Test Environment Set Up .....** 307

## CalCM Test Environment Set Up

Setting up a test environment helps you to determine if there is a difference between the production and test environment. In addition, this setup can be used for other testing purposes.

1. Follow the procedure described in “[Configuring the CalCM Environment](#)” and “[Starting the CalCM Daemon](#)” to set up a CalCM test environment.
2. Follow the procedures described in “[Configuring Calibre Jobs to Run Under CalCM](#)” and “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” to launch jobs in the CalCM test environment.

After running the jobs in the test environment, collect the following information for reporting.

[CalCM]

- calcmd.conf
- cluster.conf
- calcmd.log
- calcm\_\*.db
- \*.conf
- calcm\_submit\_job
- calcm\_send\_message

[Job]

- job.conf
- job.env
- rules.log

[*job ID*]

- calcm\_run

- calcm\_mtflex.conf
- calcm\_announce.tcl
- calcm\_launch.tcl
- Calibre\*Log.\*

# Appendix E

## Developing CalCM Applications

---

This section provides information on using Tcl to develop custom CalCM applications.

Tcl Basics .....	309
CalCM Application Development Overview.....	310
Creating an Activity Log Application .....	318
Setting Up a Notification Application .....	322

## Tcl Basics

This section provides fundamental information for using the Tcl programming language.

- Tcl is case sensitive.  
The core set of Tcl commands are all lower case.
- Order matters.  
Statements are executed sequentially within a procedure.
- Many characters have special meanings in Tcl.

The following are the special characters used in this chapter:

**Table E-1. Tcl Special Characters**

Character	Meaning
;	The semicolon terminates the previous command, allowing you to place more than one command on the same line.
\	Causes backslash substitution. Normally this is used to remove the special meaning of a metacharacter. Be careful not to have whitespace on the same line after a backslash that terminates a line, as this can cause Tcl interpretation errors.
\n	The backslash with the letter “n” creates a new line.
\$	The dollar sign in front of a variable name instructs the Tcl interpreter to access the value stored in the variable.
[ ]	Brackets cause command substitution, instructing the Tcl interpreter to treat everything within the brackets as the result of a command. Command substitution can be nested within words.

**Table E-1. Tcl Special Characters (cont.)**

Character	Meaning
{ }	Braces instruct the Tcl interpreter to treat the enclosed words as a single string. The Tcl interpreter accepts the string as is, without performing any variable substitution or evaluation.
“ ”	Quotes instruct the Tcl interpreter to treat the enclosed words as a single string. However, when the Tcl interpreter encounters variables or commands within string in quotes, it evaluates the variables and commands to generate a string.

- Comments are a type of command.

In Tcl, comments are indicated by “#”. It must be the first non-whitespace character of the command; nothing after it on the line is acted upon. However, it is parsed.

Because comments are parsed, they must be syntactically correct. For example, the comment below gives an error because it causes the braces to be unbalanced:

```
# if (some condition) {  
if { new text condition } {  
...  
}
```

Also, the # must be the first character of the command. That means that the first line in the following example is valid, because the comment character is preceded by a semicolon, but the third line is not, because the second line ended with a backslash.

```
} ;# End check_device switch  
perc::check_net -condition calc_adjacent_path \  
# "Net with MOS devices connected to different PAD paths"
```

## CalCM Application Development Overview

This section describes the process of developing an application for CalCM by analyzing a sample “Activity Log” Tcl application (*calcm\_actlog\_app.tcl*). This application is provided in an encrypted format in the software tree. The Activity Log application dumps snapshots of a cluster state into a log file, which can then be used to analyze cluster activity over a period of time.

When you invoke CalCM, the application reads the configuration file which includes the following APPLICATION TCL statement, used to initialize the *calcm\_actlog\_app.tcl* application:

```
APPLICATION TCL $CALCM_HOME/calcm_actlog_app.tcl FILE [  
    LOGPATH = My_PATH  
    USERNAME = Me  
    JOBQUEUE = This_directory  
]
```

The *calcml\_actlog\_app.tcl* file includes “proc” statements that perform the following functions:

- **Application Entry Point** — The “proc `nemo::application`” statement defines the entry point for CalCM applications.
- **Application Initialization** — The “proc `configureApp`” statement parses the application section of the configuration file.
- **Status Update** — The “proc `updateLog`” statement creates a snapshot of the cluster state and updates the information in the log file. A snapshot is a collection of consistent information about the state of a cluster at a specific moment in time.
- **Message Processing** — The “proc `processMessage`” statement processes information received by CalCM and displays the information in a web browser.

## Application Entry Point

Calibre Cluster Manager uses “events” to notify hosted applications about conditions that occur during the execution. Events are processed by application-defined callback routines that implement the desired behavior.

Each event that occurs during CalCM execution is broadcast to all applications, which can either react to the event or ignore it. In general, applications should ignore an event if they cannot “understand” its meaning; this provides for backward compatibility. The order in which a particular event is delivered to its recipients is unspecified and may vary even from one event instance to another.

The following types of events exist:

- Configuration — This event allows applications to set various options by interpreting respective sections of the CalCM configuration file. It occurs once when the application is loaded.
- Startup — This event occurs once during CalCM startup after all applications have been configured and indicates that the host has initialized successfully and applications can begin to perform their function.
- Shutdown — This event occurs once during CalCM shutdown before the application is unloaded from memory. It allows applications to perform a graceful cleanup.
- Status Update — This is a periodic event that occurs every time information about the global cluster state is updated. It allows applications to detect important changes in cluster state and react accordingly.
- Message — This event occurs when CalCM receives a message from an external source. It allows applications to implement remote communication and control via an application-defined text-based protocol.

The following Tcl application implements a reaction to events by processing a series of subcommands. The `nemo::application` procedure is the callback interface for event handling. Comments are used to describe the event reaction.

### Figure E-1. Defining Event Subcommands in the `nemo::application` Statement

```
proc nemo::application {subCmd args} {
    switch $subCmd {
        // Parse the configuration section and read path to log file
        configure { configureApp [lindex $args 0] }
        // Return the application name "Activity Log".
        name { return "Activity Log" }
        // Open and initialize the log file.
        attach { openLogFile }
        // Close the log file.
        detach { closeLogFile }
        // Write a new entry to the log file.
        update_status { updateLog [lindex $args 0] }
        // Handle HTTP request from a web browser and reply with a HTML page
        // containing the current log file size and a link to its contents.
        process_message { processMessage [lindex $args 0] }
    }
}
```

## Application Initialization

The `configure` subcommand allows an application to initialize itself by parsing the corresponding section of a configuration file, which is passed as an argument. The syntax of the configuration section is defined by the application. The following CalCM configuration file configures the “Activity Log” and associated applications:

```
APPLICATION TCL $CALBR_APPS/calcm/calcm_actlog_app.tcl FILE [
    LOGPATH=/home/calcm/calcmlog.txt
]
```

The “name” subcommand is used by CalCM to obtain the human-readable name of the application. The returned string is used for display and identification purposes only and does not necessarily have to be unique, although the use of clearly distinguishing application names is certainly encouraged. If the application does not process the name event (in other words, if the result of the query is empty), the path to the application script is used by CalCM.

The `attach` and `detach` subcommands represent the “Startup” and “Shutdown” events. The difference between `configure` and `attach` is that the latter is only invoked after CalCM has successfully started and is always complemented by `detach`.

## Figure E-2. Application Initialization

```

proc configureApp {configText} {
    global logFilePath
    foreach line [split $configText "\n"] {
        regexp {^\s*LOGPATH\s*=\s*(\S+)} $line -> logFilePath
    }
}

proc openLogFile {} {
    global logFilePath fileId
    set fileId [open $logFilePath "w"]
    puts $fileId "JOBS,Running jobs,MTflex,Hyperscaling"
    puts $fileId "CPUS,CPU usage,Total,Active"
}

proc closeLogFile {} {
    global fileId
    close $fileId
}

```

## Status Update

The cluster state is represented in the form of a “snapshot,” which is a collection of consistent information about the cluster state at a certain moment in time. A cluster snapshot contains objects of different types that form a hierarchy. The most general type is named “Object,” where “Job,” “Process,” and “Node” are three different kinds of objects. There is a specific type of “Process” named “Calibre”, which, in turn, has two different types named “Calibre Master” (primary process) and “Calibre Remote” (remote process). There are additional object types that represent miscellaneous system information, such as “Interface” (network interface) and “Filesystem” (file system). Refer to the “[CalCM Nemo API Reference Dictionary](#)” for more information.

The taxonomy of a snapshot embodies similarities between different types, or classes, of objects that can be present in a cluster snapshot. These similarities are reflected in the commands of the Nemo API where options for some commands are also available in commands that correspond to specific object types. For example, the options available in the [nemo::object](#) command can also be found in [nemo::job](#) and [nemo::process](#). For example, the following two lines of code are equivalent:

```

set jobOwner [nemo::job attr $jobObj "owner"]
set jobOwner [nemo::object attr $jobObj "owner"]

```

All commands in the Nemo API follow this pattern. For instance, options available in [nemo::process](#) are also available in [nemo::master](#), [nemo::remote](#), and [nemo::calibre](#). The choice of which command name to use in each particular situation obviously depends on whether a concrete object type is known, but factors like readability and clarity of code must also be taken into account.

A reference to the most current snapshot is passed as an argument to the update status subcommand. The snapshot is guaranteed to remain unchanged and consistent as long as at least one reference to it exists in one of the running applications. The only allowed usage of the snapshot reference is as an argument to the `nemo::snapshot` procedure.

Each cluster snapshot contains a list of active Calibre jobs (`nemo::snapshot` jobs) and available nodes (`nemo::snapshot` nodes). Each Calibre job has a one-to-many association with Calibre primary processes (`nemo::job` masters) that represent Hyperscaling HDBs (in case of Calibre MTflex, there is a single HDB0). Each Calibre primary has a one-to-many association with Calibre remote processes connected to the primary (`nemo::master` remotes). On the other hand, each cluster node has a one-to-many association with processes that are currently running on that node (`nemo::node` processes).

**Figure E-3. Cluster State**

```
proc updateLog {snapshotObj} {
    global updateCount fileId
    set t [nemo::snapshot timestamp $snapshotObj]
    set entryId [format "T%04d" $updateCount]
    set entryTime [clock format $t -format {%T,%d-%b-%Y} -gmt 0]
    set mtflexJobs 0
    set hyperscalingJobs 0
    set totalCpus 0
    set activeCpus 0
    foreach jobObj [nemo::snapshot jobs $snapshotObj] {
        set n [nemo::job number_of_masters $jobObj]
        if {$n == 1} { incr mtflexJobs }
        if {$n > 1} { incr hyperscalingJobs }
    }
    foreach nodeObj [nemo::snapshot nodes $snapshotObj] {
        incr totalCpus [nemo::node cpu_count $nodeObj]
        incr activeCpus [nemo::node number_of_processes $nodeObj
            -type calibre]
    }
    puts $fileId "JOBS,$entryId,$mtflexJobs,$hyperscalingJobs"
    puts $fileId "CPUS,$entryId,$totalCpus,$activeCpus"
    puts $fileId "ZZZZ,$entryId,$entryTime"
    flush $fileId
    incr updateCount
}
```

## Message Processing

Upon startup, CalCM creates a TCP/IP server (default port number is 9900) that can be used by external clients to connect to the Cluster Manager and exchange information with running applications using an application-defined text-based communication protocol. Currently, CalCM imposes only one common convention on protocols used by applications: a double “Carriage Return” sequence (“\n\n”) must be used as the message terminator. This is compatible with HTTP protocol used by web browsers, which allows CalCM applications to implement web-based interfaces.

Messages received by CalCM are delivered to applications via a process message subcommand of the nemo::application callback interface. A reference to the object representing the received message is passed as the subcommand's argument and can be used with nemo::message to retrieve message attributes.

**Figure E-4. Message Processing**

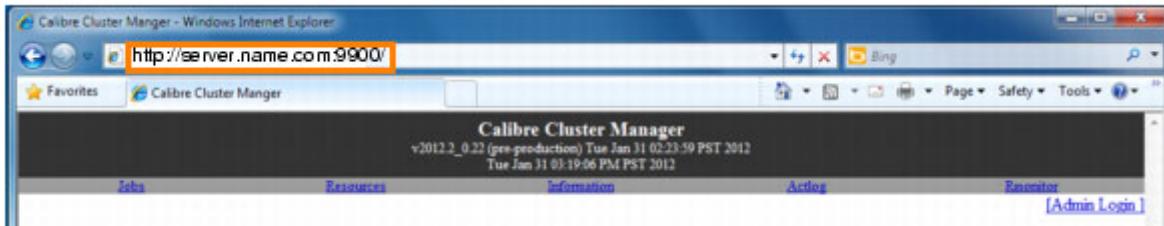
```

proc processMessage {messageObj} {
    global logFilePath
    set messageText [nemo::message text $messageObj]
    if {[regexp {^GET\s+/actlog/?(\S*)\s+HTTP} $messageText -> requestDir]} {
        set replyChannelId [nemo::message channel $messageObj]
        puts $replyChannelId "<html><title>[nemo::application
            name]</title><body>"
        switch $requestDir {
            "" {
                set fileSize [file size $logFilePath]
                puts $replyChannelId "Current log file size: $fileSize<br>"
                puts $replyChannelId "<a href=\"/actlog/view\">View log
                    file</a>"
            }
            "view" {
                puts $replyChannelId "<pre>"
                set f [open $logFilePath "r"]
                while {[gets $f line] != -1} {
                    puts $replyChannelId $line
                }
                close $f
                puts $replyChannelId "</pre>"
            }
            default {
                puts $replyChannelId "<font size=\"+3\""
                    color="red">404</font>"
            }
        }
        puts $replyChannelId "</body></html>"
    }
}

```

The following steps describe the message processing of the CalCM Activity Log application.

1. The CalCM Application is invoked in a browser window by entering the server name and port number. For example:



The solicit command is invoked only once when the CalCM Activity Log application is initialized:

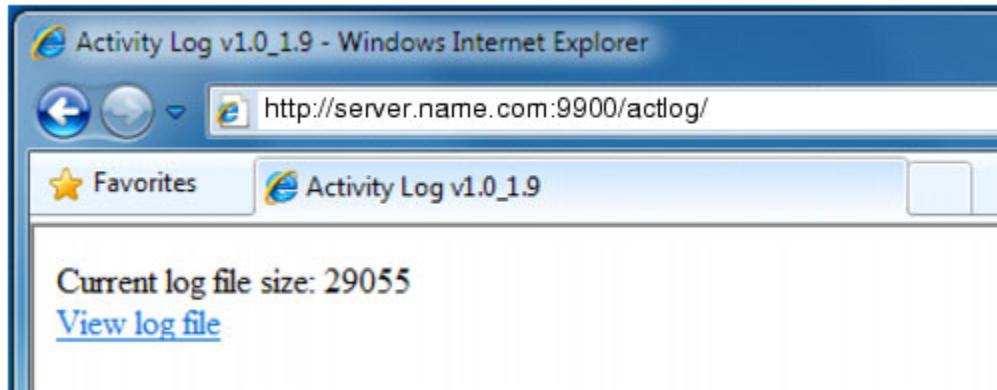
```
# nemo::application --
#
#       Entry point of CalCM application.
#
proc nemo::application {subCmd args} {
    switch $subCmd {
        configure { configureApp [lindex $args 0] }
        name {
            set v "$Revision: 1.9 $"
            regexp {Revision: ([0-9.]+)} $v x subversion
            return "Activity Log v1.0_${subversion}"
        }
        attach { openLogFile }
        detach { closeLogFile }
        update_status { updateLog [lindex $args 0] }
        solicit { return {^GET\s+/actlog/?.*} }
        process_message { processMessage [lindex $args 0] }
    }
}
```

2. Upon clicking the **Actlog** link or entering the URL, the web browser sends the “GET / actlog” message to the CalCM daemon.



3. The CalCM daemon then calls the process\_message command.
4. The CalCM Activity Log application processes the message.
5. The CalCM Activity Log application sends the results to the browser.

6. The browser window is updated to display the results.



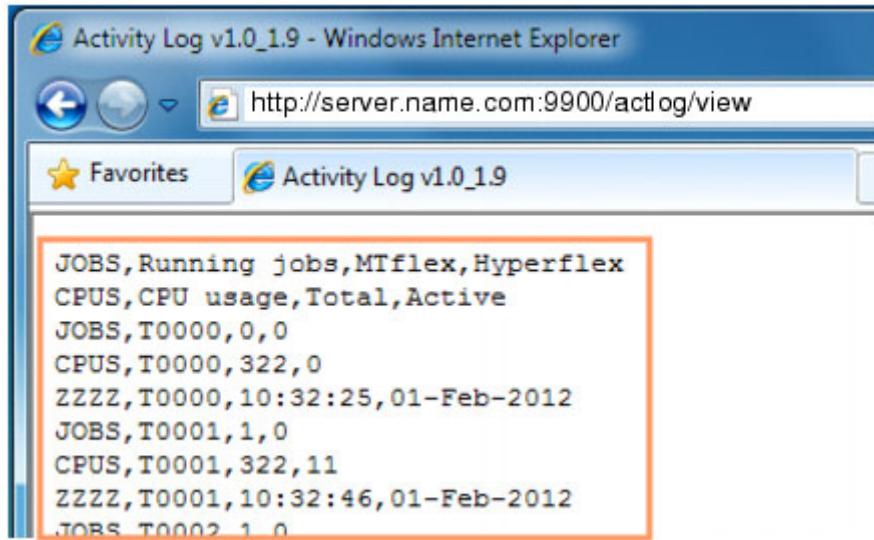
7. Clicking the **View log file** link or entering the URL sends the “GET /actlog/view” message to the CalCM daemon:

```
--- APP UPDATE (Tue Jan 31 15:52:12 2012) ---
Received connection request from sj184dy165:53774
sj184dy165:53774: GET /actlog/view HTTP/1.1
Connection with sj184dy165:53774 closed
```

8. The solicited message is then processed by process\_message.
9. The message is parsed and processed based on a URL with the “/actlog/view” string:

```
if {[regexp {^GET\s+/actlog/?(\S*)\s+HTTP} $messageText -> requestDir]} {
    set replyChannelId [nemo::message channel $messageObj]
    puts $replyChannelId "<html><title>[nemo::application name]</title><body>"
    switch $requestDir {
        ""{
            set fileSize [file size $logFilePath]
            puts $replyChannelId "Current log file size: $fileSize<br>"
            puts $replyChannelId "<a href=\"/actlog/view\">View log
file</a>"
        }
        "view" {
            puts $replyChannelId "<pre>"
            set f [open $logFilePath "r"]
            while {[gets $f line] != -1} {
                puts $replyChannelId $line
            }
        }
    }
}
```

10. The results are sent to the browser and the browser is updated.



## Creating an Activity Log Application

In this procedure, you create a CalCM Activity Log application.

### Prerequisites

- An ASCII editor
- Knowledge of HTML is necessary

### Procedure

1. Set your working directory to *\$HOME/calcm*.

```
$ cd $HOME/calcm
```

2. Open an ASCII editor and enter the following commands that define the log file path and the increment value for each update:

```
# calcm_actlog_app.tcl --
#
# CalCM application "Activity Log"

# Default path to log file.
set filePath "<path
to calcn_examples>/calcm.log"

# Increments after each update.
set updateCount 0
```

3. Enter the following nemo::application command to establish the callback interface of the CalCM application:

```
#  
# Entry point of CalCM application.  
#  
proc nemo::application {subCmd args} {  
    switch $subCmd {  
        configure { configureApp [lindex $args 0] }  
        name { return "Activity Log" }  
        attach { openLogFile }  
        detach { closeLogFile }  
        update_status { updateLog [lindex $args 0] }  
        process_message { processMessage [lindex $args 0] }  
    }  
}
```

4. Enter the configureApp command which is used to parse the application section of the configuration file. The configureApp procedure is called by the nemo::application configure subcommand.

```
#  
# Parse application section of configuration file.  
#  
proc configureApp {configText} {  
    global filePath  
    foreach line [split $configText "\n"] {  
        regexp {^S*LOGPATH=S*=(S+)} $line -> filePath  
    }  
}
```

5. Enter the openLogFile, closeLogFile, and updateLog commands which open, close, and update the log file, respectively. The openLogFile procedure is called by the nemo::application attach subcommand, closeLogFile procedure is called by the nemo::application detach subcommand, and updateLog procedure is called by the nemo::application update\_status subcommand.

```
#  
# Open and initialize log file at configured location.  
#  
proc openLogFile {} {  
    global logFilePath fileId  
    set fileId [open $logFilePath "w"]  
    puts $fileId "JOBS,Running jobs,MTflex,Hyperscaling"  
    puts $fileId "CPUS,CPU usage,Total,Active"  
}  
  
#  
# Close currently open log file.  
#  
proc closeLogFile {} {  
    global fileId  
    close $fileId  
}  
  
#  
# Update log file.  
#  
proc updateLog {snapshotObj} {  
    global updateCount fileId  
    set t [nemo::snapshot timestamp $snapshotObj]  
    set entryId [format "T%04d" $updateCount]  
    set entryTime [clock format $t -format {%T,%d-%b-%Y} -gmt 0]  
    set mtflexJobs 0  
    set hyperscalingJobs 0  
    set totalCpus 0  
    set activeCpus 0  
    foreach jobObj [nemo::snapshot jobs $snapshotObj] {  
        set n [nemo::job number_of_masters $jobObj]  
        if {$n == 1} { incr mtflexJobs }  
        if {$n > 1} { incr hyperscalingJobs }  
    }  
    foreach nodeObj [nemo::snapshot nodes $snapshotObj] {  
        incr totalCpus [nemo::node cpu_count $nodeObj]  
        incr activeCpus [nemo::node number_of_processes $nodeObj  
            -type calibre]  
    }  
    puts $fileId "JOBS,$entryId,$mtflexJobs,$hyperscalingJobs"  
    puts $fileId "CPUS,$entryId,$totalCpus,$activeCpus"  
    puts $fileId "ZZZZ,$entryId,$entryTime"  
    flush $fileId  
    incr updateCount  
}
```

6. Enter the processMessage command which is used to process HTTP requests. The processMessage procedure is called by the nemo::application process\_message subcommand.

```

#
# Process HTTP requests.
#
proc processMessage {messageObj} {
    global logFilePath
    set messageText [nemo::message text $messageObj]
    if {[regexp {^GET\s+/actlog/?(\S*)\s+HTTP} $messageText -> requestDir]} {
        set replyChannelId [nemo::message channel $messageObj]
        puts $replyChannelId "<html><title>[nemo::application name]"
        "</title><body>"
        switch $requestDir {
            "" {
                set fileSize [file size $logFilePath]
                puts $replyChannelId "Current log file size:
$fileSize<br>"
                puts $replyChannelId "<a href=\"/actlog/view\">View log
file</a>"
            }
            "view" {
                puts $replyChannelId "<pre>"
                set f [open $logFilePath "r"]
                while {[gets $f line] != -1} {
                    puts $replyChannelId $line
                }
                close $f
                puts $replyChannelId "</pre>"
            }
            default {
                puts $replyChannelId "<font size=\"+3\""
                "color=\"red\">404</font>"
            }
        }
        puts $replyChannelId "</body></html>"
    }
}

```

7. Save the file as *calc\_actlog\_app.tcl*.
8. Close the file.
9. Verify your working directory is set to *\$HOME/calc* and invoke CalCM.

```
$ calcmd -simulation -config calcmd.conf
```

10. Allow the application to run for several minutes.
11. Press Ctrl-C to exit CalCM.

12. Review the *calclog.log* file that was generated by the Activity Log application. The format of the log file is as follows:

```
JOBS,Running jobs,MTflex,Hyperscaling
CPUS,CPU usage,Total,Active
JOBS,T0000,1,0
CPUS,T0000,19,8
ZZZZ,T0000,15:25:41,01-Dec-2009
JOBS,T0001,1,0
CPUS,T0001,19,8
ZZZZ,T0001,15:27:42,01-Dec-2009
JOBS,T0002,1,0
CPUS,T0002,19,8
ZZZZ,T0002,15:29:47,01-Dec-2009
JOBS,T0003,1,0
CPUS,T0003,19,8
ZZZZ,T0003,15:31:52,01-Dec-2009
```

Notice the log file is updated approximately every two minutes. This is the default update interval.

## Results

You should now have an activity log application and can use the information learned in this exercise to create custom CalCM Tcl applications.

## Related Topics

[nemo::application](#)  
[nemo::job](#)  
[nemo::message](#)  
[nemo::node](#)  
[nemo::snapshot](#)

# Setting Up a Notification Application

This procedure describes how to set up a Tcl notification application for long-running jobs using job meta-data from the JOB INFO keyword in the *job.conf* file.

## Prerequisites

- An ASCII editor for defining the Tcl procedure.
- Knowledge of Tcl scripting language or appropriate reference.
- Browser access to the CalCM dashboard web application. See “[Executing Calibre Jobs to Run Under CalCM with the Dashboard](#)” on page 55.

## Procedure

1. Specify a runtime alert threshold per job using the **JOB INFO** keyword in the *job.conf* file. Units are in seconds.

```
JOB INFO runtime_alert=300
```

2. Define a Tcl procedure in **\$CALCM\_PLATFORM\_INCLUDE** to access this information using non-null default values. This is necessary so that jobs without this value do not cause issues.

```
proc get_jobinfo_runtime_alert {jobObj} {
    set jobinfo_dict [nemo::job attr $jobObj info]
    set value [dict get $jobinfo_dict "runtime_alert"]
    if { $value == {} } {
        set value [expr 72*3600] ;# 72 hours alert by default
    }
    return $value
}
```

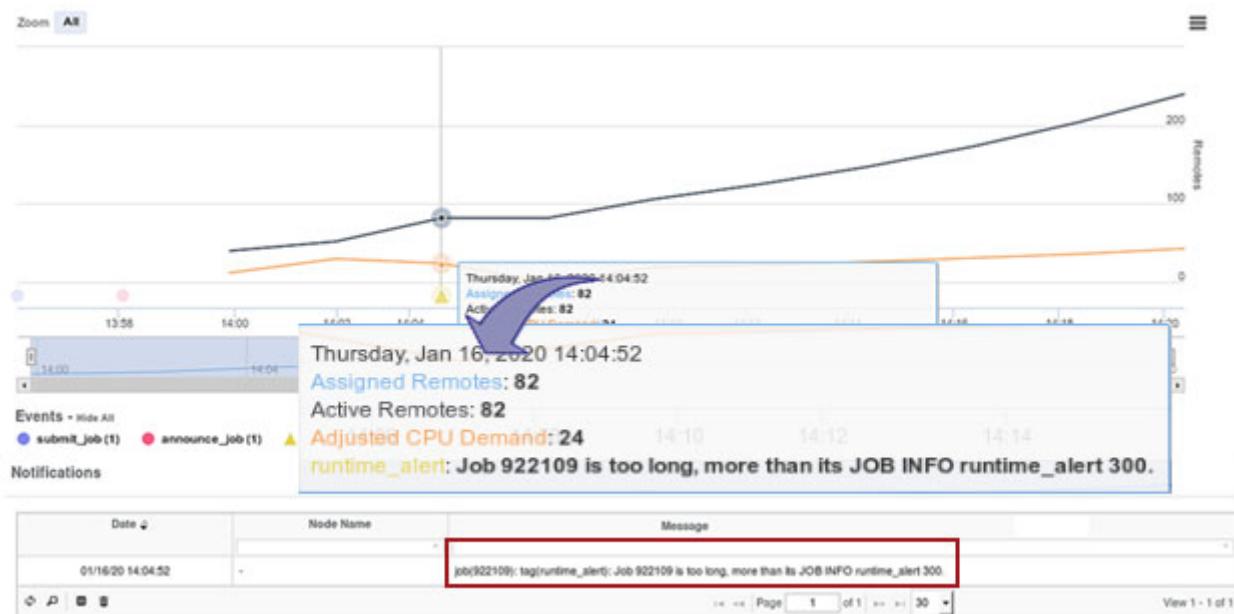
3. Add the following definitions in the **calcm\_notification\_app.tcl** so that the notification is raised only once for the job, not at every *n* cycles:

```
ATTRTRACKERS = {"runtime_alert,$jobID" 2 [expr [nemo::job duration $jobObj] > [get_jobinfo_runtime_alert $jobObj]] {$value == 1 && $pvalue == 0}}
JOBFILTERS   = {evalAttr "runtime_alert,$jobID" {$count == 1}
{tag(runtime_alert): Job $jobID is too long, more than its JOB INFO runtime_alert [get_jobinfo_runtime_alert $jobObj].}}
```

## Results

The resulting notifications are shown in the CalCM dashboard web application (login with user login or as root to see all notifications).

**Figure E-5. Notification in CalCM Dashboard Job Detail Plot**



You can easily view the jobs that raised a recent runtime\_alert.

**Figure E-6. Notifications Table in CalCM Dashboard**

The screenshot shows a table titled "Notifications" with columns for Date, Type, Job, Node, User, and Message. Two rows are listed, both of which are highlighted with a red box and a blue arrow pointing to them from the main message area below the table.

Date	Type	Job	Node	User	Message
01/16/20 14:04:52	Job	922109	-	-	job(922109): tag(runtime_alert): Job 922109 is too long, more than its JOB INFO runtime_alert 300.
01/16/20 13:47:57	Job	922083	-	-	job(922083): tag(runtime_alert): Job 922083 is too long, more than its JOB INFO runtime_alert 300.

# Appendix F

## CalCM Nemo API Reference Dictionary

---

This section provides reference information on the Tcl commands that can be used to create or customize CalCM applications. Information is also provided on the cluster snapshot and related object types.

Refer to [Syntax Conventions](#) for information on the conventions used in this chapter.

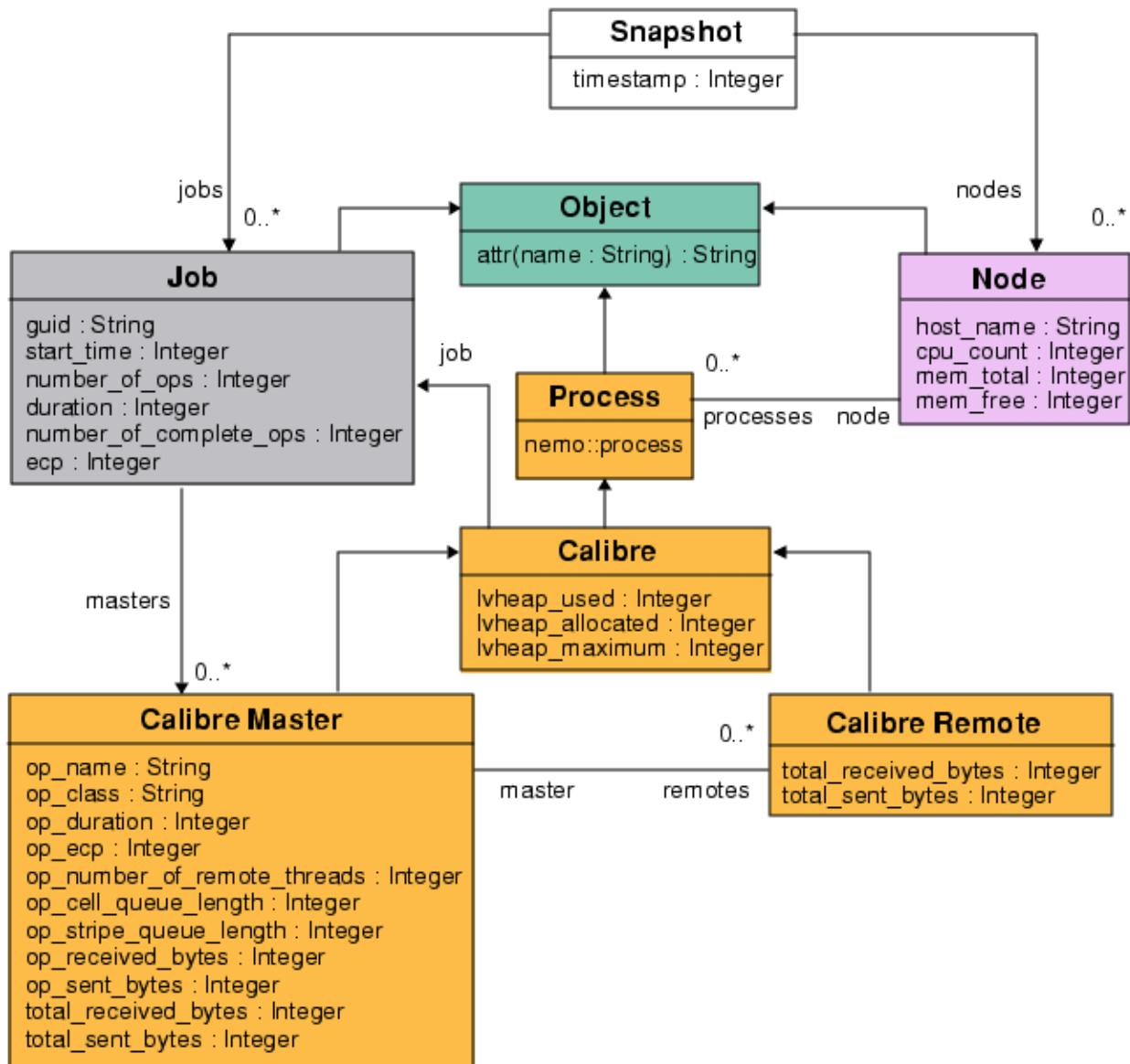
<b>Cluster Snapshot .....</b>	<b>325</b>
<b>Cluster Snapshot Object Types .....</b>	<b>326</b>
<b>CalCM Nemo API Reference .....</b>	<b>328</b>

## Cluster Snapshot

The cluster state is represented in the form of “snapshots.” A snapshot is a collection of consistent information about the cluster state at a specific moment in time.

[Figure F-1](#) identifies the essential attributes of the snapshot components, as well as the relationships between the components.

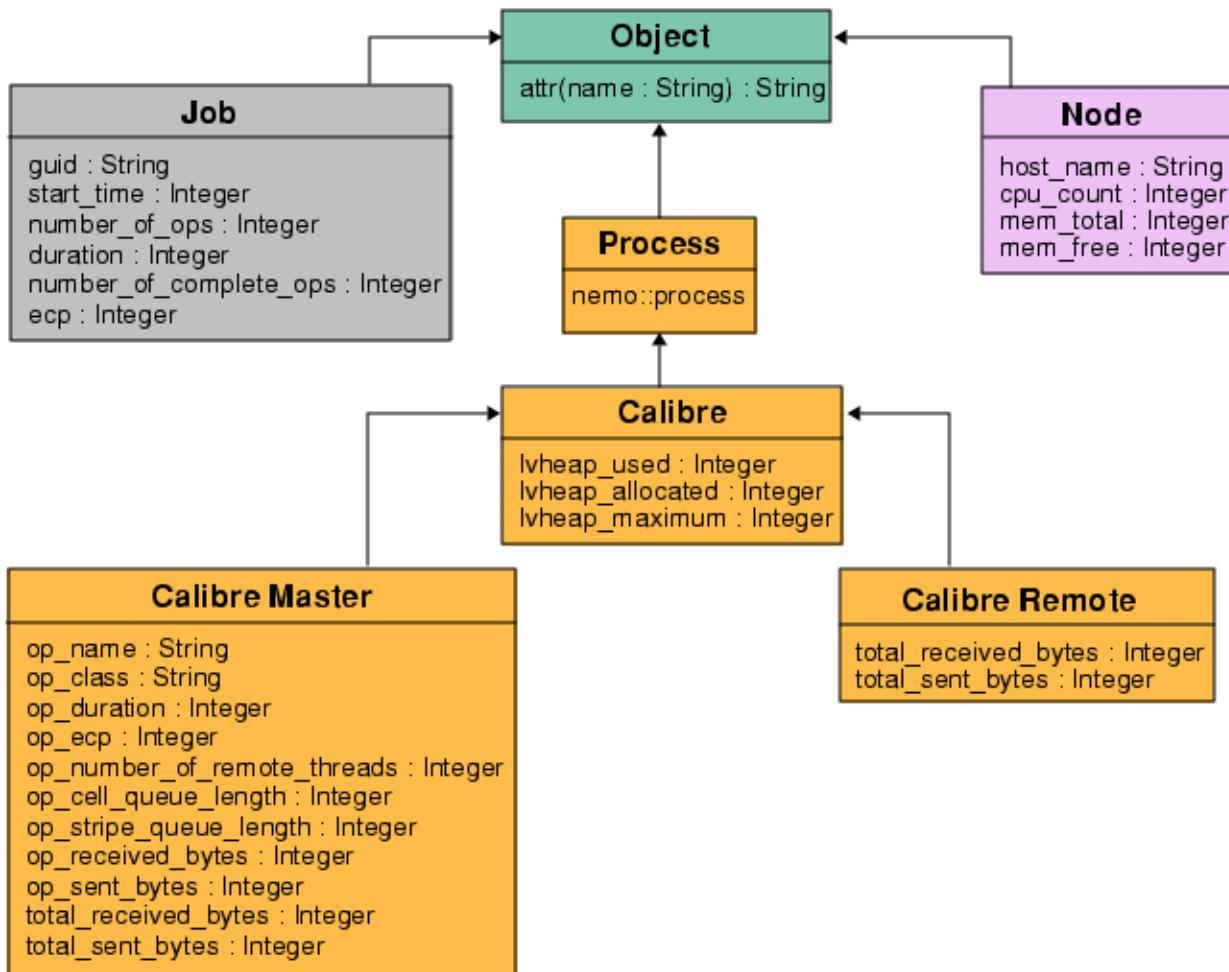
**Figure F-1. Cluster Snapshot**



## Cluster Snapshot Object Types

A cluster snapshot contains objects of different types that form the hierarchy. The most general type is the “Object” type, while “Job,” “Process,” and “Node” are three different object types. The “Calibre” type is a specific “Process” type, which has the two types “Calibre Master” (primary process) and “Calibre Remote” (remote process).

**Figure F-2. Hierarchy of Object Types in Cluster Snapshot**



# CalCM Nemo API Reference

This section provides reference information for the Tcl commands that are available for creating or customizing CalCM applications.

**Table F-1. Tcl Nemo API Commands**

Command Name	Description
<a href="#">nemo::alter</a>	Sets or changes custom attributes.
<a href="#">nemo::application</a>	Establishes the callback interface of CalCM application.
<a href="#">nemo::calibre</a>	Returns information about a Calibre job object.
<a href="#">nemo::control</a>	Provides the ability to interact with CalCM.
<a href="#">nemo::filesystem</a>	Returns information about the file system.
<a href="#">nemo::host</a>	Returns hardware information about the cluster node.
<a href="#">nemo::interface</a>	Returns information about the network interface.
<a href="#">nemo::job</a>	Returns information on the Calibre job that is running.
<a href="#">nemo::master</a>	Returns information about the Calibre primary process.
<a href="#">nemo::message</a>	Returns message attributes.
<a href="#">nemo::node</a>	Returns information about the cluster node.
<a href="#">nemo::object</a>	Returns information about an object in the cluster snapshot.
<a href="#">nemo::process</a>	Returns information about a process running in the cluster snapshot.
<a href="#">nemo::remote</a>	Returns information about a remote Calibre process.
<a href="#">nemo::remotedata</a>	Returns information about a Calibre RDS (Remote Data Server).
<a href="#">nemo::singleton</a>	Evaluate a Tcl script or Tcl file using the singleton interpreter.
<a href="#">nemo::snapshot</a>	Analyze the cluster snapshot.

## nemo::alter

Sets or changes custom attributes.

### Usage

**nemo::alter *obj attrName attrValue [attrName attrValue] ...***

### Arguments

- ***obj***  
Specifies an object in the cluster snapshot whose attribute values you wish to set or change.
- ***attrName attrValue***  
Identifies an attribute name and value associated with the specified object (***obj***). You can specify more than one attribute name and value for an object.

### Description

Sets or changes the values of one or more custom attributes of an object in a cluster snapshot. The attributes then become available to all applications via the “[nemo::object](#) attr” command. Attribute values are stored as text strings. This allows you to pass a Tcl list to nemo::alter, but the list will actually be stored as a string representation.

### Examples

In the following example, the “priority” and “\_tag” attributes are set or changed for the object named \$jobObj.

```
# "priority" attribute will be visible via CalCM web interface;
# "_tag" attribute is for internal use and should stay hidden.
nemo::alter $jobObj "priority" $jobPriority "_tag" 1
```

## **nemo::application**

Establishes the callback interface of CalCM application.

### **Usage**

**nemo::application configure *configText***  
**nemo::application name**  
**nemo::application attach**  
**nemo::application detach**  
**nemo::application process\_message *messageObj***  
**nemo::application update\_status *snapshotObj***

### **Arguments**

- **configure *configText***  
Allows an application to initialize itself by parsing the application-specific section read from a configuration file. This is the first procedure called by nemo::application.
- **name**  
Obtains the human-readable name of the application. Application names are intended for display only. If this option is not defined, the path to the application Tcl script is used as the application name.
- **attach**  
Called once during the initialization stage before any message processing occurs.
- **detach**  
Called once during the cleanup stage. This is the last procedure called on by nemo::application.
- **process\_message *messageObj***  
Passes a message to the application (see [nemo::message](#)). The application analyzes the message attributes and, if recognized, processes the message. Unrecognized messages are silently ignored.
- **update\_status *snapshotObj***  
Informs the application about an update of the cluster state. The system guarantees that the same consistent snapshot argument (see [nemo::snapshot](#)) is passed to all applications during one update cycle.

### **Description**

Applications hosted by CalCM must provide this callback interface in order to be able to interact with the system. The callback interface is implemented as a single nemo::application procedure with different options. Unrecognized options are silently ignored by applications.

## Examples

```
#  
# calcm_mlog_app.tcl --  
#  
# Sample CalCM application "Message Log"  
#  
set filePath "calcmlog.txt"  
  
proc nemo::application {subCmd args} {  
    global filePath  
    switch $subCmd {  
        configure {  
            set configText [lindex $args 0]  
            regexp {LOGPATH\s*=\s*(\S+)} $configText matchVar filePath  
        }  
        name {  
            return "Message Log"  
        }  
        attach {  
            set logfile [open $filePath "w"]  
        }  
        detach {  
            close $logfile  
        }  
        process_message {  
            set messageObj [lindex $args 0]  
            set messageText [nemo::message text $messageObj]  
            if {$messageText != ""} {  
                puts $logfile -----  
                puts $logfile $messageText  
                flush $logfile  
            }  
        }  
    }  
}
```

## **nemo::calibre**

Returns information about a Calibre job object.

### Usage

**nemo::calibre job *processObj***  
**nemo::calibre cpu\_time *processObj***  
**nemo::calibre lvheap\_used *processObj***  
**nemo::calibre lvheap\_allocated *processObj***  
**nemo::calibre lvheap\_maximum *processObj***  
... plus the options of [nemo::object](#) and [nemo::process](#).

### Arguments

- **job *processObj***  
Returns the name of the job object belonging to the Calibre process (see [nemo::job](#)).
- **cpu\_time *processObj***  
Returns the amount of CPU time used by Calibre process, in seconds.
- **lvheap\_used *processObj***  
Returns the amount of LVHEAP used by Calibre process, in MB.
- **lvheap\_allocated *processObj***  
Returns the amount of LVHEAP allocated by Calibre process, in MB.
- **lvheap\_maximum *processObj***  
Returns the maximum amount of LVHEAP that has been allocated by Calibre process, in MB.

### Description

This command is used to return information about a specified Calibre job object (*processObj*) that is running in a cluster snapshot. The command arguments determine the type of information about the Calibre job object that is returned. Process objects are returned by the “[nemo::job master](#)” and the “[nemo::master remotes](#)” statement.

### Examples

```
set jobObj [nemo::calibre job $processObj]
```

## **nemo::control**

Provides the ability to interact with CalCM.

### **Usage**

```
nemo::control update_interval
nemo::control server_addr
nemo::control server_port
nemo::control announce_job host port [jobID]
nemo::control forget_job host port
nemo::control register_node host [ncpu]
nemo::control unregister_node host [kill]
nemo::control reserve_cpu jobObj n nodeObj ...
nemo::control supply_cpu jobObj n nodeObj ...
nemo::control revoke_cpu jobObj n [nodeObj] ...
nemo::control guid_digit jobID
nemo::control start_guid jobID
nemo::control end_guid jobID
nemo::control next_guid jobID
nemo::control acquire_snapshot
nemo::control release_snapshot snapshotObj
```

### **Arguments**

- **update\_interval**  
Returns the update interval specified in [SERVER INTERVAL](#).
- **server\_addr**  
Returns the network address that should be used to establish a connection with the Calibre Cluster Manager. Applications are encouraged to use this command rather than obtain the network name of a local host. This is because on a machine with several Network Interface Cards (NICs), CalCM can be bound to a particular network interface rather than to INADDR\_ANY.<sup>1</sup>
- **server\_port**  
Returns the port number that should be used to establish connection with Calibre Cluster Manager.

---

1. Binding CalCM to a concrete network interface and port number is not currently supported.

- **announce\_job host port [jobID]**

Announces the Calibre job to the Cluster Manager. The host port number and job ID can optionally be passed to this argument. The job announcement does not immediately lead to its addition to the cluster snapshot. Rather, CalCM attempts to connect to the job via the specified control port. If the job responds properly, it is added to the cluster snapshot. The job is removed from the snapshot if it stops responding to status requests. However, the job remains known to CalCM until the command nemo::control forget\_job is called.

- **forget\_job host port**

Removes a job from a list of known jobs. This option automatically leads to the removal of a job from the current snapshot. Snapshots that were acquired earlier are not affected.

- **register\_node host [ncpu]**

Registers a node with the cluster manager with an optional number of CPUs. Registered nodes appear in subsequent cluster snapshots even if no jobs are running, until the nodes are explicitly unregistered. Unregistered nodes associated with Calibre jobs are not implicitly registered and appear in cluster snapshots only as long as the jobs exist.

- **unregister\_node host [kill]**

Unregisters the node specified by the *host* value and revokes the remotes. The specified host only appears in subsequent cluster snapshots if it belongs to a Calibre job. If *kill* is set to 1, then this option kills the remotes instead of revoking them.

- **reserve\_cpu jobObj n nodeObj...**

Reserves *n* number of CPUs on *nodeObj* (see [nemo::node](#)). This prevents the allocation of the same nodes to other jobs because of a delay in the update. The reservation will be canceled when the remote is connected or after some timeout. The reserved process has a PID less than 0. It is possible to specify 0 for the *jobObj* as if a job is not yet launched.

- **supply\_cpu jobObj n nodeObj...**

Signals a job (see [nemo::job](#)) to use more processing power by utilizing *n* additional CPUs on *nodeObj* (see [nemo::node](#)). The effect of this command (if any) is not immediate. You must check the cluster state during the next update cycle to see if and how the job responded to the command. True is returned upon success of the request.

- **revoke\_cpu jobObj n [nodeObj] ...**

Signals a job (see [nemo::job](#)) to release the specified number of CPUs. If a particular node is specified (see [nemo::node](#)), the CPUs on the corresponding host are released. As with the “nemo::control supply\_cpu” statement, the effect of this command (if any) is not immediate. You must check the cluster state during the next update cycle to see if and how the job responded to the command.

- **guid\_digit jobID**

Set the number of digits for the job GUID to uniquely identify a Calibre job.

- **start\_guid jobID**

Set the start job GUID to be assigned to uniquely identify a Calibre job.

- **end\_guid *jobID***

Set the maximum job GUID to be assigned to uniquely identify a Calibre job.

- **next\_guid *jobID***

Sets the next job GUID to be assigned to uniquely identify the next Calibre job.

- **acquire\_snapshot**

Returns a snapshot of the current cluster state (see [nemo::snapshot](#)). The returned object is guaranteed to remain valid and immutable until the “nemo::control release\_snapshot” statement is executed on the current cluster. Note that two consecutive calls to “nemo::control acquire\_snapshot” may return completely different objects if the cluster state is updated between the calls. However, in such a case each snapshot is guaranteed to be self-consistent.

- **release\_snapshot *snapshotObj***

Indicates that the application no longer needs the specified snapshot. This does not necessarily free resources used by the released object because other references to it may exist in current or other applications.

Generally, you should not use this procedure because an acquired snapshot is automatically released when appropriate. This can occur when the corresponding Tcl variable becomes unavailable because it goes out of scope or is assigned a different value, for example. The internal Tcl reference counting mechanism ensures that objects are deleted when appropriate. The preferred method for explicitly releasing a snapshot is to unset the corresponding variable, which guarantees that snapshot will not be mistakenly used after having been released.

## Description

This command provides the ability to interact with CalCM.

## Examples

```
set snapshotObj [nemo::control acquire_snapshot]
```

## **nemo::filesystem**

Returns information about the file system.

### Usage

**nemo::filesystem node *filesystemObj***  
**nemo::filesystem name *filesystemObj***  
**nemo::filesystem type *filesystemObj***  
**nemo::filesystem total\_space *filesystemObj***  
**nemo::filesystem free\_space *filesystemObj***  
**nemo::filesystem available\_space *filesystemObj***  
... plus the options of [nemo::object](#).

### Arguments

- **node *filesystemObj***

Returns the cluster node the file system belongs to (see [nemo::node](#)).

- **name *filesystemObj***

Returns the name of file system or the mount point. A shorter notation can be used, in which the file system object is treated as a string variable. The following two lines of code are equivalent:

```
if {[nemo::filesystem name $filesystemObj] == "/" } { ... }  
if {$filesystemObj == "/" } { ... }
```

- **type *filesystemObj***

Returns the type of file system (for example: ext3, vfat, or nfs).

- **total\_space *filesystemObj***

Returns the total amount of space allocated for the file system, in MB.

- **free\_space *filesystemObj***

Returns the amount of free space, in MB.

- **available\_space *filesystemObj***

Returns the amount of space available to non-root, in MB.

### Description

This command is used to return information about a specified file system (*filesystemObj*) in the cluster snapshot. The command arguments determine the type of file system information that is returned. A list of file systems is returned by the “[nemo::node filesystems](#)” statement.

## Examples

```
set valueFromNemo [nemo::filesystem name $nemoFsObj]
```

## **nemo::host**

Returns hardware information about the cluster node.

### **Usage**

**nemo::host is\_registered *nodeObj***  
**nemo::host status *nodeObj***  
**nemo::host flag *nodeObj***  
**nemo::host timestamp *nodeObj***  
**nemo::host host\_name *nodeObj***  
**nemo::host mem\_total *nodeObj***  
**nemo::host mem\_free *nodeObj***  
**nemo::host swap\_total *nodeObj***  
**nemo::host swap\_free *nodeObj***  
**nemo::host disk\_total *nodeObj***  
**nemo::host disk\_free *nodeObj***  
**nemo::host load\_1 *nodeObj***  
**nemo::host load\_5 *nodeObj***  
**nemo::host load\_15 *nodeObj***  
**nemo::host os\_name *nodeObj***  
**nemo::host os\_release *nodeObj***  
**nemo::host os\_platform *nodeObj***  
**nemo::host cpu\_count *nodeObj***  
**nemo::host cpu\_clock *nodeObj***  
**nemo::host boottime *nodeObj***  
**nemo::host bytes\_in *nodeObj***  
**nemo::host bytes\_out *nodeObj***  
**nemo::host proc\_total *nodeObj***  
**nemo::host proc\_run *nodeObj***

### **Arguments**

- **is\_registered *nodeObj***

Returns whether the ***nodeObj*** is registered in *calcm\_hmonitor\_app.tcl*. Only the registered nodes can be used to launch remotes.

- **status *nodeObj***

Returns the status of the *nodeObj*. The meaning of the bit flag is as follows:

- **0** — The remote host is OK.
- **1** — The remote host is down and not available.
- **2** — The remote host is closed with the close\_node message.
- **4** — The remote host is on hold in *calcm\_hmonitor\_app.tcl* for the reset.

- **flag *nodeObj***

Returns the flag specified with “[nemo::control register\\_node](#)”.

- **timestamp *nodeObj***

Returns the time stamp of the node as an integer number of seconds from the “epoch.” The time stamp indicates when the node was last updated.

- **host\_name *nodeObj***

Returns the network name of the cluster node. A shorter notation can be used, in which the node object is treated as string variable. The following two lines of code are equivalent:

```
if {[nemo::node host_name $nodeObj] == "localhost"} { ... }  
if {$nodeObj == "localhost"} { ... }
```

- **mem\_total *nodeObj***

Returns the total amount of physical memory (in MBs) installed on the cluster node.

- **mem\_free *nodeObj***

Returns the amount of free physical memory (in MBs) available on the cluster node.

- **swap\_total *nodeObj***

Returns the total amount of swap space (in MBs) available on the cluster node.

- **swap\_free *nodeObj***

Returns the amount of free swap space (in MBs) available on the cluster node.

- **disk\_total *nodeObj***

Total available disk space in GB.

- **disk\_free *nodeObj***

Total free disk space in GB.

- **load\_1 *nodeObj***

Returns the system load averaged over the last minute. This value typically represents the number of active tasks on the system averaged over a period of time.

- **load\_5 *nodeObj***

Returns the system load averaged over the last 5 minutes.

- **load\_15 *nodeObj***  
Returns the system load averaged over the last 15 minutes.
- **os\_name *nodeObj***  
Returns the OS name (for example, “Linux”).
- **os\_release *nodeObj***  
Returns the OS release info (for example, “#1 SMP Thu Apr 22 00:09:47 EDT 2004”).
- **os\_platform *nodeObj***  
Returns the name of the hardware platform (for example, “x86 64”).
- **cpu\_count *nodeObj***  
Returns the number of physical CPUs.
- **cpu\_clock *nodeObj***  
Returns the CPU clock speed (in MHz).
- **boottime *nodeObj***  
Returns the last time that the system was started.
- **bytes\_in *nodeObj***  
Returns the number of bytes in per second through the network.
- **bytes\_out *nodeObj***  
Returns the number of bytes out per second through the network.
- **proc\_total *nodeObj***  
Returns the total number of processes.
- **proc\_run *nodeObj***  
Returns the total number of running processes.

## Description

This command is used to return hardware information about a cluster node (*nodeObj*) in the Ganglia or CalScope report. The information is returned only if the *calcmon\_hmonitor\_app.tcl* is registered and the correct Ganglia server is connected. The command arguments determine the type of information about the cluster node that is returned. A list of known cluster nodes is returned by the “[nemo::snapshot nodes](#)”; the “[nemo::process node](#)” statement returns the node for a specific process.

## Examples

```
set osRelease [nemo::host os_release $nodeObj]
```

# nemo::interface

Returns information about the network interface.

## Usage

**nemo::interface node *interfaceObj***  
**nemo::interface name *interfaceObj***  
**nemo::interface speed *interfaceObj***  
**nemo::interface addr *interfaceObj***  
**nemo::interface netmask *interfaceObj***  
... plus the options of [nemo::object](#).

## Arguments

- **node *interfaceObj***  
Returns the cluster node the interface belongs to (see [nemo::node](#)).
- **name *interfaceObj***  
Returns the name of network interface. A shorter notation can be used, in which the interface object is treated as string variable. The following two lines of code are equivalent:

```
if {[nemo::interface name $interfaceObj] == "eth0"} { ... }  
if {$interfaceObj == "eth0"} { ... }
```
- **speed *interfaceObj***  
Returns the speed of the network interface, in Mbit per second.
- **addr *interfaceObj***  
Returns the IP address of the network interface.
- **netmask *interfaceObj***  
Returns the network mask.

## Description

This command is used to return information about a specified network interface (*interfaceObj*) in the cluster snapshot. The command arguments determine the type of network interface information that is returned. The list of network interfaces is returned by the “[nemo::node](#) interfaces” statement.

## Examples

```
set valueFromNemo [nemo::interface name $nemoIfaceObj]
```

## **nemo::job**

Returns information on the Calibre job that is running.

### **Usage**

**nemo::job control\_addr *jobObj***  
**nemo::job control\_port *jobObj***  
**nemo::job guid *jobObj***  
**nemo::job start\_time *jobObj***  
**nemo::job number\_of\_ops *jobObj***  
**nemo::job masters *jobObj***  
**nemo::job number\_of\_masters *jobObj***  
**nemo::job master *jobObj* [n]**  
**nemo::job timestamp *jobObj***  
**nemo::job duration *jobObj***  
**nemo::job stage *jobObj***  
**nemo::job number\_of\_complete\_ops *jobObj***  
**nemo::job ecp *jobObj***  
**nemo::job cpu\_demand *jobObj***  
**nemo::job cpu\_adjustable *jobObj***  
**nemo::job max\_count *jobObj***  
**nemo::job number\_of\_remove\_pending *jobObj***  
**nemo::job number\_of\_unmatched\_pending *jobObj***  
... plus the options of [nemo::object](#).

### **Arguments**

- **control\_addr *jobObj***

Returns the network address that can be used to establish the control connection with a Calibre job.

- **control\_port *jobObj***

Returns the port number that can be used to establish the control connection with a Calibre job.

- **guid *jobObj***

Returns the unique identifier of a Calibre job. A shorter notation can be used, in which the job object is treated as string variable. The following two lines of code are equivalent:

```
if {[nemo::job guid $jobObj] == $guid} { ... }
if {$jobObj == $guid} { ... }
```

- **start\_time *jobObj***

Returns the start time of a Calibre job as an integer number of seconds from the “Epoch”.

- **number\_of\_ops *jobObj***

Returns the total number of operations in a Calibre job.

- **masters *jobObj***

Returns a list of Calibre primary processes executing the job (see [nemo::master](#)). Calibre MTflex jobs always have exactly one primary, whereas Hyperscaling jobs have multiple primaries, each controlling one HDB. The first element in the returned list of primary processes always corresponds to HDB0.

- **number\_of\_masters *jobObj***

Returns the number of Calibre primary processes executing the job. Calibre MTflex jobs always have exactly one primary, whereas Hyperscaling jobs have multiple primaries, each controlling one HDB.

- **master *jobObj* [*n*]**

By default, returns the main primary process controlling HDB0 (see [nemo::master](#)). An optional index can be provided, in which case the process controlling HDB *n* is returned.

- **timestamp *jobObj***

Returns the timestamp of a Calibre job as an integer number of seconds from the “Epoch.” The timestamp indicates when the job was last updated.

- **duration *jobObj***

Returns the duration of a Calibre job, in seconds.

- **stage *jobObj***

Returns a short text description of the current job stage.

- **number\_of\_complete\_ops *jobObj***

Returns the number of complete operations in the Calibre job.

- **ecp *jobObj***

Returns an estimated completion percentage of the Calibre job (0-100).

- **cpu\_demand *jobObj***

Returns an indicator of CPU demand by the Calibre job. If positive, the value represents the number of additional CPUs that can be added to the job to improve performance. If

negative, the absolute value represents the number of CPUs that the job is not utilizing. If zero, then CPU utilization by job is optimal.

- **cpu\_adjustable *jobObj***

Returns whether the request to supply CPUs can be done. At the early stage, the adjustment of CPUs is not accepted and the allocated CPU is not used. Therefore, it is advised to check the return flag of this command before allocating CPUs.

- **max\_count *jobObj***

Returns the maximum number of CPUs specified in the Calibre MTflex configuration file. A request to add CPUs that exceeds the limit is ignored.

- **number\_of\_remove\_pending *jobObj***

Returns the number of pending removes of remotes in the previous revoke process.

- **number\_of\_unmatched\_pending *jobObj***

Returns the number of unmatched physical/virtual remotes in the previous revoke process.

## Description

This command is used to return information about a specified Calibre job (*jobObj*) in a cluster snapshot. The command arguments determine the type of information about the job that is returned. A list of jobs is returned by the “[nemo::snapshot jobs](#)” command. The command “[nemo::calibre job](#)” also returns information about a given job.

## Examples

```
set opNumber [nemo::job number_of_complete_ops $jobObj]
```

## nemo::master

Returns information about the Calibre primary process.

### Usage

```
nemo::master hdb_number processObj
nemo::master op_name processObj
nemo::master op_class processObj
nemo::master op_class_name processObj
nemo::master op_duration processObj
nemo::master op_ecp processObj
nemo::master op_number_of_awake_threads processObj
nemo::master op_number_of_rcs processObj
nemo::master op_number_of_remote_threads processObj
nemo::master op_cell_queue_length processObj
nemo::master op_stripe_queue_length processObj
nemo::master op_received_bytes processObj
nemo::master op_sent_bytes processObj
nemo::master total_received_bytes processObj
nemo::master total_sent_bytes processObj
nemo::master number_of_remotes processObj
nemo::master remotes processObj
nemo::master number_of_remotedata processObj
nemo::master remotedata processObj
... plus the options of nemo::object, nemo::process, and nemo::calibre.
```

### Arguments

- **hdb\_number** *processObj*  
Returns the number of HDBs controlled by the Calibre primary process.
- **op\_name** *processObj*  
Returns the name of the current operation.
- **op\_class** *processObj*  
Returns the class of the current operation (for example: DRC:11).

- **op\_class\_name *processObj***  
Returns the class name of the current operation (for example: OPCSBAR).
- **op\_duration *processObj***  
Returns the duration of the current operation, in seconds.
- **op\_ecp *processObj***  
Returns the estimated completion of the current operation, in percentage (0—100).
- **op\_number\_of\_awake\_threads *processObj***  
Returns the number of awake threads used by the current operation. This number includes the number of active threads on both the primary and the remotes.
- **op\_number\_of\_rcs *processObj***  
Returns the number of RCS processes allocated to the primary process.
- **op\_number\_of\_remote\_threads *processObj***  
Returns the number of active remote threads used by the current operation.
- **op\_cell\_queue\_length *processObj***  
Returns the length of the cell queue used by the current operation.
- **op\_stripe\_queue\_length *processObj***  
Returns the length of the stripe queue used by the current operation.
- **op\_received\_bytes *processObj***  
Returns the number of bytes received by the current operation from remotes.
- **op\_sent\_bytes *processObj***  
Returns the number of bytes sent by the current operation to remotes.
- **total\_received\_bytes *processObj***  
Returns the total number of bytes received by the primary process from remotes.
- **total\_sent\_bytes *processObj***  
Returns the total number of bytes sent by the primary process to remotes.
- **number\_of\_remotes *processObj***  
Returns the number of connected remotes.
- **remotes *processObj***  
Returns a list of connected remotes.
- **number\_of\_remotedata *processObj***  
Returns the number of connected RDS.
- **remotedata *processObj***  
Returns a list of connected RDS.

## Description

This command is used to return information about the Calibre primary process (*processObj*) in a cluster snapshot. The command arguments determine the type of information about the primary process that is returned. A list of primary processes is returned by the “[nemo::job master](#)” or the “[nemo::remote master](#)” statement.

## Examples

```
foreach masterObj [nemo::job masters $jobObj] {  
    incr totalRemoteProcesses [nemo::master number_of_remotes $masterObj]  
    incr totalActiveProcesses [nemo::master op_number_of_remote_threads  
        $masterObj]  
    lappend masterOpClass [nemo::master op_class $masterObj]  
}
```

## **nemo::message**

Returns message attributes.

### **Usage**

**nemo::message timestamp *messageObj***

**nemo::message addr *messageObj***

**nemo::message user *messageObj***

**nemo::message text *messageObj***

**nemo::message channel *messageObj***

### **Arguments**

- **timestamp *messageObj***

Returns the timestamp of the message received as an integer number of seconds from the “epoch.” The timestamp indicates when the message was received.

- **addr *messageObj***

Returns the address where the message is sent as in *hostname:port*.

- **user *messageObj***

Returns the username of the user who sent the message.

- **text *messageObj***

Returns the text contained in the message (*messageObj*). A shorter notation can be used, in which the message object is treated as string variable. The following two lines of code are equivalent:

```
if {[nemo::message text $messageObj] == "stop"} { ... }  
if {$messageObj == "stop"} { ... }
```

- **channel *messageObj***

Returns the channel that can be used to reply to the message. Standard Tcl library functions, such as puts and fconfigure, can be used with the returned object. The application does not have to close the channel.

### **Description**

This command is used to return the attributes of the specified message object (*messageObj*). The command arguments determine the type of information about the message that is returned. The message object is passed to the application using the “[nemo::application process\\_message](#)” statement.

## Examples

```
proc nemo::application {subCmd args} {
    switch $subCmd {
        process_message {
            set messageObj [lindex $args 0]
            if {$messageObj == "How are you?"} {
                set channelId [nemo::message channel $messageObj]
                puts $channelId "I'm fine, thank you!"
            }
        }
    }
}
```

## **nemo::node**

Returns information about the cluster node.

### Usage

```
nemo::node timestamp nodeObj
nemo::node host_name nodeObj
nemo::node number_of_processes nodeObj [-type {generic | calibre | calibre_master | calibre_remote}]
nemo::node processes nodeObj [-type {generic | calibre | calibre_master | calibre_remote}]
nemo::node mem_total nodeObj
nemo::node mem_free nodeObj
nemo::node swap_total nodeObj
nemo::node swap_free nodeObj
nemo::node load_1 nodeObj
nemo::node load_5 nodeObj
nemo::node load_15 nodeObj
nemo::node os_name nodeObj
nemo::node os_version nodeObj
nemo::node os_release nodeObj
nemo::node os_platform nodeObj
nemo::node slot_count nodeObj
nemo::node cpu_count nodeObj
nemo::node cpu_model nodeObj
nemo::node cpu_vendor nodeObj
nemo::node cpu_clock nodeObj
nemo::node number_of_interfaces nodeObj
nemo::node interfaces nodeObj
nemo::node number_of_filesystems nodeObj
nemo::node filesystems nodeObj
nemo::node number_of_unmatched_pending nodeObj
... plus the options of nemo::object.
```

## Arguments

- **timestamp *nodeObj***

Returns the time stamp of the node as an integer number of seconds from the “epoch.” The time stamp indicates when the node was last updated.

- **host\_name *nodeObj***

Returns the network name of the cluster node. A shorter notation can be used, in which the node object is treated as string variable. The following two lines of code are equivalent:

```
if {[nemo::node host_name $nodeObj] == "localhost"} { ... }  
if {$nodeObj == "localhost"} { ... }
```

- **number\_of\_processes *nodeObj* [-type**

{generic | calibre | calibre\_master | calibre\_remote}]

Returns the number of processes running on the cluster node.

The -type argument can be used to limit command output to only processes of a specific kind:

generic — Counts all processes.

calibre — Counts only Calibre processes.

calibre\_master — Counts only Calibre primary processes.

calibre\_remote — Counts only Calibre remote processes.

The default behavior is to count all processes (-type generic).

- **processes *nodeObj* [-type {generic | calibre | calibre\_master | calibre\_remote}]**

Returns a list of processes running on the cluster node (see [nemo::process](#)). The -type argument can be used to limit command output to only processes of a specific kind:

generic — Returns all processes.

calibre — Returns only Calibre processes.

calibre\_master — Returns only Calibre primary processes.

calibre\_remote — Returns only Calibre remote processes.

The default behavior is to return all processes (-type generic).

- **mem\_total *nodeObj***

Returns the total amount of physical memory (in MBs) installed on the cluster node.

- **mem\_free *nodeObj***

Returns the amount of free physical memory (in MBs) available on the cluster node.

- **swap\_total *nodeObj***

Returns the total amount of swap space (in MBs) available on the cluster node.

- **swap\_free *nodeObj***  
Returns the amount of free swap space (in MBs) available on the cluster node.
- **load\_1 *nodeObj***  
Returns the system load averaged over the last minute. This value typically represents the number of active tasks on the system averaged over a period of time.
- **load\_5 *nodeObj***  
Returns the system load averaged over the last 5 minutes.
- **load\_15 *nodeObj***  
Returns the system load averaged over the last 15 minutes.
- **os\_name *nodeObj***  
Returns the OS name (for example, “Linux”).
- **os\_version *nodeObj***  
Returns the OS version (for example, “2.4.21-15.EL”).
- **os\_release *nodeObj***  
Returns the OS release info (for example, “#1 SMP Thu Apr 22 00:09:47 EDT 2004”).
- **os\_platform *nodeObj***  
Returns the name of the hardware platform (for example, “x86 64”).
- **slot\_count *nodeObj***  
Returns the number of slots allocated to the cluster node.
- **cpu\_count *nodeObj***  
Returns the number of physical CPUs.
- **cpu\_model *nodeObj***  
Returns the CPU model (for example, “Intel(R) Xeon(TM) CPU 3.20GHz”).
- **cpu\_vendor *nodeObj***  
Returns the name of CPU vendor (for example, “GenuineIntel”).
- **cpu\_clock *nodeObj***  
Returns the CPU clock speed (in MHz).
- **number\_of\_interfaces *nodeObj***  
Returns the number of network interfaces available at the cluster node.
- **interfaces *nodeObj***  
Returns the list of network interfaces available at the cluster node (see [nemo::interface](#)).
- **number\_of\_filesystems *nodeObj***  
Returns the number of file systems available at the cluster node.

- **filesystems *nodeObj***

Returns a list of file systems available at the cluster node (see [nemo::filesystem](#)).

- **number\_of\_unmatched\_pending *nodeObj***

Returns the number of unmatched physical/virtual remotes in the node caused by the previous revoke process.

## Description

This command is used to return information about a cluster node (***nodeObj***) in a cluster snapshot. The command arguments determine the type of information about the cluster node that is returned. A list of known cluster nodes is returned by the “[nemo::snapshot nodes](#)”; the “[nemo::process node](#)” statement returns the node for a specific process.

## Examples

```
#  
# printNodeTab --  
#  
# Prints table of all cluster nodes to stdout.  
#  
proc printNodeTab {snapshotObj} {  
    puts "-----+-----+-----"  
    puts "| Host Name | CPUs | RAM |"  
    puts "-----+-----+-----"  
    foreach nodeObj [nemo::snapshot nodes $snapshotObj] {  
        set h [nemo::node host_name $nodeObj]  
        set c [nemo::node cpu_count $nodeObj]  
        set m [nemo::node mem_total $nodeObj]  
        puts [format "| %9s | %4d | %7d |" $h $c $m]  
    }  
    puts "-----+-----+-----"  
}
```

## **nemo::object**

Returns information about an object in the cluster snapshot.

### **Usage**

**nemo::object attr *obj attrName***  
**nemo::object attr\_names *obj***

### **Arguments**

- **attr *obj attrName***

Returns the value of a named custom attribute of an object. If the attribute is not defined, an empty string is returned.

- **attr\_names *obj***

Returns a list of all custom attribute names.

### **Description**

This command is applicable to all entities that can be present in a cluster snapshot, such as Calibre jobs (see [nemo::job](#)), cluster nodes (see [nemo::node](#)), and processes (see [nemo::process](#)).

### **Examples**

```
#  
# jobPriority --  
#  
# Return priority of given job.  
#  
proc jobPriority {jobObj} {  
    set p [nemo::object attr $jobObj "priority"]  
    if {$p == ""} { set p 0 }  
    return $p  
}
```

## nemo::process

Returns information about a process running in the cluster snapshot.

### Usage

```
nemo::process node processObj
nemo::process type processObj
nemo::process is [generic | calibre | calibre_master | calibre_remote] processObj
nemo::process pid processObj
nemo::process name processObj
nemo::process user_time processObj
nemo::process system_time processObj
nemo::process vmsize processObj
nemo::process rss processObj
nemo::process owner processObj
nemo::process cpu_usage processObj
... plus the options of nemo::object.
```

### Arguments

- **node *processObj***  
Returns the cluster node on which the specified process (*processObj*) is running (see [nemo::node](#)).
- **type *processObj***  
Returns the process type as one of the following string constants:
  - generic — The process type is a generic, non-Calibre process.
  - calibre\_master — The process type is a Calibre primary process.
  - calibre\_remote — The process type is a Calibre remote process.
- **is [generic | calibre | calibre\_master | calibre\_remote] *processObj***  
Returns a boolean indicating whether the process is of the specified type.
- **pid *processObj***  
Returns the process identifier. The PID should be greater than 0, with the exception of a reserved process, which is less than or equal to 0.
- **name *processObj***  
Returns the process name.

- **user\_time *processObj***

Returns the amount of CPU time (in seconds) spent by the user executing process instructions.

- **system\_time *processObj***

Returns the amount of CPU time (in seconds) spent by the system executing process tasks.

- **vmsize *processObj***

Returns the virtual memory size (in MB).

- **rss *processObj***

Returns the resident set size, in MB.

- **owner *processObj***

Returns the username of the process owner.

- **cpu\_usage *processObj***

Returns the approximate percentage of current CPU usage (0—100).

## Description

This command is used to return information about a specific process (*processObj*) in a cluster snapshot. The command arguments determine the type of information returned about the process object. A list of processes running on a cluster node is returned by the “[nemo::node processes](#)” statement.

## Examples

```
set hostName [nemo::node host_name [nemo::process node $masterObj]]
```

## nemo::remote

Returns information about a remote Calibre process.

### Usage

```
nemo::remote master processObj
nemo::remote total_received_bytes processObj
nemo::remote total_sent_bytes processObj
nemo::remote rds_send_blocks processObj
nemo::remote rds_send_rtt processObj
nemo::remote rds_send_retx processObj
nemo::remote rds_recv_blocks processObj
nemo::remote rds_recv_rtt processObj
nemo::remote rds_recv_retx processObj
```

... plus the options of [nemo::object](#), [nemo::process](#), and [nemo::calibre](#).

### Arguments

- **master *processObj***  
Returns the primary object that is connected to the remote.
- **total\_received\_bytes *processObj***  
Returns the total number of bytes received by the remote process from the primary.
- **total\_sent\_bytes *processObj***  
Returns the total number of bytes sent by the remote process to the primary.
- **rds\_send\_blocks *processObj***  
Returns the total number of SEND transactions by the remote process.
- **rds\_send\_rtt *processObj***  
Returns the average number of SEND round trip transaction time by the remote process.
- **rds\_send\_retx *processObj***  
Returns the total number of SEND transactions retransmitted by the remote process.
- **rds\_recv\_blocks *processObj***  
Returns the total number of RECV transactions by the remote process.
- **rds\_recv\_rtt *processObj***  
Returns the average number of RECV round trip transaction time by the remote process.
- **rds\_recv\_retx *processObj***  
Returns the total number of RECV transactions retransmitted by the remote process.

## Description

This command is used to return information about a remote Calibre process (*processObj*) in a cluster snapshot. The command arguments determine the type of information returned about the remote process. A list of connected remotes is returned by the “[nemo::master](#) remotes” statement.

## Examples

```
#  
# printJobHosts --  
#  
# Prints names of all hosts used by Calibre job to stdout.  
#  
proc printJobHosts {jobObj} {  
    set masterObj [nemo::job master $jobObj]  
    setGet nodeArray([nemo::master node $masterObj]) 1  
    foreach remoteObj [nemo::master remotes $masterObj] {  
        set nodeArray([nemo::remote node $remoteObj]) 1  
    }  
    foreach hostName [lsort [array names nodeArray]] {  
        puts $hostName  
    }  
}
```

## **nemo::remotedata**

Returns information about a Calibre Remote Data Server (RDS).

### Usage

**nemo::remotedata master *processObj***  
**nemo::remotedata rds\_type *processObj***  
**nemo::remotedata rds\_send\_blocks *processObj***  
**nemo::remotedata rds\_send\_retx *processObj***  
**nemo::remotedata rds\_recv\_blocks *processObj***  
**nemo::remotedata rds\_recv\_retx *processObj***  
**nemo::remotedata rds\_errors *processObj***

... plus the options of [nemo::object](#), [nemo::process](#), and [nemo::calibre](#).

### Arguments

- **master *processObj***  
Returns the primary object that is connected to the RDS.
- **rds\_type *processObj***  
Returns the type of RDS. Valid types include:
  - 1 — The main RDS
  - 2 — The backup RDS
- **rds\_send\_blocks *processObj***  
Returns the total number of SEND blocks by the RDS.
- **rds\_send\_retx *processObj***  
Returns the total number of SEND blocks retransmitted by the RDS.
- **rds\_recv\_blocks *processObj***  
Returns the total number of RECV blocks by the RDS.
- **rds\_recv\_retx *processObj***  
Returns the total number of RECV blocks retransmitted by the RDS.
- **rds\_errors *processObj***  
Returns the number of non-catastrophic errors that caused an RDS not to respond to an RDS client request.

### Description

This command is used to return information about a Calibre RDS (*processObj*) in a cluster snapshot. The command arguments determine the type of information returned about the remote

process. The “**nemo::master remotedata**” statement returns information about the connected RDS host, node, and the number of send and receive data transactions between the primary host and the RDS host.

## Examples

```
foreach data [nemo::master remotedata $hdb] {  
    set node [nemo::remotedata node $data]  
    puts "RDS_NODE: [nemo_node::host_name $node]"  
    puts "RDS_SEND_BLOCKS: [nemo::remotedata rds_send_blocks $data]"  
    puts "RDS_RECV_BLOCKS: [nemo::remotedata rds_recv_blocks $data]"  
}
```

## nemo::singleton

Evaluate a Tcl script or Tcl file using the singleton interpreter.

### Usage

**nemo::singleton evalscript *tcl\_script***

**nemo::singleton evalfile *tcl\_file***

### Arguments

- **evalscript *tcl\_script***

Evaluate an inline Tcl script in singleton interpreter. This is equivalent to calling the Tcl **eval** command in singleton interpreter.

- **evalfile *tcl\_file***

Evaluate Tcl file in singleton interpreter. This is equivalent to calling the Tcl **source** command in singleton interpreter.

### Description

This command allows you to evaluate a Tcl script or Tcl file using the singleton interpreter, a global CalCM Tcl interpreter.

This interpreter supports Tk and other supported graphics packages (for example, BLT), unlike the other interpreters that are used for CalCM applications.

### Examples

```
# define plot function
nemo::singleton evalfile [file join $lib_path plot_lib.tcl]
# plot specified data and save to specified file
nemo::singleton evalscript "plot $data $image_file"
```

## nemo::snapshot

Analyze the cluster snapshot.

### Usage

**nemo::snapshot timestamp *snapshotObj***

**nemo::snapshot jobs *snapshotObj***

**nemo::snapshot nodes *snapshotObj***

### Arguments

- **timestamp *snapshotObj***

Returns the timestamp of the snapshot as an integer number of seconds from the “Epoch.”  
The timestamp indicates when the snapshot was last updated.

- **jobs *snapshotObj***

Returns a list of active Calibre jobs running on the cluster (see [nemo::job](#)).

- **nodes *snapshotObj***

Returns a list of known cluster nodes (see [nemo::node](#)).

### Description

This statement is used to analyze the cluster snapshot. A *cluster snapshot* is a collection of consistent information about the state of a cluster at a certain moment in time. The snapshot object is passed to the application using the “[nemo::application update\\_status](#)” statement or the “[nemo::control acquire\\_snapshot](#)” statement.

### Examples

```
#  
# printClusterNodes --  
#  
# Prints names of all known cluster nodes to stdout.  
#  
proc printClusterNodes {} {  
    set snapshotObj [nemo::control acquire_snapshot]  
    foreach nodeObj [nemo::snapshot nodes $snapshotObj] {  
        puts [nemo::node host_name $nodeObj]  
    }  
    # No need to release snapshot, it will be released automatically.  
}
```

# Appendix G

## Configuring and Using the CalCM TAT Application

---

This section includes procedures that explain how to set up and run the TAT application with CalCM.

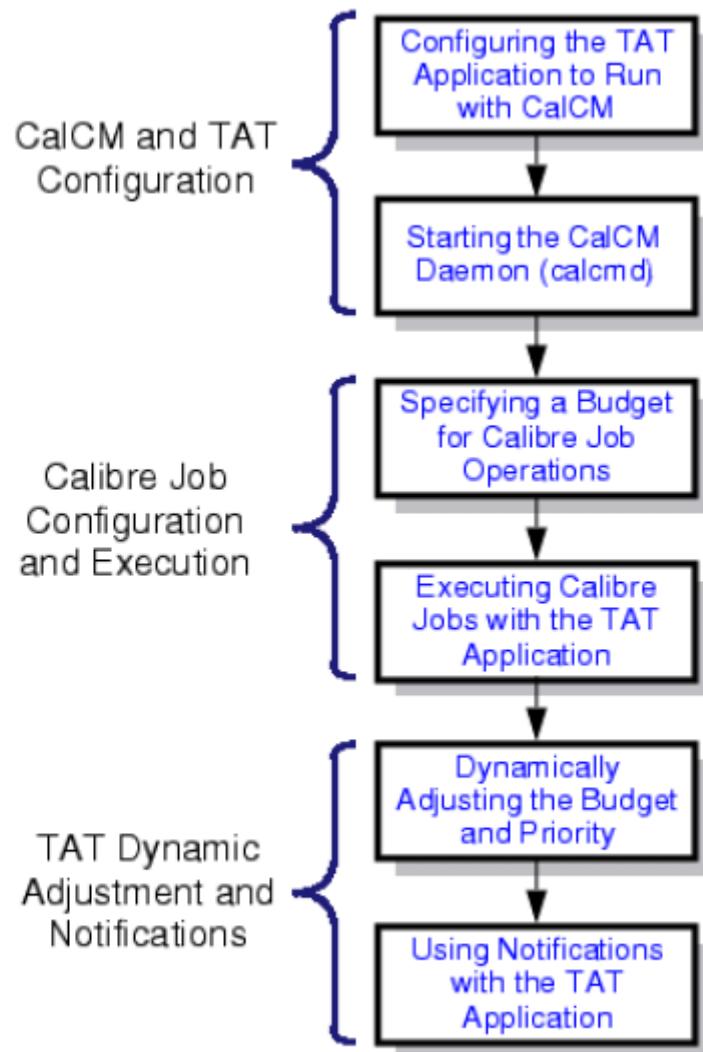
Refer to “[CalCM TAT](#)” on page 37 for the concepts related to using this application.

<b>CalCM TAT Workflow .....</b>	<b>363</b>
<b>Configuring the TAT Application to Run with CalCM .....</b>	<b>364</b>
<b>Starting the CalCM Daemon .....</b>	<b>366</b>
<b>Specifying a Budget for Calibre Job Operations .....</b>	<b>367</b>
<b>Executing Calibre Jobs with the TAT Application.....</b>	<b>369</b>
<b>Dynamically Adjusting the Budget and Priority .....</b>	<b>372</b>
<b>Using Notifications with the TAT Application .....</b>	<b>373</b>

## CalCM TAT Workflow

This section identifies the workflow for setting up and running the TAT application with CalCM. Each block in the workflow is a link to a procedure in this chapter.

**Figure G-1. Workflow for Running TAT with CalCM**



## Configuring the TAT Application to Run with CalCM

This procedure describes how to configure the TAT application to run with CalCM by creating a job attribute file and defining the TAT configuration variables.

### Prerequisites

- The environment is configured according to the process described in “[Configuring the CalCM Environment](#)” and jobs are configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#).”

## Procedure

Edit the CalCM configuration file (*calcmd.conf*) located in the *\$CALCM\_HOME/config* directory.

- a. Verify the CalCM server and license statements are correctly defined for your environment. Refer to “[Configuring the CalCM Environment](#)” on page 45” for information.
- b. Define the following arguments for the TAT application:
  - o DBPATH — Specify the path to the TAT database file. The default is *calcmtat.db*.
  - o MAX\_REMOTE — Specify the maximum number of remotes that the TAT application can request per job. This value must be larger than the MIN\_REMOTE value but should not be larger than the REMOTE MAX value specified in the job configuration file. The default value is 10000.
  - o MIN\_REMOTE — Specify the minimum number of remotes that the TAT application must connect to before launching a job. This value should not be smaller than the REMOTE MIN value specified in the job configuration file. The default value is 10.

For example:

```
// "Job Queue" Tcl application.  
APPLICATION TCL $CALCM_APP/calcmtat_app.tcl FILE [  
    ...  
]  
  
APPLICATION TCL $CALCM_APP/calcmtat_app.tcl FILE [  
    DBPATH = $CALCM_HOME/dbfiles/calcmtat.db  
    MAX_REMOTE = 1000  
    MIN_REMOTE = 200  
    ADJUST_BUDGET = true  
]  
  
// "Resource Manager" Tcl application.  
APPLICATION TCL $CALCM_APP/calcrmrmanager_app.tcl FILE [  
    ...  
]
```

- c. In order to use automatic budgeting mode, as described in “[Specifying a Budget for Calibre Job Operations](#)” on page 367, you must also define the SCALE\_OPS argument to identify the scalable operations for this mode. For example:

```
APPLICATION TCL $CALCM_APP/calcmtat_app.tcl FILE [  
    ...  
    SCALE_OPS = "LITHO OPCBIAS OPCSBAR"  
    ...  
]
```

When you run CalCM with the TAT application, the *calcm\_tat\_app.tcl* file looks in the CalCM configuration file for definitions for these arguments. If the arguments are not defined, then the default values defined by the application are used for the job. Refer to the [calcm\\_tat\\_app.tcl](#) reference page for more information on these arguments and their defaults.

## Starting the CalCM Daemon

This procedure describes how to start the CalCM daemon and view job information in a browser using the CalCM dashboard web application.

### Prerequisites

- CalCM TAT is configured according to the steps in “[Configuring the TAT Application to Run with CalCM](#).”
- You must know the server name on which you will run the CalCM daemon and the server port (defined in previous procedure) used to communicate with the daemon.
- Browser access.

### Procedure

1. Set your working directory to `$CALCM_HOME` and start the CalCM daemon.

```
$MGC_HOME/bin/calcmd -config config/calcmd.conf
```

The CalCM daemon startup process creates the `calcm_submit_job` script in the *jobqueue* directory. This script can be used by the end users to launch Calibre jobs under CalCM.

Upon starting, the CalCM daemon generates a transcript which is captured in `$CALCM_HOME/calcmd.log`. Refer to “[CalCM Daemon Transcript Example](#)” on page 291 for an example of this log file.

2. Open a browser and, using the following format, enter the server name on which the CalCM daemon is running and the server port specified in the CalCM configuration file (`calcmd.conf`).

```
http://server_name:server_port
```

The CalCM dashboard web application shown in [Figure G-2](#) displays the Active Jobs page with status and resource information.

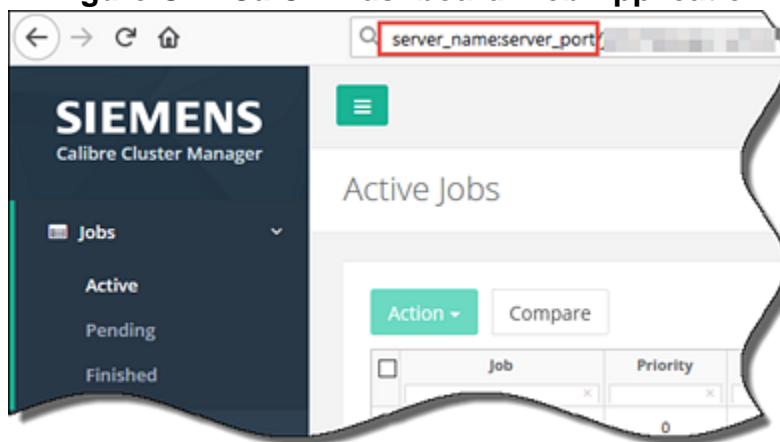
---

#### Note

 The CalCM dashboard displays a web login page requesting authentication information (user name and password). Some menus and actionable items are only accessible at certain user levels. For information on user authentication in the CalCM dashboard, see the LDAP-related configuration keywords in the [calcm\\_http\\_server\\_app.tcl](#).

---

**Figure G-2. CalCM Dashboard Web Application**



When Calibre jobs are launched, the browser updates to display the active and pending jobs. For more information on using the CalCM dashboard, refer to “[Using the CalCM Dashboard Web Application](#)” on page 54.

## Results

The CalCM daemon is now running. It is recommended that you start a backup CalCM daemon as described in “[Running CalCM with a Backup Daemon](#)” on page 106.

# Specifying a Budget for Calibre Job Operations

The following procedure describes how to create job configuration files that support manual and automatic budgeting modes for Calibre operations.

## Prerequisites

- Jobs should be configured as described in “[Configuring Calibre Jobs to Run Under CalCM](#).”
- ADJUST\_BUDGET must be enabled (default) in `calcm_tat_app.tcl`.
- For automatic budgeting mode, the SCALE\_OPS argument in the CalCM configuration file must identify the scalable operations for this mode.

## Procedure

1. Edit the job configuration file (`job.conf`) for job1 located in the `$CALCM_HOME/jobqueue/job1` directory.
  - a. Define the **JOB BUDGET** statement for your particular operations.

For example, you may want to define a budget of 80 minutes for dense OPC operations, 25 minutes for OPCverify operations, and use the \_EXTRA\_ keyword to specify a budget of 5 minutes for the remaining non-LITHO operations:

```
JOB BUDGET dens_out=80;opcv_out=25;_EXTRA_=5
```

- b. You can use the **JOB TAT\_PRIORITY\_BUMP** statement to define a higher priority for job1. For example:

```
JOB TAT_PRIORITY_BUMP 20
```

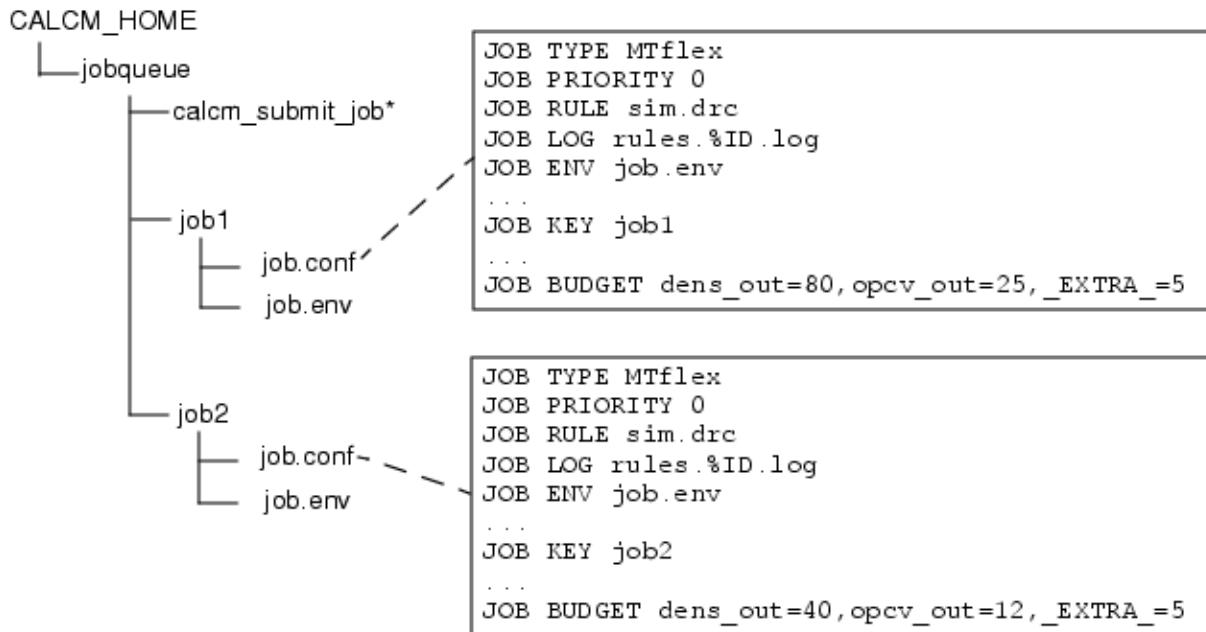
The default priority bump value (10) is a global value defined by the [calcm\\_tat\\_app.tcl](#) application.

- c. You can optionally specify a **JOB KEY**.

You use this statement to assign a unique job key, or alias, to any Calibre job that is run using the TAT application. For example:

```
JOB KEY nmopc_job1
```

**Figure G-3. CalCM Job Configuration File**



\* The `calcm_submit_job` script is automatically created by the CalCM job queue application.

- d. Save and close the job configuration file.
2. Edit the job configuration file (*job.conf*) for job2 located in the `$CALCM_HOME/jobqueue/job2` directory.

- a. Define the **JOB BUDGET** statement for automatic budgeting mode. To enable this mode, you must use the “\_AUTO\_” keyword and specify a total budget for the job. For example:

```
JOB BUDGET _AUTO_;TOT:400
```

In addition, the SCALE\_OPS argument in the CalCM configuration file must identify the scalable operations for automatic budgeting mode (see “[Configuring the TAT Application to Run with CalCM](#)”).

Only the operations specified by SCALE\_OPS argument in the CalCM configuration file are distributed across multiple CPU cores.

- b. You can optionally specify a **JOB KEY** for job2.
- c. Save and close the job configuration file.

## Results

You now have two jobs, job1 and job2, that are configured to run with the CalCM TAT application. Job1 is set up to run in manual budgeting mode, while job2 is set up to run in automatic budgeting mode. During the Calibre run, TAT dynamically adjusts the budget for an operation when:

- ADJUST\_BUDGET is enabled.
- The remaining job duration exceeds the accumulated budget.

Dynamically adjusting the budget for an operation ensures the job will meet the specified job budget. The budget adjustment is achieved through database queries that continuously update the job budget based on the status of the current job and previously-collected job data.

## Related Topics

[JOB BUDGET](#)

[JOB KEY](#)

# Executing Calibre Jobs with the TAT Application

This procedure describes how to submit jobs that run under CalCM TAT and use the budget defined for the jobs.

## Prerequisites

- The “progress meter” feature, supported by various Calibre setlayer operations, must be enabled. The progress meter provides metrics that are used by CalCM TAT for analysis and projection of runtime.

For applications, such as Calibre OPCVerify, the progress meter is enabled or disabled by defining the progress\_meter command in a Litho setup file. Refer to the appropriate application documentation for information on using the progress meter.

- The CalCM daemon (calcld) must be running as described in “[Starting the CalCM Daemon](#)” on page 366”.
- Job configuration file.

## Procedure

1. If you have not already done so, open a browser and enter the server name on which the CalCM daemon is running and the server port specified in the CalCM configuration file (*calcld.conf*). This displays the CalCM dashboard web application. The default view is the Active Jobs page in the **Jobs** menu item on the left side of the displayed web page.

---

### Note

 The CalCM dashboard displays a web login page requesting authentication information (user name and password). Some menus and actionable items are only accessible at certain user levels. For information on user authentication in the CalCM dashboard, see the LDAP-related configuration keywords in the [calcm\\_http\\_server\\_app.tcl](#).

---

2. Set your working directory to *CALCM\_HOME/jobqueue*.

This directory should contain the calcld\_submit\_job script, which is automatically created when you start the CalCM daemon.

3. Enter the following command to submit job1:

```
./calcld_submit_job $PWD/job1/job.conf
```

In the **Jobs** menu item of the browser window, select the **Pending Jobs** menu option. You should see job1 appear in this category. Depending on various factors, such as the job configuration or resource status, it may take a few minutes for the browser to update and display the job.

4. Enter the following command to submit job2:

```
./calcld_submit_job $PWD/job2/job.conf
```

In the **Jobs** menu item of the browser window, job2 should appear in the Pending Jobs or Active Jobs category. Depending on various factors, such as the job configuration or resource status, it may take a few minutes for the browser to update and display the job.

**Figure G-4. Active Jobs View**

Action	Job	Priority	User	Start	Elapsed	Mode	Status	Master	Remotes	Active Remotes	Allocated Remotes
	464	0	shawn	09/15/21 19:26:14	00:01:01	HYPER2_M...	START/AN...		5/12	14	12
	463	0	shawn	09/15/21 19:20:22	00:04:53	HYPER2_M...	START/AN...		12/23	12	23
	462	0	shawn	09/15/21 19:18:27	00:06:48	HYPER2_M...	START/AN...		20/29	20	20
	461	0	shawn	09/15/21 19:17:18	00:07:57	HYPER2_M...	START/AN...		6/21	0	21

- Click the job ID in the first column of the Active Jobs page to see current information for the job.

**Figure G-5. Job Detail View**



## Results

When you run the Calibre job, the CalCM TAT application uses information collected by the progress meter to calculate the following for the job:

- Computes the estimated total remaining time.
- Computes the CPU demand using the time left (remaining budget), estimated total CPU time, and queue length.
- Computes a priority demand.

The TAT application uses the collected information to:

- Schedule an execution order that minimizes resource demand conflicts.

- Adjust the job priority to control the execution order of operations and to give more resources to jobs with a higher possibility of missing its budgeted time.

## Dynamically Adjusting the Budget and Priority

This procedure describes how to use the TAT calcm\_send\_message and set\_priority\_bump commands to dynamically adjust the budget and priority for a job running under CalCM.

### Prerequisites

- The **CALIBRE\_ENABLE\_SETLAYER\_ERT** environment variable must be set.
- The jobs you submitted in the procedure “[Executing Calibre Jobs with the TAT Application](#)” on page 369 must be running.

### Procedure

1. Verify the current job budget that is defined in the job configuration file.

```
JOB BUDGET dens_out=80,opcv_out=25,_EXTRA_=5
```

You can also review the transcript for this host:

```
Received connection request from calmisc:41987
: host:41987: announce_job host:41985:41986 id="1" resource=""
    key="opc_job1" flag=0 quota="DEPT2/jsmith"
cluster="1:2:500:500:2"
    budget="dens_out=80,opcv_out=25,_EXTRA_=5"
    directory=<path>/CALCM_HOME/jobqueue/job1" rule="sim.drc"
    priority=0 log="rules.1.log" tat_priority_bump="20"
mode="MTFLEX"
    start=1340225222 user="jsmith"
    Job "1" (calmisc:41985) is announced.
    ...
```

2. Use the **set\_budget** messaging command with the calcm\_send\_message script to change the budget of job1. The syntax is:

```
calcm_send_message set_budget job_id budget_string
```

For example:

```
calcm_send_message set_budget 1 "dens_out=100,opcv_out=30, /
    _EXTRA_=30"
```

3. Use the **set\_priority\_bump** messaging command with the calcm\_send\_message script to change the priority of job2. The syntax is:

```
set_priority_bump job_id priority_bump
```

For example:

```
calcm_send_message set_priority_bump 2 40
```

The CalCM daemon transmits the following message in return:

```
...
Received connection request from <host>:46280
<host>:46280: set_priority_bump 2 40
TAT: Set new priority bump 40 to job '2'
Connection with <host>:46280 closed
...
```

4. CalCM regularly performs an application update to collect information on the current status, calculate whether any adjustments need to be made to current jobs, and implement those adjustments. On the next update cycle, CalCM adjusts the resources for job 2 based on the new priority.

```
--- APP UPDATE (Tue May 22 15:31:28 2012) ---
```

## Related Topics

[set\\_budget](#)

[set\\_priority\\_bump](#)

# Using Notifications with the TAT Application

In the following procedure, you configure TAT to support the notification of runtime events.

## Prerequisites

- CalCM configuration file (*calcmd.conf*)
- Job configuration file (*job.conf*)
- CalCM daemon is running

## Procedure

1. Edit the CalCM configuration file (*calcmd.conf*) located in the *config* directory to add support for estimation exceed and runtime exceed events.

- a. Add the lines shown in bold to the configuration file:

```
...
APPLICATION TCL $CALCM_APP/calcm_tat_app.tcl FILE [
    DBPATH = $CALCM_HOME/dbfiles/calcm_tat.db
    MAX_REMOTE = 1000
    MIN_REMOTE = 200
NOTIFICATION_FILE_PATH = $CALCM_HOME/notify.sh
    NOTIFICATION_EXCEED_RATIO = 30
    NOTIFICATION_INTERVAL = 5
    NOTIFICATION_ESTATUS = 1
]
...
```

The NOTIFICATION\_FILE\_PATH argument specifies the path to a notification script (*notify.sh*) that you create in step 2. The NOTIFICATION\_EXCEED\_RATIO argument specifies a ratio for detecting the conditions under which to issue a notification event. The NOTIFICATION\_INTERVAL argument specifies a minimum time interval for issuing a notification event.

- b. Save and close the file.
2. Create a *notify.sh* script in the \$CALCM\_HOME directory.
  - a. In an ASCII editor, add the following:

```
#!/bin/csh

if ($1 == "CHECK") then
    # This is CHECK event
    set mbody="CHECK point error, ID:$2, Operation:$3, Time:$4,
Duration:$5"
    echo "$mbody" | mail -s "$mbody" calcm_admin@company.com
else if ($1 == "ESTATUS") then
    # This is ESTATUS event
    set mbody="ESTATUS event, ID:$2, Estatus:$3, Start:$4, End:$5"
    echo "$mbody" | mail -s "$mbody" calcm_admin@company.com
endif
```
  - b. Save and close the file.
3. Edit the job configuration file (*job.conf*) located in the *job1* directory to add support for a check point event.
  - a. Modify the JOB BUDGET statement to include the CHECK argument. For example:

```
JOB BUDGET CHECK:dens_out=80;opcv_out=25
```

The CHECK argument can be specified multiple times in a JOB BUDGET string or in other JOB BUDGET statements.
  - b. Save and close the file.
4. Execute *job1* under CalCM.

5. Review the transcript for any “CHECK”, “ESTIMATE”, or “EXCEED” notifications.

## Results

During the run, notifications are sent when the following events occur:

- Check point — This event occurs when the dens\_out or opcv\_out operation do not finish within the time specified by the JOB BUDGET statement. The following parameters are passed to the notification script (notify.sh) upon execution:

```
"CHECK" job_id operation_name specified_time job_duration quota user
```

- Estimation exceed — This event occurs when the estimated job duration exceeds the budget by more than the ratio specified by the NOTIFICATION\_EXCEED\_RATIO argument (30%). In this procedure, the budgeted time for the dens\_out procedure is 80 minutes and the opcv\_out procedure is 25 minutes. An estimation exceed notification is sent when the estimated duration for the dens\_out procedure exceeds 104 ( $80 + 80 \times 0.3$ ) minutes or the opcv\_out procedure exceeds 32.5 ( $25 + 25 \times 0.3$ ) minutes. The following parameters are passed to the notification script (notify.sh) upon execution:

```
"ESTIMATE" job_id operation_name exceed_time budget job_duration quota user
```

- Runtime exceed — This event occurs when the actual job duration exceeds the total budget by more than the ratio specified by the NOTIFICATION\_EXCEED\_RATIO argument. In this procedure, the total budget is 105 ( $80 + 25$ ) minutes. A runtime exceed notification is sent when the job duration exceeds the budget of 136.5 ( $105 + 105 \times 0.3$ ) minutes. The following parameters are passed to the notification script (notify.sh) upon execution:

```
"EXCEED" job_id operation_name exceed_time budget job_duration quota user
```

- Exit status — This event occurs when the CalCM job finishes. In this procedure, a value of “1” is specified for the NOTIFICATION\_ESTATUS argument which causes an “exit status” event to occur regardless of the exit status. The following parameters are passed to the notification script (notify.sh) when the job completes with a non-zero exit status:

```
"ESTATUS" job_id exit_status start_time end_time quota user
```

## Related Topics

[JOB BUDGET](#)

[calcm\\_tat\\_app.tcl](#)



# **Appendix H**

## **CalCM Supported Platforms**

---

CalCM supports various platform tools by providing methods to integrate with different platform and system configurations.

For CalCM platform customization to your environment, refer to your Siemens representative for assistance.

<b>Customizing CalCM for Remote Shell. . . . .</b>	<b>378</b>
<b>Platform Tool Job and Remotes Control. . . . .</b>	<b>382</b>
<b>Using MySQL in CalCM. . . . .</b>	<b>386</b>
<b>Upgrading the CalCM Database . . . . .</b>	<b>387</b>

## Customizing CalCM for Remote Shell

---

You can specify the remote shell settings that CalCM uses to run Calibre jobs and Calibre remotes.

The default platform configuration for CalCM is to control Calibre jobs with a platform tool and remotes with remote shell. For remote shell users, CalCM uses the Linux secure shell (SSH) by default.

<b>Remote Shell Considerations in CalCM .....</b>	<b>378</b>
<b>Specifying the RSH Remote Shell in CalCM.....</b>	<b>378</b>
<b>Setting Up SSH Connectivity Without Password Authentication.....</b>	<b>379</b>

## Remote Shell Considerations in CalCM

CalCM is configured by default to run Calibre remotes using the SSH utility. This avoids the connections limits imposed by the remote shell (RSH) utility.

The RSH or rlogin utility only uses trusted or privileged ports for outgoing communication. These ports have port numbers that are below 1024. In a typical RSH configuration, port numbers 513-1023 are used. By using this port numbering scheme, the maximum number of connections through RSH is limited. For this reason, CalCM is configured to run Calibre remotes using SSH protocol by default, rather than RSH, because SSH does not have this limitation.

---

**Note**

 CalCM runs with Calibre remotes should use an SSH authentication key instead of password access. See “[Setting Up SSH Connectivity Without Password Authentication](#)” on page 379

---

### Related Topics

[Calibre Administrator’s Guide \[Calibre Administrator’s Guide\]](#)

[Specifying the RSH Remote Shell in CalCM](#)

## Specifying the RSH Remote Shell in CalCM

You can optionally specify for CalCM to use the RSH remote shell utility instead of the default SSH remote shell utility.

---

**Note**

 The RSH utility imposes certain connection limits when running CALCM with Calibre jobs. See “[Remote Shell Considerations in CalCM](#)” on page 378.

---

## Prerequisites

- You have knowledge of your existing network configuration used to run CalCM and launch Calibre jobs, including authentication procedures, networking concepts, Linux commands, and Tcl basics.
- You have a have a CalCM configuration file (*calcmd.conf*) available as described in “[CalCM Configuration File \(calcmd.conf\)](#)” on page 26.
- You have a *calcめ\_application\_lib.platform* (platform include script) created for your platform environment.

## Procedure

1. Specify the “CALCM\_PLATFORM\_INCLUDE” environment variable in the CALCM configuration file as follows:

```
// CalCM configuration file
VARIABLE CALCM_HOME /home/calcめ
VARIABLE CALCM_APP $CALCM_HOME/app
VARIABLE $CALCM_PLATFORM_INCLUDE calcめ_application_lib.platform
```

2. Set the “rsh” and “rshFlags” Tcl variables settings in the CalCM platform include file:

```
// CalCM platform include file
set rsh "rsh"
set rshFlags "-n"
```

## Related Topics

[Remote Shell Considerations in CalCM](#)

[Configuring the CalCM Environment](#)

# Setting Up SSH Connectivity Without Password Authentication

You can configure an SSH remote shell authentication key that enables CalCM to connect to other hosts without a manually entered password.

## Prerequisites

- You have knowledge of your existing network configuration used to run CalCM and launch Calibre jobs, including authentication procedures, networking concepts, Linux commands, and Tcl basics.
- You have setup the CalCM configuration files, the platform include script, and variables specifying your environment.

```
// CalCM configuration file
VARIABLE CALCM_HOME /home/calcm
VARIABLE CALCM_APP $CALCM_HOME/app
VARIABLE $CALCM_PLATFORM_INCLUDE calcm_application_lib.platform
```

## Procedure

1. Login to the machine to be used for running the CalCM daemon. This machine must have a network connection to all Calibre primary and remote hosts.
2. Check that your home directory permissions are set to 755. This means that you, the owner, can read, write, and execute (r-w-x) files in your home directory, and all other user groups have read and execute permissions (r-x) only.
3. Verify that the machine you are using has a `~/.ssh` directory; otherwise create the directory, and set the permissions to 700 (owner r-w-x) as follows:

```
% mkdir ~/.ssh
% chmod 700 ~/.ssh
```

4. Check that you already have a public/private authentication key (`~/.ssh/id_rsa.pub`), if not, generate the key as follows:

```
% cd ~/.ssh
% ssh-keygen -t rsa
```

Accept the default values, and hit “enter” on your keyboard without entering a pass-phrase, since you are setting up password-less access.

5. Enable your login machine to trust the connection to another host without manually entering your password by copying your public SSH key into the `~/.ssh/authorized_keys` file as follows:

```
% cd ~/.ssh
% cat id_rsa.pub >> authorized_keys
% chmod 600 authorized_keys
```

6. Enable your login machine to trust all the hosts in your cluster. If this is not setup, SSH prompts you to accept the new host key each time there is a connection to a new or upgraded host.

```
% echo "StrictHostKeyChecking no" >> ~/.ssh/config
% chmod 600 ~/.ssh/config
```

Each new host key is automatically added to the `~/.ssh/known_hosts` file, and changed host keys are replaced silently.

7. Check your SSH configuration, and ensure that all files have the correct access permissions so that SSH does not refuse the password-less access.

```
% cd ~/.ssh
% ls -la
drwx----- 2 <login> <group> 4096 Jul 21 03:14 .
-rw----- 1 <login> <group> 813 Jun 13 2016 authorized_keys
-rw----- 1 <login> <group> 395 Apr 11 08:23 config
-rw----- 1 <login> <group> 1675 May 20 2016 id_rsa
-rw-r--r-- 1 <login> <group> 400 May 20 2016 id_rsa.pub
-rw----- 1 <login> <group> 125428 Jul 20 06:30 known_hosts
```

8. You can test your password-less SSH setup by using SSH to connect to another machine and running a simple command as follows:

```
% ssh <remote_machine_name> uname -n
```

When SSH connects to the machine, you are not prompted to manually enter a password or pass-phrase, and the remote host name is returned at the terminal prompt.

# Platform Tool Job and Remotes Control

---

CalCM supports IBM® Platform™ LSF® (LSF) and Univa® Grid Engine® (Grid) platform tools for controlling Calibre jobs and remotes. One method is to control the Calibre jobs with the platform tool and the remotes with remote shell. The other method is to control both the Calibre jobs and the remotes with the platform tool.

<b>Using LSF to Control Jobs Only .....</b>	<b>382</b>
<b>Using LSF to Control Jobs and Remotes.....</b>	<b>383</b>
<b>Using Grid to Control Jobs Only .....</b>	<b>384</b>
<b>Using Grid to Control Jobs and Remotes .....</b>	<b>385</b>

## Using LSF to Control Jobs Only

CalCM supports the LSF platform tool to control Calibre jobs only. Remotes are launched with remote shell.

In this scenario, CalCM Job Queue application prepares the run scripts needed for the execution, and LSF executes the actual Calibre jobs. The ordered steps of this process are as follows:

1. The LSF launch script (calcm\_run.lsf) is submitted to an LSF queue using the bsub command.
2. LSF executes the calcm\_run.lsf script when a host is available.
3. The calcm\_run.lsf script submits a job to CalCM using calcm\_submit\_job and waits until the run scripts are prepared.
4. The CalCM Job Queue application prepares the run scripts (calcm\_run, calcm\_announce.tcl, calcm\_launch.tcl, and calcm\_mtflex.conf).
5. The calcm\_run script is executed under the user's context.

You can disable the CalCM Job Queue that invokes the Calibre jobs by specifying the “CALCM\_PLATFORM” environment variable as “XSH\_RSH.”

The number of digits in the job ID (including alphabetic characters) can exceed the default, so the job ID should be adjusted using GUIDDIGIT as follows in the CalCM configuration file:

```
// CalCM configuration file
VARIABLE $CALCM_PLATFORM XSH_RSH
APPLICATION TCL calcm_jobqueue_app.tcl FILE [
    GUIDDIGIT = 7
    ...
]
```

To specify the total number of jobs, use “\_PLATFORM\_” in “MASTER HOST.”

```
//-----  
// MASTER CONFIGURATION  
//-----  
MASTER HOST _PLATFORM_ SLOT 10  
//-----  
// REMOTE CONFIGURATION  
//-----  
REMOTE HOST node800 SLOT 4  
REMOTE HOST node802 SLOT 4
```

## Using LSF to Control Jobs and Remotes

CalCM supports the LSF platform tool to control both Calibre jobs and remotes.

The scenario for using LSF to control jobs and remotes is similar to “Using LSF to Control Jobs Only” in launching Calibre jobs. However, the difference is that the remotes are also launched using LSF instead of remote shell.

You can disable the CalCM Job Queue that invokes Calibre jobs and launches the remotes by specifying the “CALCM\_PLATFORM” environment variable as “XSH” and specifying “CALCM\_PLATFORM\_INCLUDE” to set the LSF specific commands and flags. To specify the total number of jobs and remotes, use “\_PLATFORM\_” in “MASTER HOST” and “REMOTE HOST.”

In this platform configuration, the number of jobs specified in “MASTER HOST” is used for display in calcm\_rmanager\_app.tcl, and the control is through LSF.

---

### Note

 The LSF job name and the group name are used to keep track of remotes. Remotes allocated in other calcmd instances can be counted. To avoid conflict, change the job and group names with env and calcHostname.

---

The number of digits in the job ID (including alphabetic characters) can exceed the default, so the job ID should be adjusted using GUIDDIGIT as follows in the CalCM configuration file:

```
// CalCM configuration file  
VARIABLE $CALCM_PLATFORM XSH  
VARIABLE $CALCM_PLATFORM_INCLUDE $CALCM_APP/calcapplication_lib.lsf  
APPLICATION TCL calcjobqueue_app.tcl FILE [  
    GUIDDIGIT = 7  
    ...  
]
```

To specify the total number of jobs, use “\_PLATFORM\_” in “MASTER HOST”.

```
//-----  
// MASTER CONFIGURATION  
//-----  
MASTER HOST _PLATFORM_ SLOT 10  
//-----  
// REMOTE CONFIGURATION  
//-----  
REMOTE HOST _PLATFORM_ SLOT 200
```

The JOB FLAG specification in the CalCM job configuration file:

```
JOB FLAG 1
```

## Using Grid to Control Jobs Only

CalCM supports the Grid platform tool to control Calibre jobs only. Remotes are launched with remote shell.

In this scenario the CalCM Job Queue application prepares the run scripts needed for the execution and the Grid platform executes the Calibre jobs. The ordered steps are as follows:

1. The LSF launch script (calcm\_run.grid) is submitted to a Grid queue using the bsub command.
2. Grid executes the calcm\_run.grid script when a host is available.
3. The calcm\_run.grid script submits a job to CalCM using calcm\_submit\_job and waits until the run scripts are prepared.
4. The CalCM Job Queue application prepares the run scripts (calcm\_run, calcm\_announce.tcl, calcm\_launch.tcl, and calcm\_mtflex.conf).
5. The calcm\_run script is executed under the user's context.

You can disable the CalCM Job Queue that invokes Calibre jobs by specifying the “CALCM\_PLATFORM” environment variable as “XSH\_RSH” To specify the total number of jobs, use “\_PLATFORM\_” in “MASTER HOST” and “REMOTE HOST.”

The number of digits in the job ID (including alphabetic characters) can exceed the default, so the job ID should be adjusted using GUIDDIGIT as follows in the CalCM configuration file:

```
// CalCM configuration file  
VARIABLE $CALCM_PLATFORM XSH_RSH  
APPLICATION TCL calcm_jobqueue_app.tcl FILE [  
    GUIDDIGIT = 8  
    ...  
]
```

To specify the total number of jobs, use “\_PLATFORM\_” in “MASTER HOST.”

```
//-----  
// MASTER CONFIGURATION  
//-----  
MASTER HOST _PLATFORM_ SLOT 10  
//-----  
// REMOTE CONFIGURATION  
//-----  
REMOTE HOST node800 SLOT 4  
REMOTE HOST node802 SLOT 4
```

## Using Grid to Control Jobs and Remotes

CalCM supports the Grid platform tool to control both Calibre jobs and remotes.

The scenario for using Grid to control jobs and remotes is similar to “Using Grid to Control Jobs Only” in launching Calibre jobs. However, the difference is that the remotes are also launched using Grid instead of remote shell.

In this platform configuration, the number of jobs specified in “MASTER HOST” is used for display in calcm\_rmanager\_app.tcl, and the control is through Grid.

---

### Note

 The Grid job name and the group name are used to keep track of remotes. Remotes allocated in other calcmd instances can be counted. To avoid conflict, change the job and group names with env and calcHostname.

---

The number of digits in the job ID (including alphabetic characters) can exceed the default, so the job ID should be adjusted using GUIDDIGIT as follows in the CalCM configuration file:

```
// CalCM configuration file  
VARIABLE $CALCM_PLATFORM XSH  
VARIABLE $CALCM_PLATFORM_INCLUDE $CALCM_APP/calc_m_application_lib.grid  
APPLICATION TCL calc_m_jobqueue_app.tcl FILE [  
    GUIDDIGIT = 8  
    ...  
]
```

To specify the total number of jobs, use “\_PLATFORM\_” in “MASTER HOST.”

```
//-----  
// MASTER CONFIGURATION  
//-----  
MASTER HOST _PLATFORM_ SLOT 10  
//-----  
// REMOTE CONFIGURATION  
//-----  
REMOTE HOST _PLATFORM_ SLOT 200
```

## Using MySQL in CalCM

CalCM administrators can configure CalCM to use MySQL database management systems.

By overriding the database query routines, MySQL can be used instead of the default database application SQLite®<sup>1</sup>. When CalCM is used with MySQL functionality, certain configurations and recommendations apply.

For detailed information on MySQL database configuration and usage, refer to “Guidelines on Setting up MySQL for CalCM” accessible from KB article #580610 on Support Center at:

<https://support.sw.siemens.com/en-US/knowledge-base/MG580610>

The DBPATH configuration variable is specified using the following format:

```
"<mysql_login>:<mysql_password>@<MySQL_server_hostname>/<mysql_schema>"
```

The environment variable statements for DBPATH and CALCM\_PLATFORM\_INCLUDE are specified and passed to the CalCM Tcl applications in the CalCM configuration file. For example,

```
// CalCM configuration file
VARIABLE DBPATH "<mysql_login>:<mysql_password>@<MySQL_server_hostname>/
<mysql_schema>"
APPLICATION TCL ... FILE [
    DBPATH = $DBPATH
    ...
]
```

It is recommended that for MySQL server version 5.7.8 and above, the server configuration variable “max\_execution\_time” be set to 600000. For MySQL server version 5.7.4 and above but before 5.7.8, the “max\_statement\_time” should be set to 600000. The values are in milliseconds and prevent long database queries from experiencing timeouts.

It is also recommended to use InnoDB as the default storage engine for CalCM. This is the default storage engine for MySQL 5.5.5 and later. Using MyISAM instead of InnoDB may create slowdowns during MySQL dump and restore.

---

1. SQLite® is a registered trademark of Hipp, Wyrick & Company, Inc.

# Upgrading the CalCM Database

You can use the features provided by the `convert_database` shell command for upgrading the CalCM database for migration.

The database generated in a previous CalCM version can be different from the latest schema structure. The conversion of the database is done automatically during the startup of the CalCM daemon. However, in some cases it may be preferable if the backup database is not converted automatically because of performance reasons. To migrate an existing database stand-alone, the `convert_database` shell command is recommended.

<b>Implementing convert_database .....</b>	<b>387</b>
<b>convert_database .....</b>	<b>388</b>
<b>Schema File Example. ....</b>	<b>390</b>

## Implementing convert\_database

You can convert an existing database to the schema defined for the CalCM application using a Tcl script that can be implemented as a shell command.

### Prerequisites

- Applicable licenses for CalCM and Calibre products. Refer to the *Calibre Administrator's Guide* for licensing information.
- Access to the location of a Calibre software tree with CalCM Tcl applications encrypted in the MGC\_HOME package (from Calibre version 2019.1 forward).
- An existing or backup database that you want to migrate.

### Procedure

- In an ASCII editor, create a shell script “`convert_database`” and add the following statements (in C-shell, for example):

```
#!/bin/csh -f
setenv MGC_HOME your path to calibre build
setenv CALCM_TCL_DIR $MGC_HOME/pkgs/icv_calcm/tcl

$MGC_HOME/bin/tclsh \
$CALCM_TCL_DIR/calcm_convert_database.tcl \
-app_dir $CALCM_TCL_DIR $argv[*]
```

- Execute the `convert_database` shell command in a Linux terminal window.

```
./convert_database -source existing.db -dest converted.db
```

If both source and destination databases are specified, the data is migrated from the source database into the destination database. See the `convert_database` shell command for a list of argument options and examples.

## convert\_database

The convert\_database shell command converts an existing database to the schema defined for the CalCM application.

### Usage

```
convert_database -source source_DB [-dest destination_DB] [-app_dir app_dir] [-app_type {job | system | rmonitor | tat | rmanager | notification}] [-schema DB_schema_file] [-pcomp]
```

### Arguments

- **-source *source\_DB***

Required argument that specifies the source database where the data is migrated from for the conversion.

- **-dest *destination\_DB***

Optional argument that specifies the destination database for the conversion. If both source and destination directories are used, the data is migrated from the source database into the destination database. This scenario is useful if CalCM is being migrated from using SQLite to MySQL.

- **-app\_dir *app\_dir***

Optional argument that specifies a directory where the required Tcl files for the CalCM application are sourced. If -app\_dir is not specified, the default path is set by the environment variable in the convert\_database shell script. See “[Implementing convert\\_database](#)” on page 387.

- **-app\_type {job | system | rmonitor | tat | rmanager | notification}**

Optional argument that specifies to convert the related database files for the given CalCM application. For example, to convert the job related database files, the -app\_type “job” option is specified. If -app\_type is not specified, by default, all application options are processed during the conversion (starting Calibre version 2019.1). To convert a database to a previous version, the application that matches that version must be used.

---

#### Note

 In the case of SQLite, it is recommended to specify -app\_type, because a different database file is generated for each application.

---

- **-schema *DB\_schema\_file***

Optional argument that specifies the schema file to be used for the conversion. If -schema is not specified, by default, the database is converted based on the schema defined within the CalCM application.

- **-pcomp**

Optional argument that specifies to force data compression during the migration process, thus saving time when the CalCM daemon starts.

## Description

The convert\_database shell command is used with arguments to convert an existing database for stand-alone database migration using the schema defined by the CalCM application or a specified schema file. The functionality of convert\_database comes from a Tcl script that is included with the CalCM Tcl applications and encrypted in the MGC\_HOME package of the Siemens EDA software tree (from Calibre version 2019.1 forward). For more information see “[CalCM Applications](#)” on page 30 and “[CalCM Tcl Application Reference](#)” on page 161.

## Examples

### Example 1

In this example, convert\_database is used with the -app\_type optional argument to convert the job related database files in *calcm\_job.db*.

```
./convert_database -source calcm_job.db -app_type job
```

This is the example standard output:

```
// Running $MGC_HOME/pkgs/icv_calcm/tcl/calcm_convert_database.tcl
Processing $MGC_HOME/pkgs/icv_calcm/tcl/calcm_jobqueue_app.tcl
WARNING: performing migration for database "calcm_job.db"...
WARNING: migrating table "active_jobs" in database "calcm_job.db"
Migrating table "active_jobs" took 1238 seconds.
WARNING: migrating table "finished_jobs" in database "calcm_job.db"
Migrating table "finished_jobs" took 1185 seconds.
WARNING: migrating table "queued_jobs" in database "calcm_job.db"
Migrating table "queued_jobs" took 380 seconds.
Finished!
```

### Example 2

In this example, convert\_database is used to migrate the data from the SQLite database *calcm\_rmonitor.db* to the MySQL database *mysqlcalcmdb* on the machine “mysqldbmachine” with username “testuser” and password “testpwd”.

```
./convert_database -source calcm_rmonitor.db -app_type rmonitor -dest
"testuser:testpwd@mysqldbmachine/mysqlcalcmdb"
```

### Example 3

In this example, the conversion is done according to the specified schema file. The compression tables are created during the migration process from the source database to the destination database.

```
./convert_database -source analysis.db -app_type system -schema
analysis.schema -dest "testuser:testpwd@mysqldbmachine/mysqlcalcmdb"
-pcomp
```

### Example 4

In this example, the conversion uses the required Tcl application files that reside in the */home/user1/app\_files* directory.

```
./convert_database -source calcm_job.db -app_dir /home/user1/app_files  
-app_type job
```

## Schema File Example

The convert\_database shell command converts the source database using the schema defined in the CalCM applications for the target database schema. If the -schema argument option is used, you can define the target database schema according to the specified schema file.

### Target Schema File

Here is an example of a target schema file (Tcl).

```
# Define the database tables and columns:  
set gridTableColumns [list \  
    "entryTime INTEGER" \  
    "totalCpus INTEGER" "totalProcesses INTEGER"]  
  
set nodesTableColumns [list \  
    "entryTime INTEGER" \  
    "hostname TEXT" \  
    "load1 FLOAT" "load5 FLOAT" "load15 FLOAT"]  
  
set jobsTableColumns [list \  
    "entryTime INTEGER" \  
    "job TEXT" \  
    "totalRemoteProcesses INTEGER" "totalActiveProcesses INTEGER" \  
    "cpuDemand INTEGER" \  
    "opClass TEXT" "opNumber INTEGER"]  
  
# Define the schema:  
set databaseTables [list \  
    "grid $gridTableColumns" \  
    "nodes $nodesTableColumns" \  
    "jobs $jobsTableColumns" ]  
  
# Or define the schema using arrays:  
set columns [list]  
array set jobAttrTypes {  
    port TEXT  
    mode TEXT  
    priority INTEGER  
    user TEXT  
    quota TEXT  
    cluster TEXT  
    directory TEXT  
    rule TEXT  
    start INTEGER  
    end INTEGER  
}  
set jobAttr [array names jobAttrTypes]  
foreach attr $jobAttr {  
    lappend columns "$attr $jobAttrTypes($attr)"  
}
```

# Appendix I

## CalScope Usage and Reference

---

The Calibre® System Monitoring Solution (CalScope) software tool is included with the CalCM+ advanced features.

<b>Introduction to Calibre System Monitoring Solution (CalScope) . . . . .</b>	<b>392</b>
CalScope Overview . . . . .	392
CalScope Flow . . . . .	392
CalScope Key Concepts . . . . .	393
CalScope Requirements . . . . .	394
CalScope Modes of Operation . . . . .	395
<b>Running CalScope . . . . .</b>	<b>396</b>
Setting Up Database Access . . . . .	396
CalScope Environment . . . . .	400
Invoking CalScope . . . . .	404
Interacting With CalScope Monitored Hosts . . . . .	404
Importing Job Information With CalScope . . . . .	406
Viewing Job Information in the CalScope Dashboard . . . . .	407
Comparing Job Information in the CalScope Dashboard . . . . .	413
<b>CalScope Command Reference . . . . .</b>	<b>420</b>
calscope . . . . .	421

# Introduction to Calibre System Monitoring Solution (CalScope)

This section introduces the Calibre® System Monitoring Solution (CalScope) software tool. It provides an overview of the product, followed by the flow, key concepts, requirements, syntax conventions, and mode of operation.

CalScope Overview .....	392
CalScope Flow .....	392
CalScope Key Concepts.....	393
CalScope Requirements .....	394
CalScope Modes of Operation .....	395

## CalScope Overview

The CalScope software tool provides a system-level solution for monitoring job and hardware information for data collection and Calibre® run analysis.

Using the functionality of CalScope, administrators and end-users of Calibre jobs in high-performance computing environments gain insight to critical system information through the job and hardware monitoring capabilities of this tool.

- **Job Importing** — Provides job and key CPU performance information from the transcript of a completed Calibre job and saves it to a database. The collected information is used for run and post-run analysis, and job and hardware correlation. Analysis can identify job performance impact due to hardware-related issues.
- **Hardware Monitoring** — Provides hardware information for identifying potential issues and saves it to a database. This functionality enables interaction with monitored hosts by providing status information, enabling add and remove of monitored hosts, and other actionable items.
- **Alert System** — Provides a built-in alert system configured to monitor alerts for job and hardware events. This functionality notifies CalScope end-users of problems occurring in the monitored cluster and enables mitigating action based on predefined criteria.

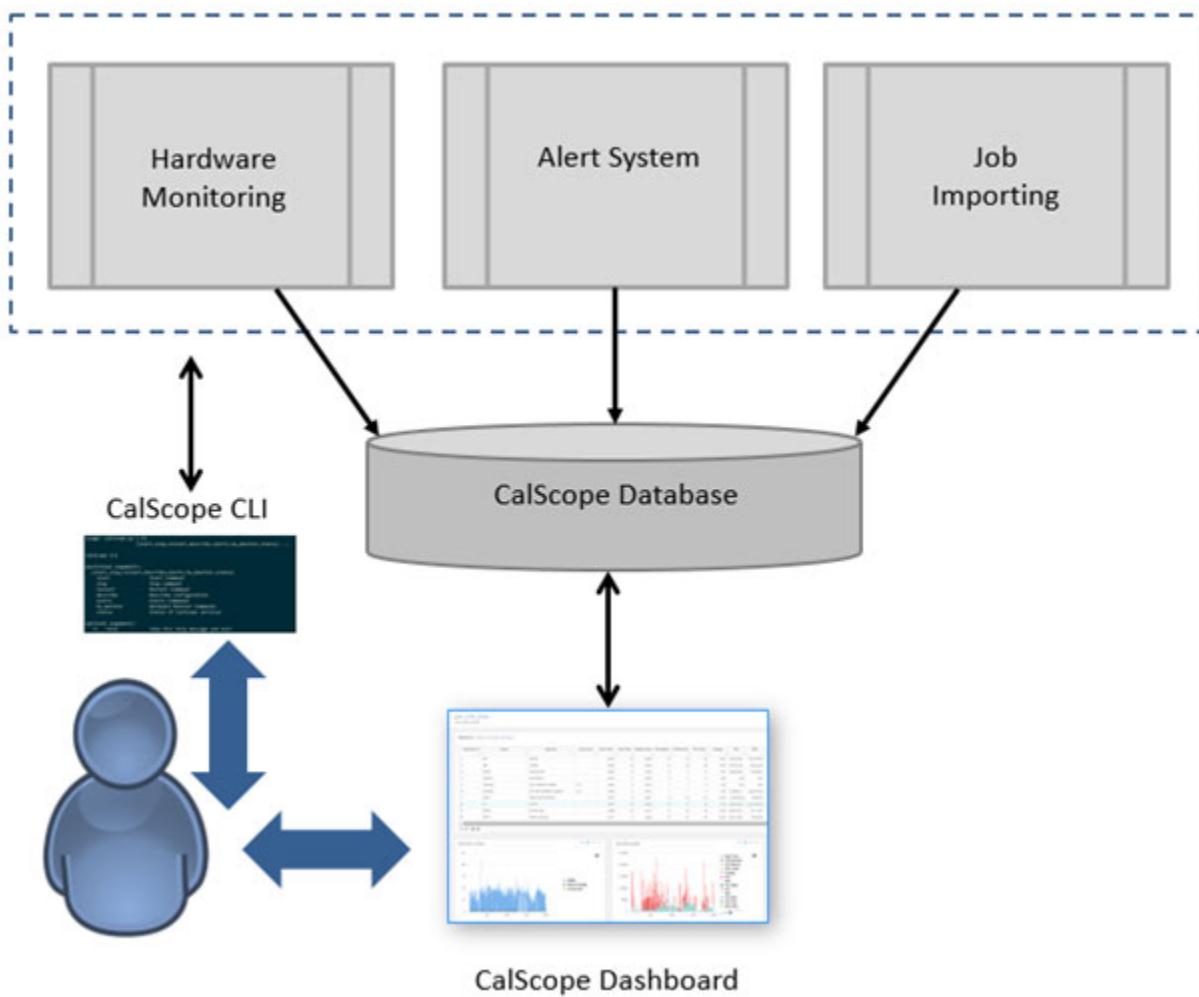
Accessing CalScope functionality is facilitated by a command-line interface in the Linux environment. Visualization of job and hardware monitoring information is provided through the web dashboard for CalScope.

## CalScope Flow

The main components of the CalScope flow work together to collect data and then outputs job and hardware-monitoring information.

The CalScope flow takes in hardware monitoring data, system alerts, job information, and end-user interactions as input and saves it to a database. The information is output to the terminal window and the CalScope dashboard. The command line interface (CLI) enables you to control the CalScope applications and services, while the graphical format of the dashboard enables you to efficiently analyze and compare Calibre runs.

**Figure I-1. CalScope Flow**



## CalScope Key Concepts

Definitions are provided for key terms and concepts used for CalScope functionality.

- **Primary Host** — The primary host is the machine from which you invoke a Calibre job. The primary host launches and connects to remote hosts and handles tasks not assigned to the remote hosts.
- **Remote Host** — The remote hosts run a Calibre task assigned by the Calibre job running on the primary host. The remote hosts must be able to access the MGC\_HOME tree for the respective platform. For Linux<sup>®</sup><sup>1</sup> environments using multiple Linux

hardware and operating systems, the remote hosts may need to access different MGC\_HOME trees for their respective platforms. The remote hosts must communicate with each other and have access to the same file system as the primary host.

- **Hardware Cluster** — A group of linked computers connected to each other through a local area network as nodes. Typically, in a single Calibre run, there is a single primary (host) machine and multiple remote (host) machines. The cluster runs a set of job operations.
- **Hardware Metrics** — A set of Calibre® MTflex™ parameters that are included in Calibre builds (2018.1 and later) for monitoring host-specific and system information. The parameters are grouped in recipes by levels. For example, for recipes *Level1* through *Level4All*, the system monitoring parameters range from basic host parameters to advanced system parameters. Calibre runs with host-monitoring enabled output certain hardware and system metrics that are collected by CalScope.

## CalScope Requirements

The environment for running CalScope has certain hardware and software configuration requirements.

### Hardware and Networking

All remote hosts must be able to communicate with all primary hosts in your hardware system and operating environment. Additionally, all files and directories must be located on a network storage accessible from CalScope’s launch host and also accessible from all hosts specified in the “host:” entries in CalScope’s config file. All other requirements for running Calibre must also be supported.

---

#### Note

 A password-less SSH connection between the CalScope server and the monitoring hosts is required. Refer to “[Setting Up SSH Connectivity Without Password Authentication](#)” in the *Calibre Cluster Manager (CalCM) User’s Manual* for more information.

---

### Licensing

Specific license requirements apply when running CalScope in addition to the necessary product licenses for running Calibre jobs. The standard Calibre license consumption formula applies to all product licenses. For complete information on all Calibre product licensing, refer to the [Calibre Administrator’s Guide](#).

CalScope features are fully enabled with Calibre Cluster Manager Plus (CalCM+) advanced features licenses.

---

1. Linux® is a registered trademark of Linus Torvalds in the U.S. and other countries.

## Calibre Version Support

CalScope is supported as of Calibre version 2021.1.

## System and Software Support

The following system and software versions are required or recommended for running CalScope:

- (Required) A dedicated CalScope server with RHEL or CentOS 7.3, or later.
- (Required) An installation of TimescaleDB software (Timescale Inc.) that supports PostgreSQL functionality. The supported version of this open-source software is available from your Calibre software tree.
- (Recommended) A Firefox 77 or later browser for using the dashboard web application. (Minimum version is Firefox 45 for the expected functionality.)

## Environment Variables

If you do not use any Siemens EDA products other than Calibre, you can set the MGC\_HOME environment variable to the location of your Calibre software tree. If you do use other Siemens EDA products in addition to Calibre, then set the CALIBRE\_HOME environment variable to the location of your Calibre software tree and use the MGC\_HOME environment variable for other Siemens EDA products. See “[“CALIBRE\\_HOME Environment Variable”](#) in the *Calibre Administrator’s Guide* for more information.

# CalScope Modes of Operation

The primary mode of operation for CalScope is through the command-line interface (CLI) and the CalScope dashboard web application.

The CalScope CLI is the starting point for invoking the tool, interacting with the monitored hosts, getting cluster status, and importing job information.

CalScope runs in a host monitoring session using the “calscope” command along with required and optional arguments in a Linux terminal. For example:

```
$MGC_HOME/bin/calscope start -config conf/calscope.yaml -hostfile conf/hostlist
```

During startup, the monitored hosts are validated, and the Linux terminal outputs the URL to the CalScope dashboard web application. You can use the dashboard for efficient visual analysis of the monitored hosts and imported job data.

# Running CalScope

This section covers how to configure the environment, invoke, and use the functionality of CalScope.

<b>Setting Up Database Access .....</b>	<b>396</b>
<b>CalScope Environment .....</b>	<b>400</b>
Configuring the CalScope Environment.....	400
CalScope Example Configuration File.....	402
<b>Invoking CalScope.....</b>	<b>404</b>
<b>Interacting With CalScope Monitored Hosts .....</b>	<b>404</b>
<b>Importing Job Information With CalScope .....</b>	<b>406</b>
<b>Viewing Job Information in the CalScope Dashboard.....</b>	<b>407</b>
<b>Comparing Job Information in the CalScope Dashboard.....</b>	<b>413</b>

## Setting Up Database Access

To set up database access for running CalScope, you must configure your environment as described.

---

### Note

 This procedure uses a provided setup script to facilitate the configuration of your database environment for running CalScope. If your environment already has a compatible version of PostgreSQL with TimescaleDB installed, you perform only a subset of the steps.

---

### Prerequisites

- The requirements are met as specified in “[CalScope Requirements](#)” on page 394.
- The PostgreSQL with TimescaleDB software installation package is available in your Calibre tree.

```
$MGC_HOME/pkgs/icv_timescale
```

Refer to the official websites for Timescale Inc. and PostgreSQL for command usage specific to these open-source database management systems.

- You have access to a dedicated database server for TimescaleDB with adequate disk space and write access.
- You have access to the *calscopedb\_setup.sh* script for setting up the CalScope database.

```
$MGC_HOME/pkgs/icv_calcm/calscope/calscopedb/calscopedb_setup.sh
```

## Procedure

1. Use password-less SSH in a Linux terminal to connect to your dedicated TimescaleDB server.

```
ssh <timescaledb_server>
```

---

### Note

---

 If your environment already has a compatible version of PostgreSQL with TimescaleDB installed, you can skip to either step 6 or 7 to create your CalScope database and schema or only the schema, respectively.

---

2. Configure your TimescaleDB setup using the following environment variables:
  - a. (Required) Specify PG\_HOME to the location for your TimescaleDB data. For example:

```
setenv PG_HOME <path_to_pg_home>
```
  - b. (Optional) Specify PGUSER; otherwise a default user name is assigned. For example:

```
setenv PGUSER <pguser_name>
```
  - c. (Optional) Specify PGPASSWORD; otherwise a default password is assigned. For example:

```
setenv PGPASSWORD <pg_password>
```
  - d. (Optional) Specify PGDATABASE; otherwise a default test database name is assigned. For example:

```
setenv PGDATABASE <pg_database>
```

3. Initialize the database with the following command-line specification (specify in a single line):

```
$MGC_HOME/pkgs/icv_calcm/calscope/calscopedb/calscopedb_setup.sh  
init
```

The following information displays:

```
INFO: Congratulations, the PostgreSQL along with TimescaleDB was  
successfully setup!  
INFO: A user cscope with password cscope was created.  
INFO: A database cscope_test owned by cscope was created.  
INFO: You can start with timescale.sh start  
INFO: You can visit it with:  
INFO: $MGC_HOME/pkgs/icv_timescale/bin/psql  
-h <timescaledb_hostname> -U cscope -d cscope_test
```

4. Create your CalScope database as follows:
  - a. If not already done, specify the PGDATABASE environment variable to change the test database name to your unique CalScope database name.
  - b. Run the following command-line specification (specify in a single line):

```
$MGC_HOME/pkgs/icv_calcm/calscope/calscopedb/calscopedb_setup.sh  
setup_calscope
```

The CalScope database with tables and schema are created.

5. (Optional) Get the usage information to connect, stop, restart, uninstall, and check status of TimescaleDB using the following command-line specification (specify in a single line):

```
$MGC_HOME/pkgs/icv_calcm/calscope/calscopedb/calscopedb_setup.sh  
help
```

This displays your environment variable settings along with the following usage information:

```
//  
...  
VERBOSE: OS : CentOS Linux release <version_7.3_or_later> (Core)  
VERBOSE: MGC_HOME : <path_to_MGC_HOME>  
VERBOSE: PG_HOME : <path_to_PG_HOME>  
VERBOSE: PKG_HOME : <MGC_HOME>/pkgs/icv_timescale  
  
Usage: timescale.sh [init|setup_calscope|  
                     start|stop|status|restart|connect|create_schema|uninstall|help]  
  
Following environment variables are used in this script:  
  MGC_HOME: required  
  PG_HOME: required, installation path which has write permission  
            and enough space.  
  
  PGUSER: optional, timescaleDB user, cscope by default.  
  PGPASSWORD: optional, timescaleDB password, cscope by default.  
  PGDATABASE: optional, timescaleDB testing DB, cscope_test by  
              default.  
  
Assumptions:  
  OS Version is RHEL7+  
  TimescaleDB package is ready in MGC_HOME/pkgs/icv_timescale.
```

6. (Optional) Perform this step if you already have PostgreSQL with TimescaleDB software installed and you need to create your CalScope database and schema:
  - a. Specify PGUSER with your TimescaleDB user name.
  - b. Specify PGPASSWORD with your TimescaleDB password.
  - c. Specify PGDATABASE with the name of your TimescaleDB database.

- d. Run the following command-line specification (specify in a single line):

```
$MGC_HOME/pkgs/icv_calcm/calscope/calscopedb/calscopedb_setup.sh  
setup_calscope
```

7. (Optional) Perform this step if you already have PostgreSQL with TimescaleDB software installed and you only need to create the schema for your existing CalScope database:

- a. Specify PGUSER with your TimescaleDB user name.
- b. Specify PGPASSWORD with your TimescaleDB password.
- c. Specify PGDATABASE with your TimescaleDB database name.
- d. Run the following command-line specification (specify in a single line):

```
$MGC_HOME/pkgs/icv_calcm/calscope/calscopedb/calscopedb_setup.sh  
create_schema
```

## CalScope Environment

The environment for running CalScope requires certain setup and configuration file specifications.

<b>Configuring the CalScope Environment .....</b>	<b>400</b>
<b>CalScope Example Configuration File.....</b>	<b>402</b>

## Configuring the CalScope Environment

You configure the environment for running CalScope using specific database and file configurations.

### Prerequisites

- The requirements are met as specified in “[CalScope Requirements](#)” on page 394.
- The database setup for TimescaleDB is completed as described in “[Setting Up Database Access](#)” on page 396.

### Procedure

1. Set your working directory to the location in which you are going to create the recommended directories and files required to run CalScope. The expected directory structure is shown.

```
calscope
++- conf
|   +- calscope.yaml
|   +- hostlist
++- session
```

2. Create directories for the configuration file and session work.

```
mkdir conf session
```

3. Create a configuration file using YAML format and copy it to your configuration directory (*conf/calscope.yaml*).

The following example shows a moderate configuration that includes database specifications, along with applications for cluster hosts monitoring and alerts. See “[CalScope Example Configuration File](#)” on page 402 for simple and full configuration examples.

```

calscope_home: /home/calscope/session

# Definitions for timescaledb db_string.
# <username> = <timescaledb_username>
# <password> = <timescaledb_password>
# <hostname> = <timescaledb_server_hostname>
# <cluster_db> = <timescaledb_database_name1>
# <alerts_db> = <timescaledb_database_name2>

databases:
  database_A:
    db_string: <username>:<password>@<hostname>/<cluster_db>
    type: timescaledb # Case sensitive.
  database_B:
    db_string: <username>:<password>@<hostname>/<alerts_db>
    type: timescaledb

applications:
  hw_monitor:
    db: database_A
    # Monitoring interval: default 10 seconds.
    monitor_interval: 20
    # Monitoring "primary host" defined.
    host: host01

  alerts:
    db: database_B
    host: host02

  dashboard:
    host: host03
    # Default: a random number.
    port: 5500

```

4. Create a text file, *conf/hostlist*, with a list of host names to monitor. This list can contain host names, IP addresses, or a mix of both.

```

# Add hosts to monitor list.
host01
host02
ip_address1
ip_address2
...

```

The host defined in the hw\_monitor section of the configuration file is the primary host for hardware monitoring and is monitored by default.

5. (Optional) Change the recipe level of the hardware monitoring metrics by updating the parameter in the hw\_monitor section of the configuration file.

```

hw_monitor:
  ...
  # Parameter to specify monitoring recipe level.
  recipe: Level4All
  ...

```

**Note**

 The default CalScope monitoring recipe is *ClusterAll*, which includes *Level4All* metrics plus additional static host metrics.

---

6. (Optional) Specify a custom recipe for the hardware monitoring metrics by updating the parameters in the hw\_monitor section of the configuration file.

```
hw_monitor:  
  ...  
  # Parameters to specify a custom recipe for hardware monitoring.  
  recipe_file: <absolute path to custom Tcl recipe file>  
  recipe_name: <name of recipe>  
  recipe_info: <absolute path to YAML file with custom metrics  
    info>  
  ...
```

## CalScope Example Configuration File

The CalScope configuration file ranges in complexity from a simple configuration to a full configuration that includes multiple database applications. You specify this file using YAML format and copy it to your configuration file directory (*conf/calscope.yaml*).

See “[Configuring the CalScope Environment](#)” on page 400 for more information and an example of a moderate configuration.

### Simple Configuration

```
calscope_home: /home/calscope/session  
  
# Definitions for db_string.  
# <username> = <timescaledb_username>  
# <password> = <timescaledb_password>  
# <hostname> = <timescaledb_server_hostname>  
# <cluster_db> = <timescaledb_database_name>  
  
databases:  
  database_A:  
    db_string: <username>:<password>@<hostname>/<cluster_db>  
    type: timescaledb # Case sensitive.
```

## Full Configuration

```
calscope_home: /home/calscope/session

# Definitions for db_string.
# <username> = <timescaledb_username>
# <password> = <timescaledb_password>
# <hostname> = <timescaledb_server_hostname>
# <cluster_db> = <timescaledb_database_name1>
# <alerts_db> = <timescaledb_database_name2>
# <logview_db> = <timescaledb_database_name3>

databases:
  database_A:
    db_string: <username>:<password>@<hostname>/<cluster_db>
    type: timescaledb # Case sensitive.

  database_B:
    db_string: <username>:<password>@<hostname>/<alerts_db>
    type: timescaledb

  database_C:
    db_string: <username>:<password>@<hostname>/<logview_db>
    type: timescaledb

applications:
  hw_monitor:
    db: database_A
    recipe: ClusterAll
    monitor_interval: 10 # Seconds
    db_interval: 60
    host: host01
    port: 5998
    max_hosts: 20000

  alerts:
    db: database_B
    host: host02
    monitor_interval: 240
    monitor_username: my_username
    monitor_group: tag_name
    monitor_severity: 2

  dashboard:
    host: host03
    port: 5600 # Default is random port.
    autolaunch: {yes | no} # Default is yes.

  job_import: # Optional
    db: database_C

# Watchdog Action. Absolute path to a user-defined shell script.
# This script is triggered in the case of a service failure.
failure_action: /home/calscope/action.sh
```

## Invoking CalScope

You invoke and interact with CalScope from the CLI.

### Prerequisites

- The requirements are met as specified in “[CalScope Requirements](#)” on page 394.
- The database setup is completed as described in “[Setting Up Database Access](#)” on page 396.

### Procedure

1. In a Linux terminal, use password-less SSH to access your dedicated CalScope server.

```
ssh my_calscope_server
```

2. Invoke CalScope by specifying the “calscope start” command along with the path to your configuration and host list files.

```
calscope start -config conf/calscope.yaml -hostfile conf/hostlist
```

3. Check the status of the host cluster by issuing the “calscope status” command.

```
calscope status
```

4. Copy and paste the URL path in the Linux terminal in your web browser to connect to the “Cluster” page of the CalScope dashboard web application and view the host-monitoring information.

```
Validating hosts...
Validating ports...
Validating database connections...
Starting CalScope session...
Copy/paste this URL into your browser:
  http://<my_calscope_server>:<port_number>/caldash/cluster
Launching CalScope Services. It may take a minute to start...
Starting Cluster...
```

---

#### Note



To view job information, see “[Importing Job Information With CalScope](#)” on page 406.

---

5. When you have finished running the host-monitoring session, close the browser window and exit CalScope by specifying the “calscope stop” command in the Linux terminal.

```
calscope stop
```

## Interacting With CalScope Monitored Hosts

You interact with the monitored hosts through the CalScope CLI.

## Prerequisites

- The requirements are met as specified in “[CalScope Requirements](#)” on page 394.
- The database setup is completed as described in “[Setting Up Database Access](#)” on page 396.
- The CalScope environment is set up according to “[Configuring the CalScope Environment](#)” on page 400.

## Procedure

1. Access your CalScope server and invoke CalScope as described in “[Invoking CalScope](#)” on page 404.
2. Use the “calscope hw\_monitor” command to interact with the hosts during the monitoring session.

Remove two hosts from the session by specifying the “hw\_monitor remove” command followed by the space-separated host names.

```
calscope hw_monitor remove -hostnames host03 host04
```

Add the two hosts back into the session by specifying the “hw\_monitor add” command followed by the space-separated host names.

```
calscope hw_monitor add -hostnames host03 host04
```

---

### Note

 The same host cannot be added to a session if it already exists in the cluster of monitored hosts. This means that the “host:” defined in the hw\_monitor section of the configuration file, which is the primary monitoring host, is monitored by default and therefore cannot be re-added to or removed from the monitored cluster.

---

3. Verify the current list of monitored hosts.

```
calscope hw_monitor list
```

The Linux terminal reports the list of host names with the associated IP addresses and core counts for each monitored host. Information on the total number of hosts being monitored and the maximum size of the hardware cluster follows.

HOSTS	IP ADDRESS	CORE COUNT
HOST host01	IP_address_host01	32
HOST host02	IP_address_host02	32
...		
TOTAL HOSTS =	1500	
MAXIMUM CLUSTER SIZE =	20000	

4. Use the help command to show additional command arguments and options.

```
calscope hw_monitor -h
```

See “[calscope](#)” on page 421 for a complete list of command arguments and their descriptions.

- When you have finished running the host-monitoring session, exit the CalScope tool by specifying the “calscope stop” command in the Linux terminal.

```
calscope stop
```

## Importing Job Information With CalScope

You can import a job transcript from a completed Calibre hardware-monitored run and use CalScope to view the job and host information for detailed run analysis.

### Prerequisites

- The requirements are met as specified in “[CalScope Requirements](#)” on page 394.
- The database setup is completed as described in “[Setting Up Database Access](#)” on page 396.
- The CalScope environment is set up according to “[Configuring the CalScope Environment](#)” on page 400. The “job\_import” application and the “db” specification must be defined in CalScope configuration file.
- A transcript file for a finished Calibre run from the hardware-monitored cluster.

### Procedure

- Access your CalScope server and invoke CalScope as described in “[Invoking CalScope](#)” on page 404.
- In a Linux terminal, specify the “calscope job import” command with the full path to the Calibre transcript for analysis.

```
calscope job import -log <full_path>/my_run.log
```

- Copy the URL path from the terminal output.

```
http://<my_calscope_server>:<port_number>/caldash/lvjobs
```

This URL opens the CalScope dashboard web application Jobs page for viewing and analyzing job and hardware-monitoring information from the imported Calibre transcript.

---

#### Note

 You can also use the CLI to import transcripts for finished jobs from the hardware-monitor cluster in the CalScope dashboard session using the “[calscope job import](#)” option.

---

# Viewing Job Information in the CalScope Dashboard

You can use the dashboard web application for CalScope to visualize graphical data for job operations and hardware and system metrics.

## Prerequisites

- The requirements are met as specified in “[CalScope Requirements](#)” on page 394.
- The database setup is completed as described in “[Setting Up Database Access](#)” on page 396.
- The CalScope environment is set up according to “[Configuring the CalScope Environment](#)” on page 400.
- A transcript file is imported for a finished Calibre run from the hardware-monitor cluster and the CalScope dashboard web application is open per “[Importing Job Information With CalScope](#)” on page 406.

## Procedure

1. From the Jobs page of the CalScope dashboard, review and compare the Calibre versions, operation modes, run data, and hardware statistics.

**Figure I-2. CalScope LogView/Monitor Host Job List**

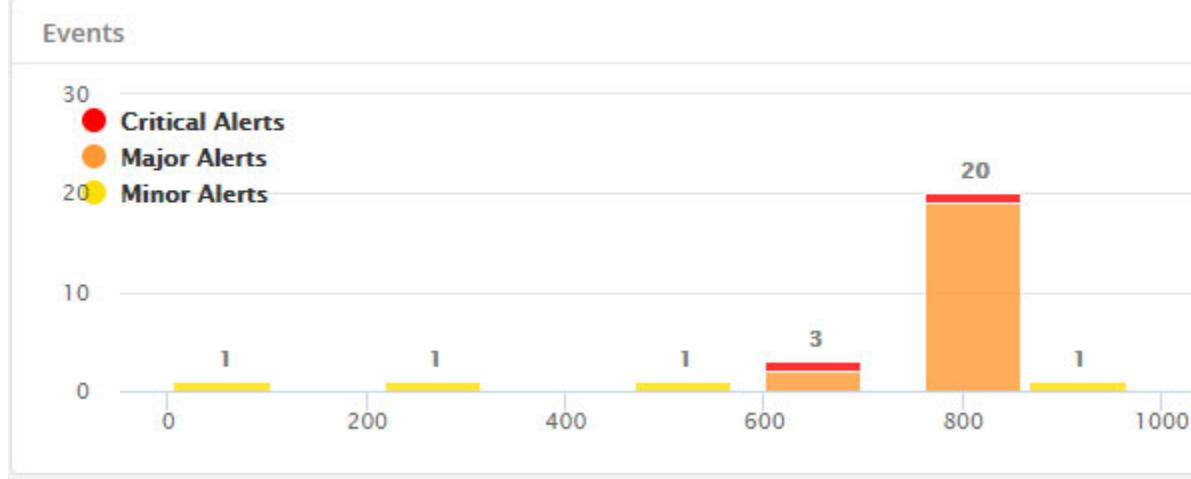
Job	Monitoring Recipe	Log File Name	Log File Path	Version	Command Line
1	UNKNOWN	opcv.log.1	v2021.3_0...	/ Running Mgc_home/pkgs/	
10	UNKNOWN	opcv.log.10	v2021.3_0...	/ Running Mgc_home/pkgs/	
100	UNKNOWN	opcv.log.100	v2021.3_0...	/ Running Mgc_home/pkgs/	
1000	UNKNOWN	opcv.log.1000	v2021.3_0...	/ Running Mgc_home/pkgs/	
1001	UNKNOWN	opcv.log.1001	v2021.3_0...	/ Running Mgc_home/pkgs/	
1002	UNKNOWN	opcv.log.1002	v2021.3_0...	/ Running Mgc_home/pkgs/	
1003	UNKNOWN	opcv.log.1003	v2021.3_0...	/ Running Mgc_home/pkgs/	
1004	UNKNOWN	opcv.log.1004	v2021.3_0...	/ Running Mgc_home/pkgs/	
1005	UNKNOWN	opcv.log.1005	v2021.3_0...	/ Running Mgc_home/pkgs/	
1006	UNKNOWN	opcv.log.1006	v2021.3_0...	/ Running Mgc_home/pkgs/	
1007	UNKNOWN	opcv.log.1007	v2021.3_0...	/ Running Mgc_home/pkgs/	
1008	UNKNOWN	opcv.log.1008	v2021.3_0...	/ Running Mgc_home/pkgs/	
1009	UNKNOWN	opcv.log.1009		/ Running Mgc_home/pkgs/	

- Click an entry in the Jobs column for detailed information about a specific job. This opens the job detail page and displays a job operations table, events chart, and plots of hardware and system monitoring metrics.

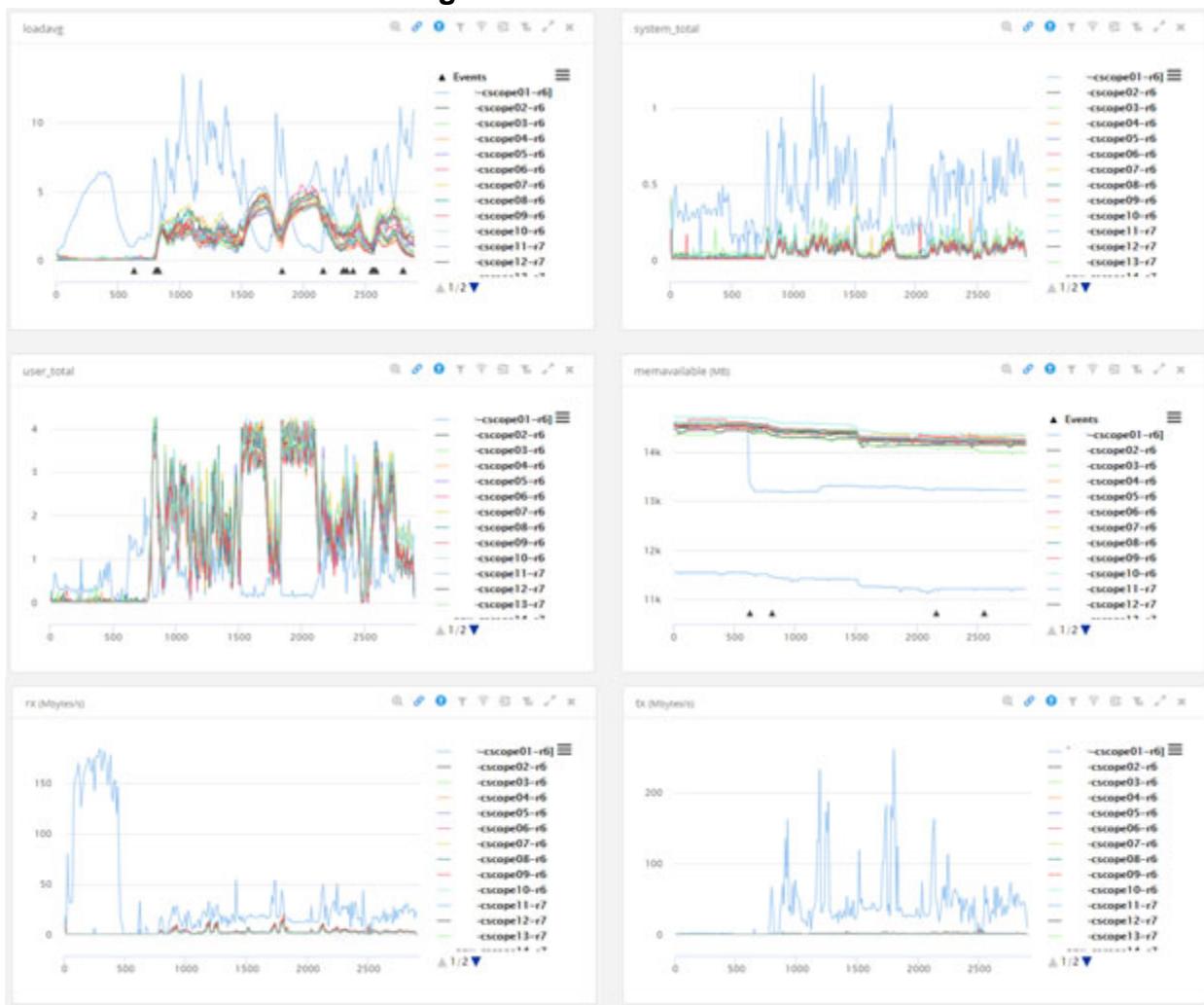
**Figure I-3. Job Operations Table**

Job: run2

Operation #	Name	Operator	Concurrent	Start Time	Real Time	Elapsed Time	CPU Master	CPU Remote	CPU Total	Scaling
1	pc	OR pc		776	2	778	4	22	26	13.00
2	pc_e1	OR pc_e1		778	0	778	0	0	0	0.00
3	pcberg	OR pcberg		778	0	778	0	0	0	0.00
4	TMP<50>	pc_e1 OR pcberg		778	0	778	0	0	0	0.00
5	e1_drawn	pc OR TMP<50>		777	1	778	1	0	1	1.00
6	noniagoc	OR noniagoc		778	0	778	0	0	0	0.00
7	nomboopcpc	OR nomboopcpc		778	0	778	0	0	0	0.00
8	celisnr	OR celisnr		778	0	778	0	0	0	0.00
9	TMP<40>	nomboopcpc OR celisnr		778	0	778	0	0	0	0.00
10	no_retarget_marker	noniagoc OR TMP<40>		778	0	778	0	0	0	0.00

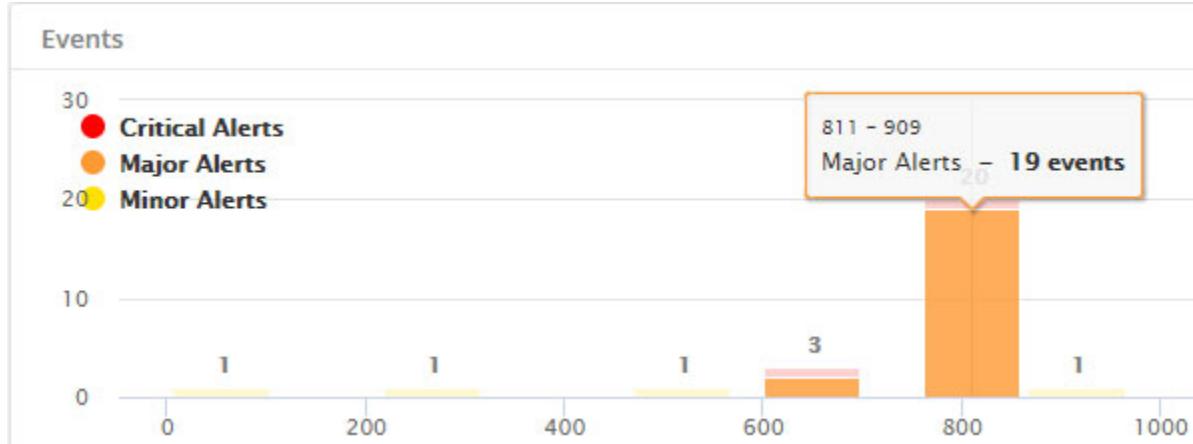
**Figure I-4. Job Events Chart**

**Figure I-5. Job Plot Metrics**



3. In the Events chart, hover your cursor over a bar to show the job alert details.

**Figure I-6. Job Events Detail**



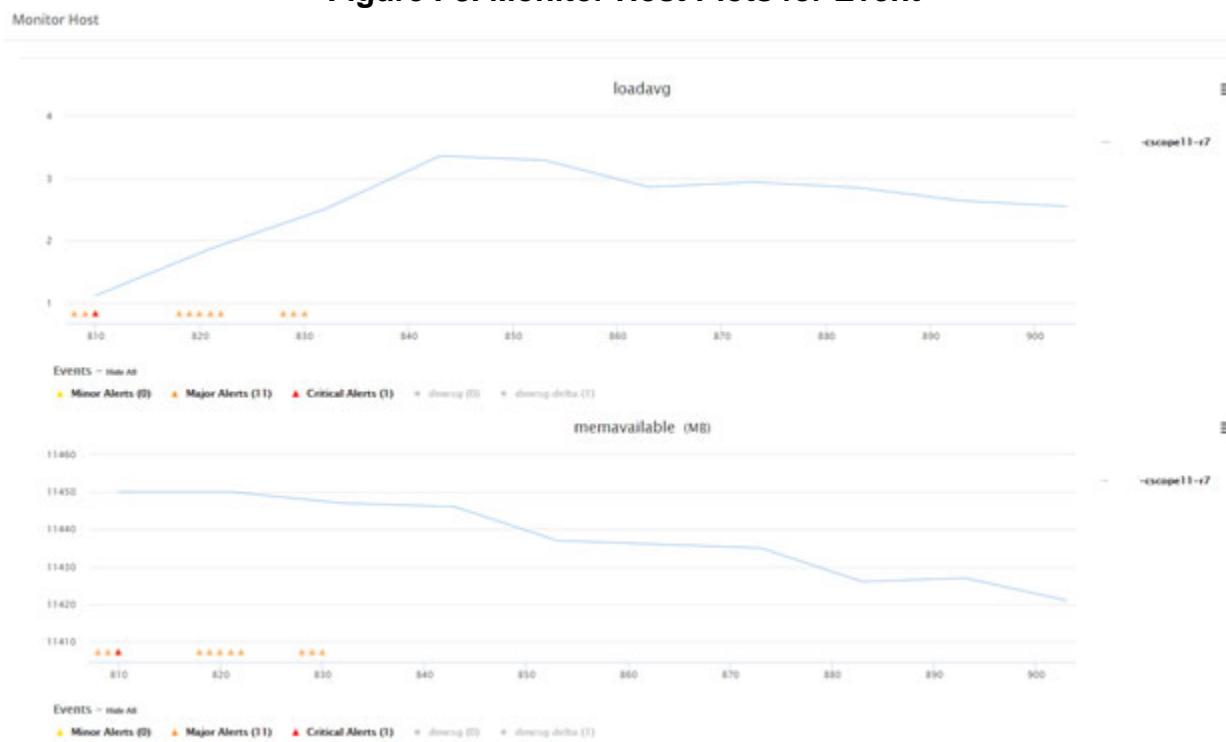
4. Click a Critical Alerts bar (red) to display corresponding host alert information.

**Figure I-7. Host Alert Information**



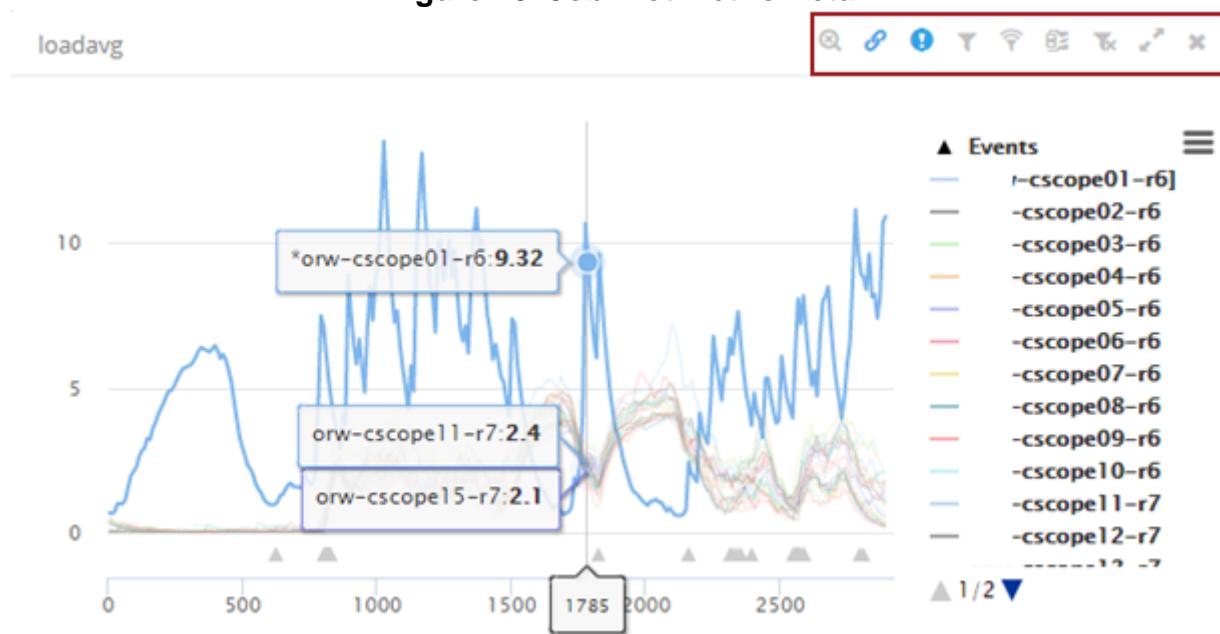
5. Click the host name to display the Monitor Host plots for the event.

**Figure I-8. Monitor Host Plots for Event**



6. Close the Monitor Host plots window.
7. In the job plot metrics, move your mouse over a plot line to display the host and alert information at a specific time. The icons in the upper-right corner of each plot provide additional chart functions. You can view the function definition by hovering over an icon with your cursor.

**Figure I-9. Job Plot Metric Detail**



- To select different metrics for plotting, click the plus sign (+) icon in the upper-right corner of the job detail page to open the Add Plots window. In this window, you can choose the LogView, Monitor Host, and hardware and system metrics to plot.

**Figure I-10. Job Add Plots Window**

Add Plots

Search Metrics

Available Metrics Select Metrics you want to see in the page

<input type="checkbox"/>	LogView	<input type="checkbox"/>	operations	<input type="checkbox"/>	operation_scaling	<input type="checkbox"/>	LVHEAP		Total 3 Metrics									
<input checked="" type="checkbox"/>	Monitor Host	<small>Selected</small>							Total 59 Metrics									
<input type="checkbox"/>	CPU								Total 19 Metrics									
<input type="checkbox"/>	cpu_freq_max	<input type="checkbox"/>	cpu_freq_min	<input type="checkbox"/>	cbt	<input type="checkbox"/>	idle_total	<input type="checkbox"/>	iowait_total	<input type="checkbox"/>	irq_total	<input checked="" type="checkbox"/>	loadavg	<input type="checkbox"/>	softirq_total	<input type="checkbox"/>	system_max	Total 13 Metrics
<input type="checkbox"/>	nice_total	<input type="checkbox"/>	procs_blocked	<input type="checkbox"/>	procs_created	<input type="checkbox"/>	procs_running	<input type="checkbox"/>	procs_total	<input type="checkbox"/>	user_max	<input checked="" type="checkbox"/>	user_min	<input type="checkbox"/>	user_total	<input type="checkbox"/>		
<input type="checkbox"/>	system_min	<input type="checkbox"/>	system_total	<input checked="" type="checkbox"/>	user_max	<input type="checkbox"/>	user_min	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		
<input type="checkbox"/>	Memory								Total 11 Metrics									
<input type="checkbox"/>	committed_as	<input checked="" type="checkbox"/>	memavailable	<input type="checkbox"/>	membuffers	<input type="checkbox"/>	memcached	<input type="checkbox"/>	memshmem	<input type="checkbox"/>	numa_hit	<input type="checkbox"/>	numa_miss	<input type="checkbox"/>				
<input type="checkbox"/>	pswpin	<input type="checkbox"/>	pswpout	<input type="checkbox"/>	swpfree	<input type="checkbox"/>	swpoutwait	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		
<input type="checkbox"/>	Network								Total 11 Metrics									
<input checked="" type="checkbox"/>	rx	<input type="checkbox"/>	rx_drops	<input type="checkbox"/>	rx_errs	<input type="checkbox"/>	tcp_socket_inuse	<input type="checkbox"/>	tcp_socket_mem	<input type="checkbox"/>	tcp_socket_orphan	<input checked="" type="checkbox"/>	tx	<input type="checkbox"/>				
<input type="checkbox"/>	tx_drops	<input type="checkbox"/>	tx_errs	<input type="checkbox"/>	udp_socket_inuse	<input type="checkbox"/>	udp_socket_mem	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		
<input type="checkbox"/>	IO								Total 18 Metrics									
<input type="checkbox"/>	disk_io_curr	<input type="checkbox"/>	disk_read	<input type="checkbox"/>	disk_read_req	<input type="checkbox"/>	disk_read_time	<input type="checkbox"/>	disk_write	<input type="checkbox"/>	disk_write_req	<input type="checkbox"/>	disk_write_time	<input type="checkbox"/>				
<input type="checkbox"/>	files_open	<input type="checkbox"/>	nfs_dir_read	<input type="checkbox"/>	nfs_dir_write	<input type="checkbox"/>	nfs_iops	<input type="checkbox"/>	nfs_map_read	<input type="checkbox"/>	nfs_map_write	<input type="checkbox"/>	nfs_rpc_backlog	<input type="checkbox"/>				
<input type="checkbox"/>	nfs_svr_read	<input type="checkbox"/>	nfs_svr_write	<input type="checkbox"/>	nfs_sys_read	<input type="checkbox"/>	nfs_sys_write	<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		<input type="checkbox"/>		

Cancel Reset **Apply**

9. Click **Apply** to apply the selected metrics and close the Add Plots window. The job detail page updates to display the new selection of plot metrics.
10. You can control how information in the job detail page is displayed both globally and locally within each metric. The row of icons in the upper-right of the page controls global options while each metric has its own row of icons to control the options for that individual metric.

**Figure I-11. Job Detail Global Options****Figure I-12. Job Detail Local Options****Table I-1. Global and Local Icon Descriptions**

<b>Icon</b>	<b>Description</b>
	Global: Resets the zoom of all the charts. Local: Resets the zoom of the individual chart.
	Global: Toggles linking on all charts and tables. Linking is when you click on an item in a chart or table and the corresponding item is highlighted in any chart or table that has linking turned on. Local: Toggles linking in the individual chart or table. Linking only highlights items in charts or tables with linking turned on.
	Global: Toggles threshold boxes on all charts. When threshold boxes are turned on, mousing over charts with defined threshold alerts will shade areas in the chart outside the threshold bounds. Local: Toggles threshold boxes of the individual chart.
	Global: Opens the Global Filters settings page where you can specify the filter conditions used to display all the charts. Local: Opens the Local Filter settings page where you can specify the filter conditions used to display in the individual chart.
	Local only: Shares the current filter settings for the individual chart to the rest of the charts.
	Global: Clears all filters in all charts. Local: Clears the filter for the individual chart.
	Global only: Opens the Add Plots page where you can select which plots to display.
	Global only: Opens a dropdown menu where you can modify such options as x-axis labels, tooltip and legend visibility, click-on action, and the threshold for showing aggregated charts.

**Table I-1. Global and Local Icon Descriptions (cont.)**

Icon	Description
	Local only: Expands the individual chart display with the ability to filter.
	Local only: Closes the individual panel.

## Comparing Job Information in the CalScope Dashboard

You can use the CalScope dashboard to compare job run time and scalability along with underlying hardware load/performance. This is especially useful to analyze slow runs and to compare benchmarks.

### Prerequisites

- The requirements are met as specified in “[CalScope Requirements](#)” on page 394.
- The database setup is completed as described in “[Setting Up Database Access](#)” on page 396.
- The CalScope environment is set up according to “[Configuring the CalScope Environment](#)” on page 400.
- A transcript file is imported for a finished Calibre run from the hardware-monitor cluster and the CalScope dashboard web application is open per “[Importing Job Information With CalScope](#)” on page 406.

### Procedure

1. In the Jobs page, click the dropdown icon at the bottom-center of the table to select the number of job records to display as shown in [Figure I-13](#). The default is 50 job records maximum per page.
2. To compare detailed job information for two jobs, choose two job IDs in the Job column of the table and click **Compare**.

---

**Note**



You can only compare two jobs at a time. Choosing more than two jobs will display a warning popup.

---

**Figure I-13. CalScope Job Selection and Compare**

The screenshot shows the Calibre Cluster Manager interface with the 'CalScope' section selected. The 'Jobs' tab is active. A 'Compare' button is located in the top left of the main content area. Two specific jobs are selected for comparison: job ID 10 and job ID 100. The main table lists numerous log entries with detailed information. At the bottom right, a dropdown menu for selecting the number of items per page is open, showing options 50, 100, and 200.

The Compare Jobs page opens and displays the information for the selected job IDs.

**Figure I-14. CalScope Compare Jobs Page**

The screenshot displays the CalScope Compare Jobs Page with two main sections: 'Jobs' and 'Operations'.

**Jobs:** This section shows a comparison between 'Job 10' and 'Job 100'. The table includes columns for Job, Version, Total Real Time, Total CPU Time, Master CPU Time, Remote CPU Time, Scaling, Monitoring Recipe, Log File Name, and several redacted log file names. Differences are highlighted in green (+) or red (-) for percentages.

Job	Version	Total Real Time	Total CPU Time	Master CPU Time	Remote CPU Time	Scaling	Monitoring Recipe	Log File Name			
10	v2021.3_0.26 (...)	00:05:44	01:42:39	00:01:33	01:41:06	17.90	UNKNOWN	opcv.log.10			
100	v2021.3_0.26 (...)	(+17.2 %)	(+0.3 %)	01:42:58	(-1.1 %)	00:01:32	(+0.3 %)	01:41:26	(-14.4 %) 15....	UNKNOWN	opcv.log.100

**Operations:** This section compares two sets of operations. The left panel shows operations for 'Job 10' and the right panel shows operations for 'Job 100'. Both panels include columns for Operation, Name, and Operator.

Operation	Name	Operator
1	POLY	OR POLY
2	POLY_OP	OR POLY_OP
3	CONTACT_OP	OR CONTACT_OP
4	ACTIVE	OR ACTIVE
5	CONTACT	OR CONTACT
6	M1	OR M1
7	M1_OP	OR M1_OP
8	img_ctr	LITHO OPCVERIFY POLY PC
9	short	LITHO OPCVERIFY POLY PC
10	open	LITHO OPCVERIFY POLY PC

**Events:** This section shows event data for both jobs, specifically for the 'operation\_scaling' event. It includes a chart and a table.

**Events - operation\_scaling:**

Event Type	Value
Load	3k
Queue	2k
OP	1k
HDB0	0

3. The Jobs table displays useful comparisons of the two jobs, including CPU time, scaling, and so on, with differences shown in red or green percentages.

**Figure I-15. Job Comparisons**

Total Real Time	Total CPU Time	Master CPU Time	Remote CPU Time	Scaling
00:03:08	01:41:06	00:01:30	01:39:36	32.27
(+83.0 %)	00:05:44	(+1.5 %)	01:42:39	(+3.3 %)
			00:01:33	(+1.5 %)
			01:41:06	(-44.5 %)
				17.90

4. The Hosts section enables you to control which hosts are shown in the hardware plots. Click on the right of the Hosts section to expand it.

By default, all hosts are shown (Show All). You can also choose to display only Common Hosts (hosts in both jobs), just hosts from Job 1, or just hosts from Job 2. In addition, you can filter using the fields in each of the table headers.

**Figure I-16. Controlling Which Hosts to Display in Plots**

The screenshot shows the 'Hosts' section of the CalScope dashboard. A dropdown menu is open, showing options: 'Show All', 'Job 1 only', 'Job 10 only', and 'Common Hosts'. The 'Common Hosts' option is highlighted with a blue background and a cursor is hovering over it. The table below lists several hosts, all of which are currently selected, as indicated by checked checkboxes in the first column. The columns include Host, CPU name, CPUs, waptotal, os\_distro, os\_release, btime, and status. The status column shows 'Up' for all hosts.

5. Control the chart features using the icons in the upper-right corner of the Compare Jobs page (shown left to right):



- Switch View** — Switches the chart views between horizontally and vertically stacked orientation.
- Common Axis** — Toggles the display of the x- and y-axis scales for runtime and scalability comparison.
- Reset Zoom** — Resets the zoom (out) across all charts.
- Link Charts** — Toggles the linking functionality across all charts.
- Toggle global threshold boxes** — Toggles all the alert threshold box settings across all charts.

- f. **Add Plots** — Opens the Add Plots window where you can select different metrics for plotting.
- g. **Chart (ellipses (...))** — Accesses the following chart options:
  - o **X-Axis labels** — Changes the x-axis between Elapsed Time, Date Time, and Epoch Time.
  - o **Click on Operation** — Changes what happens when you click on the charts.
  - o **Chart Legend (Monitor Host)** — Toggles the visibility of the chart legends.
  - o **Chart Tooltip (Monitor Host)** — Toggles the visibility of tooltips when mousing over charts.
  - o **Show aggregated charts if #host>** — If the number of hosts exceed this specified value, the charts will be aggregated.
6. Click on a row in the Operations table to highlight the corresponding area in the charts.

**Figure I-17. Highlight an Operation**

The figure displays two identical CalScope dashboard interfaces side-by-side, illustrating how to highlight specific operations across different data sources.

**Left Dashboard View:**

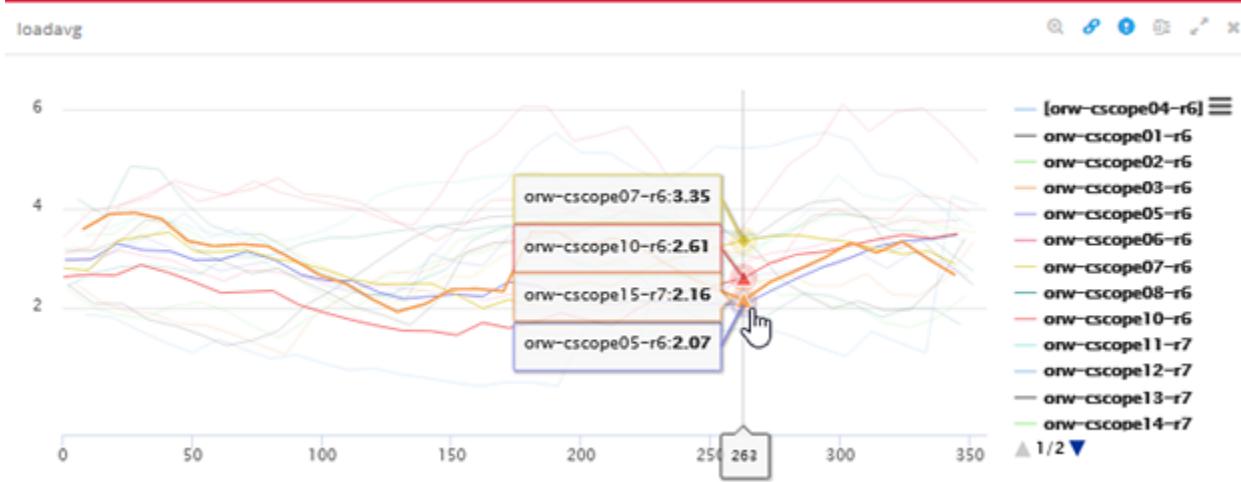
- Operations Table:** Shows a list of 31 operations. The 8th row, labeled "img\_ctr", is highlighted with a light blue background. A red arrow points from the "Events" chart below to this row.
- Events Chart:** Titled "operation\_scaling", it shows a step function plot. The Y-axis ranges from 0 to 3k, and the X-axis ranges from 0 to 400. The plot shows a sharp rise from 0 to approximately 3.5k at X=100, followed by a drop to about 1k at X=300.
- loadavg Chart:** Titled "loadavg", it shows multiple time-series lines for various hosts. The Y-axis ranges from 0 to 6, and the X-axis ranges from 0 to 400. A red arrow points from this chart to the "img\_ctr" row in the Operations table.

**Right Dashboard View:**

- Operations Table:** Shows a list of 31 operations. The 5th row, labeled "ACTIVE", is highlighted with a light blue background.
- Events Chart:** Titled "operation\_scaling", it shows a step function plot similar to the left side, with a sharp rise from 0 to approximately 3.5k at X=100 and a subsequent drop.
- loadavg Chart:** Titled "loadavg", it shows multiple time-series lines for various hosts. The Y-axis ranges from 0 to 100, and the X-axis ranges from 0 to 400. A red arrow points from this chart to the "ACTIVE" row in the Operations table.

7. Mouseover the charts to view the information about each item and click on the items to highlight the corresponding operations in the Operations table.

Figure I-18. Mouseover the Chart



## Results

You have used table and chart functions in the CalScope dashboard to selectively compare two jobs and control how it is displayed.

## CalScope Command Reference

---

Specific command line arguments and options are used when invoking and interacting with the CalScope tool.

**calscope** ..... 421

# calscope

Command line executable for CalScope.

You specify the calscope command to invoke and interact with the CalScope tool.

## Usage

### Invocation Command-Line Arguments

```
calscope start {-config config_file [-hostfile hostlist] | [-h]}
```

### Interactive Command-Line Arguments

```
calscope {  
    stop {[-force] | [-h]} |  
    restart {[-config config_file] [-force] [-no_remotes] | [-h]} |  
    describe {application | [-h]} |  
    alerts {create -condition "condition"  
            [-host_id host_id] [-severity {1,2,3}] [-description description]  
            [-group group] [-action_path action_path] | [-h]} |  
    delete {alert_id | [-h]} |  
    disable {alert_id | [-h]} |  
    export {-report_file report_file [-severity {1,2,3}] [-username username]  
            [-group group] | [-h]} |  
    import {-report_file report_file [-severity {1,2,3}] [-username username]  
            [-group group] | [-h]} |  
    [-h]} |  
    hw_monitor {add {-hostnames hostname [hostnames...]} | -hostfile hostfile_name | [-h]} |  
    remove {-hostnames hostname [hostnames...]} | -hostfile hostfile_name | [-h]} |  
    update {-hostfile hostfile_name | [-h]} |  
    list [-h] |  
    recipe {-recipe_name recipe_name  
            [-recipe_file recipe_file] [-recipe_info recipe_info_file] | [-h]} |  
    [-h]} |  
    job import {-log primary_transcript [primary_transcript ...] [-view] | [-h]} |  
    status [-h] |  
    [-h]  
}
```

## Arguments

- **start**

Required argument that invokes the CalScope tool.

**-config config\_file** — Specifies a required configuration file that contains paths and specifications for the session directory, database structure, environment variables, and information used for the hardware, job monitoring, and dashboard applications. The file format follows the YAML syntax convention. See “[Configuring the](#)

[“CalScope Environment”](#) on page 400 and [“CalScope Example Configuration File”](#) on page 402 for examples.

-hostfile *hostlist* — A text file that specifies the list of host names, IP addresses, or a mix of both to monitor with each host name or IP address on a separate line.

-h — Displays a help message listing the required and optional arguments for the specified command and exits.

- **stop**

Required argument option that ends the CalScope session.

-force — Specifies to force the CalScope move the session and exit, even if the stop does not complete successfully. May be specified as -f.

-h — Displays a help message listing the required and optional arguments for the specified command and exits.

- **restart**

Required argument option that restarts the CalScope tool session with the specified options.

-config *config\_file* — Specifies a configuration file to be used when restarting the session. See the “calscope start” argument for file information.

-force — Specifies to *not* confirm before restarting the cluster remotes.

-no\_remotes — Specifies to *not* restart the cluster remotes.

-h — Displays a help message listing the required and optional arguments for the specified command and exits.

- **describe**

Required argument option that describes the configuration parameters used for the session default.

***application*** — Specifies the application to describe. Requires one of “dashboard”, “alerts”, “hw\_monitor” or “all” to be specified. Specifying “all” describes the configuration parameters for all three applications.

-h — Displays a help message listing the required and optional arguments for the specified command and exits.

- **alerts**

Required argument option that enables the alert monitoring system in CalScope. The system uses predefined alert criteria to notify users of potential problems with the monitored cluster so that mitigating actions can be taken. Requires the “alerts” application section and “db” definition in the CalScope configuration file.

**create** — Specifies to create an alert for the given condition.

**-condition “*condition*”** — Specifies a condition that triggers an alert event and causes it to persist. The behavior is such that only new triggered alerts are reported, and ongoing alert conditions are not repeatably reported. The condition search string is enclosed in quotation marks with ordered case-sensitive parameters.

Reports two types of alert conditions:

- **Metric Alerts** — Based on a search of the hardware metrics information from the hardware monitor service. These search strings include metric names, logical constraint operators, and constraint limit values. For example:

```
calscope alerts create -condition "loadavg > 1 and \
    MemAvailable < 13500"
...
Created Alert ID = 1 at Epoch: 1595893481
```

Percentage constraint values are supported for the following metrics:

- system\_total
- memavailable
- swapfree

For example:

```
alerts create -condition "system_total > 90%"
alerts create -condition "memavailable < 85% and \
    loadavg > 1"
```

- **dmesg Alerts** — Based on a search of the dmesg message Linux information. These search strings include comma-separated arguments for message names, logical constraint operators, occurrence count constraints, constraint limit values, and occurrence duration for which the count is met (in seconds). For example:

```
calscope alerts -condition "VMCI,>,0,3600"
...
Created Alert ID = 2 at Epoch: 1595893521
```

-host\_id *host\_id* — Specifies a host to apply the constraint condition. The default is all monitored hosts.

-description *description* — Specifies a description of the alert to display in the dashboard events and in the monitoring transcript. The default is no description.

-severity {1,2,3} — Specifies the severity level of the alert to display in the dashboard events and in the monitoring transcript. Levels correspond to “Minor”, “Major”, and “Critical” categories, respectively. The default level is 2.

-group *group* — Specifies a user-created tag name for grouping, categorizing, and filtering the alerts.

-action\_path *action\_path* — Specifies the full path to an executable shell script that performs a mitigating action for the triggered alert condition. The script only executes if the user running the monitoring is the creator of the alert. The monitoring system provides the host name (where the event occurred) and the monitor host metrics to the script.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

**delete** — Deletes an alert and all the related events for the specified alert\_id.

**alert\_id alert\_id** — Specifies an alert\_id from a created alert or an exported YAML format report.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

**disable** — Disables an active alert for the specified alert\_id by removing it from checking in further monitoring cycles.

**alert\_id alert\_id** — Specifies an alert\_id from a created alert or an exported YAML format report.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

## **export**

**-report\_file report\_file** — Enables the export of alert information from the system monitoring database to a YAML format file.

**-severity {1,2,3}** — Specifies of the severity level of the alert to export.

**-username username** — Specifies the user name of a user-created alert to export.

**-group group** — Specifies the user-created tag name for the alert group to import.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

## **import**

**-report\_file report\_file** — Enables the import of alert information from the system monitoring database to a YAML format file.

**-severity {1,2,3}** — Specifies the severity level of the alert to import.

**-username username** — Specifies the user name of a user-created alert to import.

**-group group** — Specifies the user-created tag name for the alert group to import.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

- **hw\_monitor**

Required argument option that specifies the hardware monitor application. This application collects hardware and system utility information from the monitored hosts and saves it to the database. Requires the “hw\_monitor” application section and “db” definition in the CalScope configuration file. Must be specified with one of the following arguments:

**add** — Specifies to add a host or multiple hosts to the existing monitored host cluster. A host can only be added once to the same group of monitored hosts. One of -hostnames or -hostfile must be specified.

**-hostnames *hostname [hostnames...]*** — Specifies the hostnames to add to the monitored host cluster. Multiple space-separated hostnames can be specified.

**-hostfile *hostfile\_name*** — Specifies a text file with a list of host names to add to the monitored host cluster. The format is one host name per line.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

**remove** — Specifies to remove a host or multiple hosts from the monitored host cluster. One of -hostnames or -hostfile must be specified.

**-hostnames *hostname [hostnames...]*** — Specifies the host names to remove from the monitored host cluster. Multiple space-separated host names can be specified.

**-hostfile *hostfile\_name*** — Specifies a text file with a list of host names to be removed from the monitored host cluster. The format is one host name per line.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

**update** — Specifies to update the host cluster to match the given host file by adding or removing hosts as needed. The -hostfile argument must be specified.

**-hostfile *hostfile\_name*** — Specifies a text file with a list of host names for the cluster update. The format is one host name per line.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

**list** — Prints all hosts in the monitored cluster to the terminal window and transcript.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

**recipe** — Specifies a custom recipe with parameters for the host monitoring session.

**-recipe\_name *recipe\_name*** — Specifies a custom recipe name. Required for custom recipes. One of -recipe\_file or -recipe\_info must be specified.

**-recipe\_file *recipe\_file*** — Specifies a custom recipe file that is a Tcl file. Required for custom recipes.

**-recipe\_info *recipe\_info\_file*** — Specifies a custom recipe file that is a YAML format file. Required for custom recipes. See the Examples section for an example of the YAML format.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

- **job**

Required argument option that specifies the job application. This application enables the import of a transcript from a finished Calibre job (run on CalScope monitored hosts) into the CalScope database. The imported information displays in the jobs view of the CalScope dashboard session. Requires the “job\_import” application section and “db” definition in the CalScope configuration file.

**import** — Imports Calibre job information from a completed primary host transcript or monitor host (*CalibreMonitorHostLog*\*) transcripts.

**-log *primary\_transcript* [*primary\_transcript...*]** — Specifies the absolute path to the transcript of a completed Calibre run. Multiple transcripts specifications use space-separated format.

**-view** — Specifies to launch a new CalScope dashboard.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

- **status**

Required argument option that reports the host information for the CalScope applications and services at the epoch time of the status request.

**-h** — Displays a help message listing the required and optional arguments for the specified command and exits.

- **-h**

Optional argument that displays a help message listing the required and optional arguments for the specified command and exits.

## Description

The “calscope start” command is used to invoke the CalScope tool from a Linux command line. During the CalScope session, you specify the calscope command with the interactive command-line arguments to access alerts, hardware monitoring, job information, and tool functionality. These commands and arguments enable you to control the CalScope applications and services.

## Examples

### Example 1

In this example, a CalScope session is invoked using the “calscope start” command along with specifications for the required configuration file and an optional host file containing a list of host names to monitor.

```
$MGC_HOME/bin/calscope start -config conf/calscope.yaml -hostfile conf/hostlist
```

### Example 2

This example shows the contents of an hw\_monitor custom recipe information YAML file. It is specified with the “hw\_monitor recipe -recipe\_info *recipe\_info\_file*” arguments.

```
sshd_status:  
    description: "Whether sshd service is up and running. 0: down, 1: up"  
disk_local_free:  
    description: "The total amount of space available in the local file  
                  disk."  
    unit: "MB"  
disk_tmp_free:  
    description: "The total amount of space available in /tmp directory."  
    unit: "MB"  
    scale: 1
```

The following describes the fields:

- **description** — Required for custom metrics at the time of its first addition into the database. If a metric description already exists in the database, a warning is issued that the description will not be updated.
- **unit** — Optional, default is empty string.
- **scale** — Optional, default value is 1.



# Index

---

## — Symbols —

[]<sup>23</sup>  
{}<sup>23</sup>  
|<sup>23</sup>

## — A —

Alert system, [392](#)  
APPLICATION, [213](#), [311](#)  
Applications  
    creating, [318](#)  
    initializing, [312](#)

## — B —

Bold words, [22](#)  
Braces, [23](#)  
Brackets, [23](#)

## — C —

CalCM  
    directory structure, [49](#)  
    events, [311](#)  
    log file, [322](#)  
    troubleshooting, [307](#)  
CalCM applications, [182](#)  
    calcmtt\_http\_server\_app.tcl, [166](#)  
    calcmtt\_jobqueue\_app.tcl, [174](#)  
    calcmtt\_rmanager\_app.tcl, [195](#)  
    calcmtt\_systemanalysis\_app.tcl, [200](#), [203](#)  
CalCM events, [311](#)  
calcmtt\_actlog\_app.tcl, [310](#), [311](#)  
calcmtt\_http\_server\_app.tcl, [166](#)  
calcmtt\_jobqueue\_app.tcl, [174](#)  
calcmtt\_notification\_app.tcl, [182](#)  
calcmtt\_rmanager\_app.tcl, [195](#)  
calcmtt\_send\_message, [127](#)  
calcmtt\_submit\_job, [129](#)  
calcmtt\_systemanalysis\_app.tcl, [200](#), [203](#)  
CalCM+  
    advanced features, [92](#)  
    cluster utilization data, [98](#)

job statistics, [92](#)  
calcmd, usage, [123](#)  
Calibre Cluster Manager  
    Activity log application, [31](#)  
    HTTP server application, [30](#)  
    Job scheduler application, [30](#)  
    Resource manager application, [31](#)  
    Resource monitor application, [31](#)

calscope, [421](#)  
Cluster, [18](#)  
Cluster configuration file statements  
    MASTER HOST, [225](#)  
    RDS HOST, [226](#)  
    REMOTE HOST, [227](#)  
Cluster snapshot, [313](#), [325](#)  
    object types, [326](#)  
Command syntax, [22](#)  
Components, [393](#)  
Configuration file  
    job, [28](#), [51](#)  
    job environment, [28](#), [52](#)  
    network, [27](#), [49](#)

Configuration file statements  
    APPLICATION, [213](#)  
    SERVER INTERVAL, [219](#)  
    SERVER PORT, [220](#)  
    SERVER TIMEOUT, [221](#)  
    VARIABLE, [222](#)

Courier font, [22](#)

## — D —

Double pipes, [23](#)

## — E —

ECP, [18](#)  
Environment variables, Calibre home, [395](#)  
ERT, [18](#)

## — F —

Flow, [393](#)  
Font conventions, [22](#)

---

## — H —

Hardware cluster, 394  
Hardware metrics, 394  
Hardware monitoring, 392  
HDB, 18  
Heavy font, 22

## — I —

Italic font, 22

## — J —

Job configuration file, 28  
Job configuration file statements  
  JOB ENV, 235  
  JOB INFO, 236  
  JOB LOG, 240  
  JOB MODE, 241  
  JOB NOTIFICATION, 244  
  JOB PRIORITY, 249  
  JOB QUOTA, 250  
  JOB RULE, 251  
  JOB TOTAL, 254  
  JOB USER, 255  
  REMOTE COUNT, 263  
  REMOTE COUNT\_MAX, 264  
  REMOTE DATA, 266  
  REMOTE DATA\_MIN, 267  
  REMOTE ENV, 270  
  REMOTE MAX, 272  
  REMOTE MIN, 275  
  REMOTE MINMAX, 277  
JOB ENV, 235  
Job environment file, 28, 52  
Job Importing, 392  
JOB INFO, 236  
JOB LOG, 240  
JOB MODE, 241  
JOB MTFLEX, 242  
JOB MTFLEX\_OVERRIDE, 243  
JOB NOTIFICATION, 244  
JOB PRIORITY, 249  
JOB QUOTA, 250  
JOB RULE, 251  
JOB TOTAL, 254  
JOB USER, 255

## — L —

LAUNCH MAXCOUNT, 258  
LBD, 18  
Licensing  
  requirements, 394

## — M —

MASTER HOST, 225  
Message processing, 314  
Minimum keyword, 23  
Mode of operation, 395

## — N —

nemo::alter, 329  
nemo::application, 330  
nemo::calibre, 332  
nemo::control, 333  
nemo::filesystem, 336  
nemo::host, 338  
nemo::interface, 341  
nemo::job, 342  
nemo::master, 345  
nemo::message, 348  
nemo::node, 350  
nemo::object, 354  
nemo::process, 355  
nemo::remote, 357  
nemo::remotedata, 359  
nemo::snapshot, 362  
Network configuration file, 27, 49  
Network Monitor, 34  
Network requirements, 394

## — P —

Parentheses, 23  
Pipes, 23  
Primary, 18  
Primary host, 393  
proc statement, 311  
Progress meter, 39

## — Q —

Quotation marks, 23

## — R —

RDS HOST, 226  
Remote, 19

---

REMOTE COUNT, [263](#)  
REMOTE COUNT\_MAX, [264](#)  
REMOTE DATA, [266](#)  
REMOTE DATA\_MIN, [267](#)  
REMOTE ENV, [270](#)  
REMOTE HOST, [227](#)  
Remote host, [393](#)  
REMOTE MAX, [272](#)  
REMOTE MIN, [275](#)

— **S** —

SERVER INTERVAL, [219](#)  
SERVER PORT, [220](#)  
SERVER TIMEOUT, [221](#)  
System support, [395](#)

— **T** —

Tcl special characters, [309](#)

— **U** —

Underlined words, [22](#)  
Usage syntax, [22](#)

— **V** —

VARIABLE, [222](#)  
Version support, [395](#)



## **Third-Party Information**

Details on open source and third-party software that may be included with this product are available in the `<your_software_installation_location>/legal` directory.

